tUME

the Universal Map Editor

User's Guide

Dan Chang
Gregg A Tavares

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

What is tUME?

tUME is an acronym for "the Universal Map Editor". While most map editors are throw-away tools built specifically for one project, tUME is designed to be versatile enough for use in a variety of different products. tUME has been used in over fifteen products on various platforms, each with unique requirements.

tUME uses tiles of any size from a single pixel up to 32768 pixels in size (256x128 pixels, 128x256, 327x100, 180x180, or any combination that's less than 32768 pixels). tUME edits rooms of any size from a single tile to 1000 x 1000 tiles, or larger. tUME edits multiple layers in a room. tUME can load thousands of tiles and tile images. tUME handles flipped tiles and various colorsets.

tUME gets the graphics for its tiles from DeluxePaint files. This allows you the most flexibility and power in the creation of graphics for tiles. By using Microsoft® Windows™ or MS-DOS® Task Swapper you may run both tUME and DPaint at the same time and switch rapidly between the two programs.

You may use tUME to define initial object (or sprite) placement for action games. You may use tUME to create collision maps. tUME can help you automate many game development tasks you may currently be doing by hand.

All of tUME's features add up to a more streamlined design process, allowing you more time to adjust game play until your products truly shine!

tUME's universality will prevent it from becoming obsolete. tUME will handle nearly every map oriented project you throw at it, thus you don't have to waste time writing a new tool for every project.

Your First Map

Let's create a simple map using tUME.

1. Create a **tileset** that contains the tile graphics for our map by loading the graphics for the tileset. From the menus, choose **Tiles|Load...|Full Tiled**. A file requester appears. Choose the file SIMPLE.LBM and click **OK**. Another dialog box appears, asking you to specify the tileset type, the tileset number, and the tile size. For now, accept the default values (tileset type = 0, tileset number = 0, size = 16x16) by clicking **OK**. The IFF picture is converted into a 16 x 16 pixel tileset. Note that pressing the left mouse button will select a single tile, and that by pressing and holding the left mouse button you can drag-select a rectangular group of tiles.

2. Now let's create a **room**. Choose **Room|Create...**, and a dialog box will appear. Let's accept the default **User Type** and **User Number**, but let's create a 50x25 tile room. Press [Tab] twice so that the number in the **Room Width** entry box is highlighted, type 50, press [Tab], then type 25 and press [Enter]. A new room will be created. Since no tiles have been pasted into this room yet, it will accept any size tiles. The room tile size will be set when you stamp a tile into the room.

3. Choose **View|Flip Panes** or press [Spacebar] to flip back to the source pane, then drag-select one or more tiles. Now press [Spacebar] again and place your tile(s) by moving the mouse

pointer to the desired location and pressing the left mouse button to place the tiles. Hold down the left mouse button to 'draw' with the tile-brush, or hold down the right mouse button to 'erase' with the tile-brush.

4. Save your map by choosing Project|Save...|Normal. Enter SIMPLE.TUM and click **OK**.

You've done it. You created your first map.

Setting up tUME

WARNING!

There are some design decisions you MUST make before you can start using tUME effectively in your project!

tUME's universality is both a blessing and a curse. It can be a blessing, as you may configure it to work with your particular project requirements. It can be also be a curse, if you pick the wrong model for your project. Since it can be difficult (sometimes impossible) to convert maps from one format to another, it behooves you to choose the right format before you spend hours drawing your maps.

tUME is a power tool. Like other tools, you may hurt yourself (i.e., wasted time) if you don't use it correctly.

Configuration Questions

Here are the questions you want to ask yourself in setting up tUME, and examples of some of the right answers:

1. What sort of maps am I trying to create?

2. What size tile will I use? Please read the **Larger Tiles** section for assistance in selecting a tile size.

3. Should I use the composite tile feature?

4. How do I use the maps saved by tUME in my game? What sort of tUMEPack will I need? Should I use MAP2PIC.EXE, tPBin.EXE, tPMCKid2.EXE, or write my own conversion program?

The Basics

Tiles

A **tile** is the smallest unit of graphics that tUME works with. Tiles are rectangular patches of pixels. Tiles graphic may be literal, where what you see is what will appear in the level, or iconic, where it may represent a monster, an attribute, or something else entirely.

A tile may be any size; e.g., 8x8, 16x16, etc. Tile sizes are typically the same size as the target hardware's character size. Thus if the target hardware supports 8x8 pixel characters, it would be reasonable to choose a tile size of 8x8 or 16x16.

A collection of tiles is called a **tileset**. Tilesets appear in **source rooms**.

NULL Tiles

When a **room** is first created it is filled with NULL tiles. These NULL tiles represent the absence of a tile. They do not represent SKY, GROUND, nor anything else except the absence of a tile. When you are creating a room remember this because it will affect your game. If you are making a platform video game and you have NULL tiles in your rooms, when tUMEPack generates your game levels it needs to put a tile everywhere there is a NULL tile and it has no idea of which tile it should put there. Should it be tile 1 from tileset 1, or tile 1 from tileset 17, or tile 19 from tileset 4? It doesn't know so it is best to fill every NULL tile with some tile. You can check for NULL tiles by pressing [Ctrl]-[or [Ctrl]-] on the keyboard to change the NULL tile color. Anywhere you see the colors change is where there is a NULL tile. Press [Ctrl]-\ on the keypad to set the color back to normal.

Tilesets

A **tileset** is created whenever you load a DPaint picture into tUME. Every time you load some tiles into tUME from a DPaint picture you create a new tileset. tUME does not group tiles from different pictures into one tileset.

To create a tileset, choose one of the following five methods: **Load...|Full Tiled**, **Load|Tiled-Blanks**, **Load|All Tiled**, **Load|As Brushes** and **Load|Boxed** (all which see, below). If you can't decide between which of the five loading methods to use, I recommend using **All Tiled** as it is not sensitive to the background color, and is more forgiving about adding tiles to the source picture.

To delete a tileset, choose **Tiles|Delete...**.

Renaming Tilesets, Replacing Tilesets

Occasionally (rarely), you may want to tell tUME to use a different ILBM file for a tileset. There are two ways to change a tileset name:

1. A. Create the replacement tileset. B. Delete or move to some other directory (hide) the old tileset. C. Load your map. When tUME can't find the old tileset, it will ask "Can't load tiles! Would you like to try a different file?" D. Click **Yes**, and use the file requester to specify the

replacement tileset name. E. Save your map.

2. A. Create the replacement tileset. B. Keep the old tileset around. C. Load your map. D. Select a tile from the old tileset. E. Select **Tiles|Set Info...** and change the filename to the replacement tileset. F. Save your map.

Organizing Tileset on Disk

When you load a tileset in tUME, all that is really added to your map is the name and path of the picture you specified. That means that if you tell tUME to load the picture c:\game\art\tree.lbm, the next to you load your map tUME is going to look for tree.lbm in the sub-directory c:\game\art, and if it can't find it there it will complain. It will ask you to tell it where it can find tree.lbm, and once you've told it, it will ask you if you want it to look for other tileset in the directory you just specified to find tree.lbm. The implication is this: if you plan to share maps and tilesets with others, it would be easiest to create the same directory structure on both machines for storing your maps and tilesets. Alternatively, if you plan to store your maps and tilesets in the same sub-directory, you can set tUME to search the current sub-directory for tilesets; see **Configuring Tileset Search Path** section of the *tUME Configuration Guide*.

Tileset Display Colors

A tileset is normally displayed using the palette found in its DPaint picture. However, there is an option to display the source tiles using the current edit room's palette. Check **View|❑EditColorsOnly** or press [Alt]-E, and the source room will be displayed using the last edit room's palette. This feature is useful when you have a tileset that is designed to use different palettes to change its appearance.

E.g., lets say you have a marble tileset that's designed to appear as green marble in the OFFICE edit room, and as grey marble in the HALL edit room. Switch to the OFFICE edit room, and choose **View|❑** Now when you switch to the marble tileset, the tiles will appear green. Switch to the HALL edit room, and then when you switch to the marble tileset, the tiles will appear grey. Uncheck **View|❑EditColorsOnly** again, and the marble tileset will be displayed using its DPaint palette.

Note that if you select this option while in a source room, it will not take effect until you switch to an edit room.

Rooms

A **room** is made of one or more **layers** (described below) stacked on top of each other. tUME has two basic types of rooms, a **source room** and an **edit room**. A source room contains one or more tilesets, as described above. A **composite room**, described below, is also considered a source room.

An edit room is a room that you create. An edit room may be a **level** (as in a game level) **room** and represent a single level in a video game, or it may be a **conversion room** (which see), and represent tile attributes definitions.

To create a room, choose **Room|Create...** To delete a room, choose **Room|Delete...** To lock a

room, choose **Room|Lock**. To clear a room, choose either **Room|Clear...|Tiles** or **Room|Clear...|Complete**. To re-size a room, choose **Room|Set Info...** and then enter the new **Room Width** and the new **Room Height**.

tUME allows as many rooms as will fit in memory. Thus, you may wish to create **parts rooms**; see **Cookbook: How do I make frequently used tile-brushes easily accessible?**

Moving about a Room

If your room is larger than can be displayed on the screen you can press the cursor keys to scroll around. Pressing [Ctrl] and the cursor keys will scroll around the room quicker. Pressing n will scroll the tile under the cursor as close to the center of the screen as possible. Pressing [Ctrl]-[Home] shows the upper-left corner of the room, and pressing [Ctrl]-[End] shows the lower-right corner of the room.

tUME will scroll automatically if the mouse pointer touches the edge of the window while you are drawing tiles or picking up a tile-brush. This allows you to draw over large areas and also permits you to pick up tile-brushes larger than the display window.

Room Display Options

tUME allows you zoom in, to examine each tile in more detail, or to zoom out, to see more of your map at once.

Choose **View|Zoom|Zoom Out** or press < ([Shift]-,) to zoom out, or choose **View|Zoom|Zoom In** or press > ([Shift]-.) to zoom in. Pressing < will cause the display to zoom out more with each press. To zoom out to the limit, press [Alt]-<, and to zoom in to the limit, press [Alt]->. To return the display back to normal, Choose **View|Zoom|❑Toggle Zoom** or press m. Pressing m repeatedly will toggle the display between the normal view and the last zoom setting.

The maximum zoom out, maximum zoom in, and all zoom settings are all configurable through the tUME.INI file.

You may disable the display of priorities by checking **View|Disable|❑Priority**. You may disable the display of tile flipping by checking **View|Disable|❑Flip**. You may disable the display of tile colorsets by checking **View|Disable|❑Colorsets**.

Room Colors

Each room has its own color palette. It gets this palette from the very first tile you stamp into a new room.

This means that if you create a bunch of blue-shaded tiles, and then you create a room using those tiles, you will have blue tiles in your source room and your edit room.

Now if you save your map, and then use DPaint's palette editor to change the blue shades to shades of green, when you reload your map, tUME will re-scan in your tiles from your new DPaint picture, and the tiles will be display in the source room using green shades. However, when you look at your previously created edit room it will still display using the original blue shades. One way to solve this problem is to copy a new palette to that edit room.

Copying Rooms Colors

To copy the colors of one room to another, go to the room you want to copy the colors from. Pick up a tile-brush. Change to the room you want to copy the colors to and choose **Room|Copy Color|All**. (see **Room Colors** above).

There are three **Copy Color** options, **All**, **Palette**, **Color Cycles** (all which see, below).

To copy only part of the palette from one room to another, see the instructions under **Colors**, **Color Sequencer Buttons**, **COPY**.

Moving Between Rooms

To move forward to the next edit room press 1; to move backward to the previous edit room, press 2.

To flip to the source pane, press [Spacebar]. To move forward to the next source room, press 1; to move backward to the previous source room, press 2.

Layers

A **layer** is a rectangular grid of tiles. Layers don't exist on their own, they are always part of a room. Currently, all the layers in a room use the same size tiles. The first layer in a room is layer one. Layer two is then drawn on top of layer one, and layer three on top of layer two, etc.

Floor

The current edit layer is called the **floor**. The floor affects what you get when you pick-up a **tile-brush** (below) and when you stamp a tile-brush. To move the floor up one layer, press [Alt]-. To move the floor down one layer, press [Alt]-↓. To move the floor to the topmost layer press [Alt]-[Shift]-. To move the floor to the bottommost layer press [Alt]-[Shift]-↓. Each room has its own floor layer.

If you want to see and edit only the floor layer, choose **Layer|❑EditOnlyFloor**.

If you choose **Project|Show Status|Room Info** or **Project|Show Status|Coordinates**, then the rightmost number is the current floor layer, and the number immediately left is the total number of layers in the room.

To add a layer to the current room choose **Layer|Add**. To insert a new layer at the floor layer (see **The Floor**, below) of the current room, choose **Layer|Insert**. To delete a layer from the current room, move the floor to the layer to be deleted and choose **Layer|Delete...** To save a layer from the current room, move the floor to the layer to be saved and choose **Layer|Save...** To load a saved layer and insert it as the current floor layer (see **The Floor**, below) of the current room, choose **Layer|Load...** To load a saved layer and append it as the topmost layer of the current room, choose **Layer|Append...**

Layer|Lock, Unlock, Visible, Invisible

You may lock a layer or make it invisible. The most useful combination is probably locked and invisible obtained by choosing **Layer|Invis+Lock**. This option makes the current floor layer locked (uneditable) and invisible (you can't see it).

E.g., if you wanted to edit layers 01 and 02 of your room and didn't want to affect layer 03 you would move the floor to layer 03 and press i making it locked and invisible. Now move the floor to layer 01 and edit away. You may set each layer's locked/unlocked and visible/invisible status separately. The floor's current locked/unlocked, visible/invisible status is indicated in the status bar by the last two characters: **v** = visible, **i** = invisible, **u** = unlocked, **l** = locked.

Tile-Brush

Editing a room in tUME is accomplished via the tile-brush feature. By using the tile-brush tool, you can pick-up a rectangular region of tiles, and carry the tile-brush around. Then you can make a copy of it (stamp it) somewhere else in the edit room, or even in a different edit room.

You can read more about tile-brushes in the **Editing Layers** and **Stratify Paste** sections, below.

Selecting a Tile-Brush

Press b to activate the tile-brush selection tool. The cursor will change to a cross-hair to indicate that you are in the tile-brush select mode. Drag-select the tiles you want to pick up in your tile-brush.

When you select a tile-brush, you get the floor layer and every layer above it. Thus, if you have three layers, and the floor is set to layer 02, then you would be grabbing two layers of tiles, layers 02 and 03.

If you've picked up a multi-layer tile-brush using the b key, pressing <SHIFT>-X (or choosing **Brush|Strip Brush**) will strip it of all but the bottom layer.

Alternatively, you may press v (or choosing **Brush|Select Plane**) to activate the single layer tile-brush selection tool. This tool will pickup only a single layer of tiles from the floor.

You may flip the tiles in the tile-brush about the x-axis by pressing x; you may flip the tiles in the tile-brush about the y-axis by pressing y.

See also **Stratify Paste** section, below.

Pasting the Tile-Brush

After drag-selecting a tile-brush, the cursor should change back to a pointer, which indicates that you may stamp the tile-brush. If it remains a cross-hair, it means you cannot stamp the tile-brush in the current room.

Press the left mouse button to stamp a copy of the tile-brush at the current location in the room. Press the right mouse button to erase (set tiles to NULL tiles).

When you change the floor you change the destination of your tile-brush. E.g., if you set the floor to layer 02 and picked up a two layer tile-brush with tiles in from layers 02 and 03 and then you moved the floor to layer 01, the tiles you picked up in layer 02 will be drawn in layer 01 and the tiles you picked up in layer 03 will be drawn in layer 02. Layer 03 will not be affected because your tile-brush is only 2 layers deep.

You may UNDO the last tiles you stamped by choosing **Brush|Undo** or by pressing u.

There are two tile-brush pasting modes; choose either **Brush|❍Paint** or **Brush|❍Replace**

(which see).

You may hide the tile-brush by un-checking **Brush|❑Show Tile-Brush**, which see.

Windows

Currently, tUME only displays one **window**, which occupies the screen area below the **status bar**. Each window has two **panes**.

Status Bar

At the top of the screen is the status bar. The status bar may be revealed or hidden by choosing **Project|❑TitleBar** or pressing [F10]. Change the status bard display by choosing **Project|Show Status|Room Info**, **Project|Show Status|User Info**, **Project|Show Status|Tiles**, or **Project|Show Status|Coordinates**.

tUME uses color registers 254 and 255 to draw its menus and color registers 252 and 253 for some windows. If you set those colors all to white or all to some other color, you will not be able to read the status bar. tUME will automatically change the colors of the status bar when you move the mouse pointer over it, but that is not usually very useful. You may tell tUME to force the last four colors to something visible by pressing [Alt]-P. Pressing [Alt]-P again will return your original colors.

Panes

Only one pane is visible at a time. To view the other pane, choose **View|Flip Panes** or press [Spacebar] or the j key. By default, all source rooms are in a list that appears in the source pane, and all edit rooms are in a separate list that appears in the edit pane. You may modify what the panes display by checking **View|Pane|❍Only Source** or **View|Pane|❍Only Edit** or **View|Pane|❍Allow All**.

Palette Requester

You can modify the color palette of any room in tUME by going to that room and pressing p or choosing **Room|Set Palette** to bring up the palette requester (also called the color sequencer):

µ §

At the top of the Color Sequencer you will see all 256 color displayed. You may choose one by selecting it or more than one color by drag-selecting them. You can modify your selected colors by dragging the small diamonds underneath the **R G B** button. Clicking on the **R G B** button with change to HSV mode so that you can modify the colors using the Hue, Saturation, Value method.

Color Cycle Example

tUME supports a very sophisticated color cycling system. Lets say you wanted one color in your palette to glow in a reddish color. (The Arcade game *Xevious* did this to make the explosion craters glow.) Click on the grey checkered box inside the area above the word **CYCLES**. Now click on **INSERT**. You will notice **00** appearing in a white box. Click on it to select it. An arrow will appear above it indicating you are editing cycle **00**.

Now click on the grey checkered box in the **CYCLE REGS** area. Click on **INSERT** and you will see a black box appear and the grey checkered box will move to the right. Notice that the number **0** is printed below the black box. This number **0** represents color REGISTER number **0**.

Now click on the grey checkered box in the **CYCLE COLORS** area and then click on **INSERT** 16 times. Note the number just left of **CYCLE COLORS** should now say 16 and the grey checkered box has scrolled off the display.

Next click on the far left inside the **CYCLE COLORS** area. An arrow should appear above the area you just clicked. Now move the **R** slider up in the **R G B** area to the top. The area just below the arrow should now be red. Click on this new red area and DRAG select to the right until the entire 16 **CYCLE COLORS** are selected. Click on **SPREAD**. You should now have a spread of colors from red to black.

Finally click on the **SPEED** slider. Hold down the mouse button drag the speed slider around. Do you see what is happening? The 16 shades of red you just created are being 'pumped' through color REGISTER 0 (zero).

Click on color register 0 in the **COLOR REGS** area and then move one of the RGB sliders. Go back and drag this **SPEED** slider around. Now the shades of red are being 'pumped' through a different color register.

Click on the color register in the **COLOR REGS** area and the click on **INSERT** 7 more times. All the new color registers you add will start as color register 0. Drag select any 8 colors from the **PALETTE** area. Click on **COPY** and the click on the first color register in the **COLOR REGS** area. Again, drag the **SPEED** slider and note how the shades of red are now being 'pumped' through the 8 registers you copied.

You can have as many cycle colors as you want 'pumped' through as many color registers as you

want. You can also **INSERT** more cycles and 'pump' a different set of cycle colors through a different set of color registers.

Color Sequencer Controls

Palette Area
This area shows all 256 colors. You can select one or more by drag selecting the colors you want. The colors are numbered down then across meaning color #0 is in the top left corner. Just below color #0 is color #1, then color #2 ... To the right of color #0 is color #8.

RGB/HSV Sliders
Here you can click on the **R G B** button to switch to HSV mode (Hue, Saturation, Value) and you can drag the sliders to effect the currently selected color or colors. The arrows above and below the sliders allow you to affect the selected colors with a little more detail.

Cycle Regs Area
This area shows the color registers of the currently selected Cycle. The numbers in the bottom of this area represent the color registers you've defined for the current cycle. The colors above the numbers are just the color those registers are currently set to above in the palette.

When this area is active (because you click on one of the color regs.) an arrow will appear above the color register you clicked on. You may drag select more than one color. Using the **R G B** sliders you can change the currently selected color registers. If you have more than 16 color registers defined you can scroll through them using the left and right arrow buttons. The number at the right of this area is the total number of color registers you have defined for this Cycle.

Cycle Colors Area
This area works almost exactly the same as the **CYCLE REGS** area except you are manipulating cycle colors instead of color registers.

Cycle Area
This area shows the Cycles you have created. You can click on any one of them and the **CYCLE COLORS** and **CYCLE REGS** areas will show you the cycle colors and color registers for the selected Cycle. When this area is inactive you will see a small pointer indicating which Cycle is currently being edited.

Speed
This sets the speed of the current Cycle. All the way to the left is OFF and extreme right is 60 times a second.

Color Sequencer Buttons

CYCLE ON/CYCLE OFF
This flag sets whether or not the selected cycles are on or off.

⇿(DIRECTION)
This sets the direction the cycle colors are moved through as they get 'pumped' into the color registers. NOTE: The cycle colors are always put into the color registers in the same order. This button just sets the direction of the Cycle. Example: The cycle colors are red, yellow, green, cyan, blue, purple. The first time the colors are placed in the color registers the will be placed in the preceding order. The next time the order will be yellow, green, cyan, blue, purple, red. If you change the direction, the order would be purple, red, yellow, green, cyan, blue.

INSERT (keyboard equivalent: i)
This button inserts a color register, cycle color or Cycle depending on which area is active. The active area has an arrow above it pointing down.

DELETE (keyboard equivalent: d)
This button deletes the currently selected color registers, cycle colors or Cycles.

COPY (keyboard equivalent: c)
This button copies colors or color registers. Here are the steps involved in copying color(s):

1. Drag-select the colors you want to copy.
2. Select **COPY**. The palette requester's title bar should read **COPY TO->**.
3. Move the mouse pointer to where you want to paste the colors, and press the left mouse button.

To cancel the copy operation after step two, press [Spacebar].

You can copy from the palette area to the palette, the color regs to the color regs, the color cycles to the color cycles. You can also copy from the palette to the cycles color or from the cycle colors to the palette.

You may also copy colors from one room's palette to another room's palette. Follow the instructions above, but insert these following steps between steps 2 and 3:

2.a. Select **OK** to leave the current room's palette.
2.b. Make the destination room visible (press [Spacebar], 1 or 2 until it becomes visible), then press p or choose **Room|Set Palette** to edit the destination room's palette. Note that the palette requester's title bar reads **COPY TO->**; this indicates your are in the middle of a copy operation. Copying to and from the color registers is another matter. If you copy from the palette to the cycle registers then the cycle registers will be set to the register numbers of the copied colors. It makes no sense to copy from the cycle colors to the cycle registers since the cycle colors do not represent any color registers. Copying from the cycle registers to the palette or the cycle colors copies the associated colors in the palette. Example: the cycle register says 23. This would mean color 23 in the palette would get copied to the destination of the copy.

SWAP (keyboard equivalent: s)
Swap works exactly the same as copy except the colors are swapped between the source and destination. The same rules apply as when copying to and from the cycle registers.

SPREAD
Spread spreads the colors from the first selected color to the last selected color. Spreading in RGB mode spreads using the red, green, and blue values. Spreading in HSV mode uses the hue, saturation and value values. This means you'll get different results depending on which mode you're in.

You can also spread the color registers which will spread the register numbers between the first and last selected registers.

BLEND (keyboard equivalent: b)
Blend is like copy except the destination colors are blended with source colors. You cannot blend color registers.

CANCEL (keyboard equivalent: [Esc])
This exits the Color Palette Editor, discards all the changes you've made.

OK (keyboard equivalent: [Enter])
This exits the Color Palette Editor, keeping all the changes you've made.

Weirdness in the Color Palette

When you insert cycle colors you will notice colors disappear from the palette. This is because the cycle colors are independent of the palette meaning that to display both the palette colors and the cycle colors we would need to show 256 + 16 colors or 272 colors. The IBM in MCGA mode can't show 272 colors at once so instead we use colors 232 through 247 to show the cycle colors. The palette colors are shown when the palette area it active and the cycle colors are shown when the **CYCLE REGS**, **CYCLE COLORS** or **CYCLES** areas are active.

Colors Menu

When the palette requester is active the menus change to allow you to save and load palettes.

Load...
Choose this option to load a palette stored in an IFF file into the currently active palette. You may not attach a key to this event. This event only works when the palette requester is active.

Save...
Choose this option to save the currently active palette to an IFF file. You may not attach a key to this event. This event only works when the palette requester is active.

Range Menu

When the palette requester is active the menus change to allow you to save and load palette ranges.

Load...

Choose this option to load a saved color cycling range stored in an IFF file into the currently selected palette range. You may not attach a key to this event. This event only works when the palette requester is active.

After selecting the file to load, a dialog will be presented, requesting you to specify the first color to copy. This defaults to the first color of the first color cycle range in the palette. Press [Enter] to accept this default value, or enter a new first paletter number. The number of colors loaded will be the number of colors selected in the palette.

Save...

Choose this option to save the currently selected palette range to an IFF file as a color cycling range. You may not attach a key to this event. This event only works when the palette requester is active.

The first color and number of colors saved in the color cycle range is based on the currently selected palette range.

Print Map Requester

tUME can print your rooms and it can print them in various sizes from one page to very large multi-page poster size printouts. Choose **Room|Print...**, and you'll see this dialog box:

To print, first decide how you large you would like your printout to be. You only specify one size parameter, either across or down, and tUME computes the other. Choose how you would like to specify the print size.

For example, choose **INCHES** and then click on the **ACROSS** number, type **10** and press [Enter] (not [Tab]!). You will notice that tUME fills in the **DOWN** number. What is happening is that tUME attempts to setup printing so that your room will scaled to 10 inches across when you print and then it computes the height the printout will need to be. For example if you room is twice as high as it is wide then if you specify 10 inches across your room is going to print 20 inches high. If you specify 10 inches down then your room is going to print 5 inches across.

tUME currently only prints in PCL 3 which means it will print on an HP Deskjet, Laserjet or compatible printer. You may tell tUME to print on a different printer or to a file by changing the tUME.INI file. See the **Configuring Printing** section of the *tUME Configuration Guide*.

While tUME is printing you may press [Esc] to abort the printout. NOTE: tUME will recover 'gracefully' by ejecting the page currently being printed and resetting your printer to standard defaults.

What is printed

tUME prints the current room in whatever state it is currently in. This means that if you have guides on they will be printed. If you have your tiles 'spaced' then spaces will be printed. If you have certain layers invisible they will not be printed.

Also, tUME prints in the current colors of the room so if you want a good printout you should probably set the background color in the palette to WHITE because most printers don't print lots of black very well.

If you print with spaced tiles, or if you have many NULL tiles, you may want to change the background color before you print. Press [Ctrl]-] to increment or [Ctrl]-[to decrement the background color.

You may want to turn off certain layers like special layers or icon layers before you print so your special tiles won't obstruct your printout. Move the floor to that layer and press i.

SPECIFY SIZE BY

These settings allow you to specify the size you want to print your room in one of four ways.

Pages

Specifying size by pages does just that. You tell tUME how many pages across or down you want your room printed.

Inches
Specifying size by inches allows you to print your room smaller than 1 page. You can specify in 100ths of an inch so you could type **5.45** inches if you want to. Note: tUME assumes a page in 10 inches across and 7.5 inches down. This leaves at least a half inch border around the each page (in which most laser printers cannot print.)

Tiles Across Page
Specifying this way you tell tUME how many tiles you want to fit per page either across or down.

Tile Width In Inches
This last way you tell tUME how large a particular tile is. If you tell tUME to print each tile at 1 inch across and your room is 500 tiles wide then tUME is going to print your room 500 inches across or at least 50 pages!

Shade Method

Choosing a different shading method instructs tUME how to determine which shade of gray a particular color should be printed in. tUME tries to simulate 64 shades of gray.

RGB Average
The red, green and blue values for a particular color are averaged

V*2+S
The 'Value' of a color is multiplied by 2 and then the color's Saturation is subtracted.

Color Correct
The red, green and blue values are each multiplied by special constants where green has the highest constant and blue has the lowest and then the results are added together. This produces shades the correspond closer to what you see on the display since green is the brightest color and blue is the darkest. This is therefore the default shade method.

Resolution

Choose 150 dots per inch or 300. 300 will look better but if your printer doesn't have enough memory for an entire 300 dot per inch page then you'll have to choose 150 dots per inch.

First Page Only

This is mostly for testing. If you are about to print a very large printout you can check this option so that only the first page is printed. This way you can check the first page to see if it is printing that way you want it to before you decide to print the entire room.

Use Light Shades

This option 'ramps' the gray scale calculations so that more light shades are used to print your

map. This results in much better looking printouts and is therefore checked by default.

Advanced tUME

Colorsets

tUME allows up to 128 colorsets. What does that mean and why do you need 128 of them? Well you probably only need 8 for today's hardware platforms. The Super Nintendo Entertainment System or SNES only allows 8 colorsets of 16 colors each. The SEGA Genesis allows only 4 colorsets of 16 color each.

Colorsets are a way of displaying and using your tiles in more than one set of colors. The default setup is for the SNES and Genesis but it may be re-configured for your particular needs (see the **Redefining Colorsets** section in the *tUME Configuration Guide*).

To use the default 8 colorsets you need to draw your tiles in a particular way. Each tile needs to use one set of 16 colors in DPaint. Either the first set of 16 colors (0 - 15) or the second set of 16 colors (16 - 31) or the third set of 16 colors (32 - 47) and on to the eighth set of 16 colors (112 - 128).

When you make your rooms in tUME using these tiles they will be drawn in the color you used in DPaint. If you want to change them to one of the other colorsets, first pick up a tile-brush, then press 1, 2, 3 through 8 on the numeric keypad to choose colorsets 1 through 8 (or 0 through 7 for you programmer types). The word **COLOR** will appear attached to the cursor, and now when you stamp a brush with the left mouse button, instead of actually placing tiles in your room, you will be changing the colorsets of ("coloring") the tiles you stamp over. To change the tiles back to their original colors, stamp with the right mouse button.

Press . (period) on the keypad to exit the colorset coloring mode and return to the regular tUME drawing mode (where you place tiles instead of just changing their colors).

Priorities

Priorities are something that is really only used on the SNES/Genesis systems, though you may find other uses for it. On those systems individual tiles may be marked as having a higher priority so that the system's sprites will appear to go behind those tiles. Editing or setting the priorities of tiles works exactly like the colorsets.

Grab a tile-brush and then press + on the numeric keypad. The word **COLOR** will appear attached to the cursor, and now when you stamp with the left mouse button, you will set the priority bit of all the tiles you stamp on to 1. If you stamp with the right mouse button, you will reset the priority bit of the tiles you stamp on back to 0.

Tiles that have there priority set to 1 will be draw in colors 128 through 255. For example if your tile uses colors 16..31 and you set its priority to 1 then it will now be drawn in colors 144..159 (16+128..31+128). It is up to you to set these colors to something you will recognize. You may want to set them to that same colors as the originals except with a lighter or darker shade, or you may want to set them all to shades of red or shades of grey. Choose anything that will make it clear which tiles have their priority set to 1.

Press . (period) on the keypad to exit priority setting mode and return to the regular tUME

drawing mode (where you place tiles instead of just changing their priorities).

Using Larger Tiles

Sometimes, you might want to use tiles that are larger than the hardware character size. Examples abound from existing products: *Super Mario Bros 2* and *3* on the NES uses 16x16 pixel tiles. *Robin Hood* for the NES uses tile that are almost a full screen in size.

Why would you want to do this? Because it allows you to create very large levels with very little memory. Think of it this way. If you are writing an NES game like Mario and the average level is 10 screens by 2 screens in total, one way you could implement the game is to store 20 screen dumps using one BYTE to represent each NES 8x8 character on the screen. Since one NES screen has 32 characters across and 30 character down that would require 32*30 bytes (960 bytes) per screen. Multiply that times 20 screens and you get 19200 bytes per level.

Since your typical NES cartridge is only 256K, you would only be able to store about 12 levels. Not very many! If instead you used a larger tile size, say 16x16 pixels, each 20 screen level would now only require 4800 bytes. You could go even farther if you wanted to. I believe *Sonic* uses 32x32 pixel tiles, which means *Sonic* can store 80 screen levels in only about 5K bytes.

If you want to use 16x16 pixel tiles, the simplest way is to use DPaint to draw 16x16 tiles. However, if you prefer to draw 8x8 tiles, and use tUME to create 16x16 tiles from the 8x8 tiles, then you'll want to create composite tiles.

Why Use Composite Tiles?

Composite tiles offer a way to create tiles that are made of a rectangular grid of smaller tiles. Thus you could make 16x16 pixel composite tiles from four 8x8 pixel tiles, or make 32x32 pixel composite tiles out of four 16x16 pixel tiles, or make 64x64 pixel tiles out of 64 8x8 pixel tiles.

This is a preference issue. For example, if you wanted to use 64x64 pixel tiles, you may simply create 64x64 pixel tiles in DPaint, create your map using the 64x64 tiles, and rely on tUMEPack on the back end to break the larger tiles down to smaller tiles (and find the duplicate tiles). tPMCKid2 does this: it is designed to work with 16x16 tiles, and will break each tile down to four 8x8 component tiles. This is the recommended technique.

On the other hand, you could use composite tiles. In this case, you would want to draw the component 8x8 tiles in DPaint. Load the 8x8 tiles into tUME, then use these 8x8 tiles to construct a composite 64x64 pixel tile. Note that the composite tiles could be edited to a limited extent without returning to DPaint.

Composite tiles becomes more useful if you wanted to generate very large tiles, say, for example 256x256 pixels in size. At this size, it may become more convenient to design the large tiles in tUME instead of DPaint.

To create composite tiles, choose **Tiles|GridRoomAsTiles...**.

Editing Composite Tiles

After you have created a composite tileset, you may redefine which tiles make up the composite tiles. To make changes to the composite tileset, choose **Room|Lock** to unlock the composite

room. The status bar will change from **CL** to **CU**. When the composite room is unlocked you may edit the composite tileset, and change the tiles that make up each composite tile. When you are finished changing the composite tiles, choose **Room|Lock** to lock the composite room again.

Thus, to continue our above example, while the composite tileset remains locked, drag-selecting will pick up a tile-brush made of 2x2 8x8 pixel composite tiles. While the composite tileset is unlocked, drag-selecting will pick up a tile-brush made of 8x8 tiles, and you can then stamp the 8x8 pixel tile-brush into the composite tileset.

Room User Types

All rooms can have a name, a type and a number. These are referred to as the Room Name, Room User Type and Room User Number. All of these have no significance or meaning except the meanings you give them for your project (which is usually the meaning assigned by your particular version of tUMEPack). To set or change the name, type or number of the current room, choose **Room|Set Info...** and enter the data you want to enter. Note that it is usually a good idea to give your rooms names because since the name is shown in the status bar it will help you to know which room you are looking at it if the imagery is unfamiliar.

Tile User Types

Tiles can also have types and numbers. As with rooms these types and numbers have only the meaning you assign to them. (see tUMEPack). To set the User Type and User Number of a tileset, first make sure you are on a Source Pane (looking at tilesets and not regular rooms). Now click on a tile that belongs to the tileset you want to give a User Type and User Number to, then choose **Tiles|Set Info...** and enter your numbers. To see what numbers have previously been assigned to a tileset use the 'User Info' mode of the status bar. (see status bar above)

Conversion Rooms

An idea we came up with when we first developed tUME was the idea of conversion rooms. A Conversion room is a room that is used to generate a table or tables for your game. Here are two example conversion rooms:

TANK Explosions

In one of our very first games, an arcade type tank combat game, we built levels out of tiles and we wanted to be able to leave explosion marks on the play field. The explosions were six tiles large, 2 tiles wide and 3 tiles tall. If an explosion happened on a particular tile we needed to know what kind of tile to change it to (i.e. a road tile would change to a road tile with an explosion crater on it). We also needed to know what tile to change the tile directly above the exploding tile to (i.e. a grass tile to a grass tile with vertical scorch marks). Then there was the tile up and to the left, the tile directly left, the tile down and to the left and the tile down or below the exploding tile. This would require one heck of a large table and would have been an incredible pain to do by hand in assembly language or C so instead of typing the table in by hand we used a Conversion room.

This conversion room was about 16 tiles wide, 16 tiles high and 7 layers deep. On the first layer

our artists placed all the tiles in the game. On the second layer he placed the tiles that the first layer tiles would change into if an explosion happened down and to the right of them. The third layer said what tile to change into if an explosion happened one tile below them. The fourth, fifth, sixth and seventh layers were the other 4 cases; to the right, direct hit, up and to the right, directly above. (See TANK.MAP.)

M.C. Kids Alt/Collide

In M.C.Kids we had over 2688 different tile images but we had only 128 different tile types so the artist and I decided to use a conversion room. This conversion room was 16 by 512 tiles large and we decided to arrange it into strips of 3. By this I mean that the first row, row zero, was filled with actual image tiles used in the game. The second row, row one, was filled with what we called, collision tiles, these tiles looked like icons and were not actually used in the game. Instead they represented tile types. Types like, solid ground, slippery ground, M's, Cards, 1Ups, Sloped Right, Sloped Left, Spinners and on and on. If an image tile in the first row had a spinner collision tile under it in the second row then that image tile was a spinner in the game. The third row, row two, was filled with alternate image tiles. These were associated to the first row tiles in that they showed what the first row tiles could change into under certain circumstances. (i.e., a M tile would change into a sky tile when the M was collected.) See MCKIDS.MAP.

The version of tUMEPack for M.C. Kids read this room and created tables where by I, as a programmer, could tell what type of tiles the M.C. Kids are standing on or bouncing on or ...

Flat/Layered/Stripped/Why?

Conversion rooms can be created any way you like; it's all a matter of writing a version of tUMEPack that will create conversion tables from your conversion rooms.

Why do you want to use conversion rooms? Because they allow more flexibility. Some programmers will hard code their tile types: Tile 0 is sky, Tile 1 is ground, Tile 2 is death, etc. We prefer to encode such information in a conversion room so that it may easily be changed. Using conversion rooms allows the artists and the game designers to make fairly major changes to the game without programmer intervention. This means they can go on an create an awesome game without having to come badger you to make nearly as many changes or special programming considerations as you might otherwise. It also puts a lot of the work in there court with should mean less work for you.

In the past we've implemented many versions of tUMEPack to interpret Conversion rooms in a particular way.

Layered

This was the method describe above under 'Tank Explosions.' Using this method, conversion tables are created to associate tiles on layer 1 with tiles on layers 2, 3, 4 ...., one table for each layer above layer 1.

Flat

This method generates the same kind of data as the Layered type but instead of using layers we use columns or rows. Column 0 would be filled with the 'original' tiles and columns 1, 2, 3, ...

would be filled with the tiles you want to associate with the tiles in column 0.

Stripped

This is the method described above under 'M.C. Kids Alt/Collide' Instead of the strips being 3 rows tall they could have been 2 rows, 4 rows or what ever we or you need.

Custom

Design your own. We will be happy to write a version of tUMEPack that will interpret a conversion room in whatever form is convenient for you.

Collision Maps

There are other ways to associate a type, collision or attribute with a tile. For example drawing your level on the first layer of a room and your collision map on the second layer using iconic tiles. Load the map **3LAYER.MAP** and look at the second layer. Notice that this layer just defines the collision map of the level. You could store both maps in your game or you could have tUMEPack create a conversion table for you. (See TPGEN.)

Object Layers

Another thing you can do with tUME is to create a layer and in that layer place objects (or sprite objects). Load the map **3LAYER.MAP** and look at the third layer. It is filled with iconic tiles that represent various sprite objects that appear in the level. Things like gophers, spiders, moving platforms, boats, zippers and other objects. The M.C. Kids version of tUMEPack would take that layer and create a table of objects where each entry in the table had four elements, X Position, Y Position, Type of Object (i.e. Gopher, Spider, Boat), Y Index. (See TPNES.)

Downloading to a Development System

tUME can support downloading the current layer of the current room to a development system from within the program, allowing you to quickly see how the graphics and colors will appear on your target system.

Currently, the program supports downloading to a Chip Level Designs connected SNES or Western Technologies SegaDev connected Genesis. On the SNES, the downloader currently supports mode 2 (8x8 pixel characters, 8 palettes of 16 colors each) and mode 3 (256-color 8x8 pixel characters). On the Genesis, the downloader supports 40 column mode.

To download the current room, first make sure that the SNES or Genesis is connected to the PC, and that tUME map downloader program is running on the SNES (or Genesis). The easiest way to load the SNES or Genesis with the correct program is to type CLDTUME (for CLD SNES) or SDTUME (for SegaDev) at the DOS prompt. If the program loads successfully, you should see a blue screen on the SNES with the following words on screen:

tUME Map Scroller

Copyright (c) 1992 Echidna

Ready.

Next, make sure that the tiles in the room you want to download are some multiple of 8x8, such as 8x8, 8x16, or 16x16. Choose **Layer|Download|16-Color Chars** to download the current layer of the current room, or choose **Layer|Download|256-Color Chars** to download a 256-color characters. Both options perform the same function on a SegaDev connected Genesis.

By default, tUME only downloads only one SNES screen worth of graphics, by you can download the entire room by choosing **Layer|Download|One Screen** to toggle off the one screen mode.

Once you have a map downloaded, if the live palette feature is active (choose **Room|Live Palette**), adjusting colors in the palette editor (press p or choose **Room|Set Palette**) will also adjust colors on the SNES or Genesis.

Cookbook

How do I make frequently used tile-brushes easily accessible?

The easiest way is to create a locked **parts room** that holds a copy of frequently used brushes. Then you can switch between your edit room and your locked 'parts' room by simply pressing [Spacebar]. Create a small room with as many layers as needed to hold your tile-brushes. Stamp your tile-brushes into this 'parts' room. Choose **Room|Lock** to make this 'parts' room appear in the source pane. Now press [Spacebar] to switch to the source pane, and then press 1 or 2 repeatedly until you find your 'parts' room. At this point, subsequent presses of the [Spacebar] will switch between your edit room and the 'parts' room.

To help you select your brushes, you may wish to place the upper-left corner of each tile-brush in a grid, and then create a guide to help you find the upper-left corner of each tile-brush.

How do I move a layer?

You can pickup an entire layer by grabbing the entire floor layer as a tile-brush using the v key (or choosing **Brush|Select Plane**) and then changing the floor to a new layer and stamping your tile-brush. Alternatively, you could save the layer you wish to move, change to the new layer, and load the saved layer back into memory.

How do I create a blueprint for large rooms?

If you are creating a really big room, and you need some help setting things in place, you can draw a small representation of your room in DPaint where each pixel represent a tile. Save your picture as a brush the size you want the room. An external program called MAKEROOM.EXE will take your picture and create a loadable tUME map with a room the size of the brush you saved. Inside this room there will be a tile #1 for each pixel of color 1 in your picture, a tile #2 for each color 2 pixel in your picture, etc. This can be helpful for things like creating large circles of tiles and other shapes.

How do I load a DPaint picture as a map?

A program called CUTTILES.EXE will take a DPaint picture and cut it into tiles. First it divides the picture into tiles and discards tiles of common colorsets and tiles that are a flipped version of other tiles. Then it creates a DPaint picture of the tiles, as well as a map to load using these new tiles to re-create your original DPaint picture in tUME. See **Importing and Exporting Maps: CUTTILES.EXE**.

How do I find tiles that are "infrequently used"?

Use the show tile usage option. See **Tiles|❑Show Tile Usage** and **Tiles|Set Usage Limit...**

How do I count tiles?

If you want to know how many times a specific tile or tile-brush appears, then choose **Brush|Count** to count the number of occurrences of the brush in the current layer.

If you want to count the number of characters in a tileset, then choose **Tiles|Count**.

How should I set DPaint up when editing tiles?

If you have tiles arranged in the upper left corner, and you are using the **Tiles|Load...|Full Tiled** option, the **Tiles|Load...|Tiled Blanks** option, or the **Tiles|Load...|All Tiled** option, then this tip is for you.

A great way to manipulate tiles in DPaint is to set up a **grid**. Click the right mouse button on the Grid Tool and type in your tile size (usually 16x16), now click on **Adjust** and press [F10] to remove the menus, move the cursor to the top left pixel and click the left mouse button. Press [F10] to bring the menus back, and then choose **Pref|Exclude Edge**. Now it is easy to grab tiles in DPaint for manipulating. Press g to turn the grid off if you want to edit an individual tile, and press g again to go back to manipulating whole tiles.

How do I copy a tUME project to another machine?

You must copy the map file, and all DPaint tileset pictures associated with that map file. There are two problems here: 1) making sure all the DPaint tileset pictures get moved to the new machine, 2) making sure that tUME can find the DPaint tileset pictures on the new machine.

tUME doesn't help you solve the first problem; the easiest solution is to ask whoever created the map to make a list of the source DPaint tileset pictures. If you are missing a tileset picture, and you can't remember its name, you can choose **Tiles|Set Info...** on that source tileset and it will display the filename it is using to load the DPaint picture. Note that you may have several different tilesets displayed in the same room; be sure to check all of them.

The second problem is that tUME remembers which directory to look in to find a DPaint tileset picture. One solution is to place the DPaint tileset pictures in the same sub-directory as the map file, and then tell tUME to search the current sub-directory by setting SearchAsSpecified=0 and SearchCurrentDir=1 in the [Load Options] group of the tUME.INI file.

If you prefer to keep DPaint tileset picture in its own directory separate from tUME map files, then you have two options: 1) make the exact same directory and sub-directory structure on the new machine as on the old; or 2) use the [Load Options] group command SearchDir=*search_directory*. See **Configuring Tileset Search Path** section of the *tUME Configuration Guide*.

How do I find the location in the tileset source room for a particular tile?

See **Tiles|Highlight Tile**, below.

How do I make the tiles automatically go to the right layers?

See the **Stratify Paste** section, below.

How do I associate a separate collision layer with my map?

Add a second layer to your room, load in collision tiles and assign them a Tileset Type of Contour Tiles, then paste these Contour Tiles into the second layer. See the second layer of the example map **3LAYER.MAP**.

How do I associate a collision tile with each image tile?

As an alternative to creating a second layer with collision information, you may create a **conversion room** (which see) that associates each **Image Tile** with a corresponding **Contour Tile**. One way to organize conversion rooms is by having two rows next to each other, where the top row specifies the image, and the bottom row specifies the contour for that image. See the **Conversion Room** in the example map **CONVERT.MAP**.

How do I specify where objects/monsters go on my map?

Add a fourth layer to your room, load in icons of **objects** tiles and assign them a Tileset Type of Object Tiles, then paste these Object Tiles into the fourth layer. If you do not want four layers, edit the tUME.INI to redefine the meaning of layers. In the example **3LAYER.MAP**, the objects have been given the tiletype of Special Tiles, and they appear in the third layer.

How do I define an invisible path for my player/vehicle to follow?

In an manner analogous to defining a collision layer, you draw special tiles that represent a path for a player or vehicle to follow. You may paste these tiles into the same layer as the Contour Tiles, or create another layer for paths only. The choice is entirely yours, and you should modify tUMEPack to process path tiles appropriately.

Reference

Project Menu

Load... (keyboard equivalent: [Alt]-l)
Choose this option to load a tUME IFF map file that was previously saved using **Project|Save...|Normal**, **Project|Save...|Save+TMGC**, or **Project|Save|Save+TMGX**. All tilesets and rooms in memory will be cleared before loading the new map.

Append...
Choose this option to append a tUME IFF map file that was previously saved using **Project|Save...|Normal**, **Project|Save...|Save+TMGC**, or **Project|Save|Save+TMGX** to the one currently in memory.

Save...
Choose one of the following three option to save all tilesets and rooms in memory to a tUME IFF map file. The Save+TMGC and Save+TMGX saves additional information that is need by some tUMEPacks. See the *tUME Programmer's Guide* for a description of TMGC and TMGX chunks.

Normal (keyboard equivalent: [Alt]-s)
Choose this option to save all tilesets and rooms in memory to a tUME IFF map file.

Save+TMGC
Choose this option to save all tilesets and rooms in memory to a tUME IFF map file. Also save an additional TMGC chunk in the tUME IFF map file that contains the graphic image of all tiles. Some tUMEPacks, such as tPMCKid2, need this information to operate properly.

Save+TMGX
Choose this option to save all tilesets and rooms in memory to a tUME IFF map file. Also save an additional TMGX chunk in the tUME IFF map file that contains the graphic image of all tiles. Some tUMEPacks, such as tPMCKid2, need this information to operate properly.

Clear...
Choose this option clear all tilesets and rooms from memory.

Show Status
Choose one of the following six options to change what is displayed in the status bar:

Room Info (keyboard equivalent: [F6])

'Room Info' for Source Room

µ §

When you choose this option on a source room, you will see a status bar as pictured above. The **Tileset Name** displays the name for the first tileset in this room. **S** stands for Source room and **L** stands for Locked Room. The rest is the Width, Height, Depth, Floor, and flags for this room; for a source room this information is not very useful. If there is a '-' dash in front of the **Tileset Name** then these tiles have been marked so that their graphics will not be saved.

'Room Info' for Edit Room

µ §

When you choose this option on an edit room, you will see the **Room Name**. **E** stands for Edit Room, **U** stands for Unlocked (Unlocked means you can edit it). The room above is 50 tiles wide, 25 tiles tall and 1 layer deep. The floor is set to layer 1 and it is visible and unlocked (editable).

User Info (keyboard equivalent: [F7])

'User Info' for Source Room

µ §

When you choose this option on a source room, you will see the above status bar. Clicking on a tile will show you that tile's **Tileset User Type** and **Tileset User Number**.

'User Info' for Edit Room

µ §

When you choose this option on an edit room, you will see the above status bar. The status bar shows the room's **Room User Type** and **Room User Number**.

Tiles (keyboard equivalent: [F5])

'Tiles' status bar for Source and Edit Room

µ §

When you choose this option, you will see the above status bar displaying information about the tile under the pointer. In a source room the tile must be clicked on but in an edit room you may just move the pointer. Shown is the Tile's **Tileset Name**, **Tileset User Number**, its **Tile Number** and **Tileset User Type**. Tiles are numbered 1 to N where N is the number of tiles in the current Tileset. If there is no tile under the cursor the message (NULL TILE) will be displayed.

Coordinates (keyboard equivalent: [F9])

Coordinates for Edit Room

µ §

When you choose this option, you will see the above status bar. Coordinates don't make much sense for a source room but in an edit room, the position of the tile under your pointer will be displayed. If you are grabbing a tile-brush then the display changes to show the brush width and height while you select it.

Version

Choose this option to display the tUME version number in the status bar.

Copyright

Choose this option to display the tUME copyright in the status bar.

❑TitleBar (keyboard equivalent: [F10])

Check this option to make the status bar appear.

About tUME

Choose this to display how much main, EMS and XMS memory is available, and other useful facts about tUME.

Quit (keyboard equivalent: [Shift]-Q or [Alt]-x or q)

Choose this option to leave tUME. Please make sure that you have saved your most recent changes using **Project|Save...|Normal** before exiting; tUME does not warn you about unsaved changes.

Tiles Menu

Load...

A tileset is created whenever you load a DPaint picture into tUME. Every time you load some tiles into tUME from a DPaint picture you create a new tileset. tUME does not group tiles from different pictures into one tileset.

When you load a tileset you are given the option to 'append' the new tileset to a pre-existing tileset source room. If you choose yes, the new tiles will be appended to the room you choose. You may have to scroll through the room to see all the tiles. If you choose not to 'append' then the new tiles are loaded into their own source room. To change the source room you are viewing press 1 or 2. 1 moves backward through the source rooms and 2 moves forward. If you get the message (no more to show) that means there is only one source room.

Choose one of the following five load options to load a DPaint picture into tUME for use as a source tileset (If you can't decide between which of the five loading methods to use, I recommend using **All Tiled** as it is not sensitive to the background color, and is more forgiving about adding tiles to the source picture):

Full Tiled

This method asks for the size of the tiles. Then it proceeds to cut tiles of the specified size starting at the top left corner of the picture and proceeding to the right. When it finds two consecutive blank tiles it will stop scanning the current row and go down to the next row of tiles. If the next row starts with two blank tiles it will stop scanning completely and assume it has finished.

NOTE: tUME decides which tiles are blank by looking at the 'background' color from your DPaint picture. The background color is the last color you selected for the right mouse button in DPaint. Pay special attention to this color. If you draw a picture with 4 tiles and pull it into tUME and then later you edit those 4 tiles in DPaint and you accidentally change the background color, tUME will now pull in a lot more then 4 tiles because it will won't find any blank tiles. To see what I mean, load **GOODBACK.LBM** using **Tiles|Load...|Full Tiled** tileset and set the size to 16x16. Then load **BADBACK.LBM** the same way. Note that the only difference in DPaint is the current background color in the tool palette (or icon bar). The biggest problem here is that your maps will look all messed up if you make this mistake. If you maps get totally screwed up remember to load your tilesets into DPaint and check their background color. That is usually the problem.

WARNING! This load method is **not** recommended if you are still adding tiles to your DPaint picture! E.g., you have twenty tiles in your picture, with five rows of four tiles each. If you add a tile to the end of the first row, **all subsequent tiles will be renumbered, while your maps will still contain the old numbers!** If you wish to add tiles to a picture loaded with the Full Tiled method, then you will need to add it to the **end** of the DPaint picture.

Tiled-Blanks

Tiled-Blanks works like Full Tiled except that when a blank tile is found, it is not added to the tileset. For example, if you loaded a DPaint picture that looked something like:

µ §
Using the Full Tiled method your tiles would be loaded as follows

tile #1	tile #2	tile #3	tile #4
tile #5	tile #6	tile #7	tile #8
tile #9	tile #10	tile #11	tile #12

Using the Tile-Blanks method you tiles would be loaded as follows instead

tile #1	NULL	tile #2	NULL
NULL	tile #3	NULL	tile #4
tile #5	NULL	tile #6	NULL

WARNING! This load method is **not** recommended if you are still adding tiles to your DPaint picture! E.g., you have twenty tiles in your picture, with five rows of four tiles each. If you add a tile to the end of the first row, **all subsequent tiles will be renumbered, while your maps will still contain the old numbers!** If you wish to add tiles to a picture loaded with the Tile-Blanks method, then you will need to add it to the **end** of the DPaint picture.

All Tiled (keyboard equivalent: [Ctrl]-I)

This method asks for the size of the tiles. Then it proceeds to cut tiles of the specified size starting at the top left corner of the picture and proceeding to the right. This method cuts out every tile in the picture on the grid regardless of whether it is 'blank' or not.

NOTE: It is **safe** to add tiles anywhere in the source DPaint picture loaded using this method, **so long as you do not change the width of the DPaint picture**. You may also add additional tiles by lengthening the picture. If you obey these two rules, the tiles will not renumber.

As Brushes

This method expects to find the tiles in a grid, with background color separating the tiles. tUME loads the picture and scans for the first pixel that is not the background color. When it finds that pixel it then assumes it is the start of the first tile and scans it to figure out the size of the tile. It then assumes that the tiles are laid out in a grid so it scans to the right to find the next tile and it scans down to find the next row of tiles. Now it knows the size of a tile and the distance between the tiles so it quickly pulls in the rest of the tiles. See example file **BRUSHES.LBM**.

NOTE: If you want to add more tiles as brushes, you should add them below all existing brushes.

Boxed

The boxed method expects to find the tiles in a grid, with a one pixel border (the 'box') around each tile. tUME loads the picture and scans for the first pixel that is not the background color and is not the same as the pixel in the top left corner of the screen. When it finds that pixel it then assumes it is the start of the first box and scans it to figure out the size of the tile. It then assumes that the tiles are laid out in a grid so it scans to the right to find the next tile and it scans down to find the next row of tiles. Now it knows the size of a tile and the distance between the tiles so it quickly pulls in the rest of the tiles. See example file **BOXED.LBM**.

NOTE: If you want to add more boxed tiles, you should add them below all existing brushes.

GridRoomAsTiles...

Choose this option to make **Composite Tiles** (which see, above).

E.g., the "native" tile size is 8x8 pixels, and you want to use 16x16 pixels characters that are created from the 8x8 pixel characters.

1. Draw your 8x8 pixel tiles and load them into tUME as 8x8 pixel tiles.

2. Create a room large enough to hold your composite tileset (say 16x16 tiles), and fill it with your 8x8 pixel tiles placing your tiles in a 2x2 tile grid (see the 16x16 Composite room in **COMPOSIT.MAP**).

3. Choose **Tiles|GridRoomAsTiles...** tUME will bring up a dialog box about the composite tileset.

4. You must give each composite tileset a unique name. If you gave your edit room a name when you created it, it appears in the topmost box as the default composite tileset name. If the topmost box is empty, then you need to click in it and type in a unique name to identify this composite tileset.

5. Press [Tab] three times to move the cursor to the Tile Width edit box then type 2. Press [Tab] once to move the cursor to the Tile Height edit box then type 2. This tells tUME we want to create composite tiles that are two tiles wide by two tiles high.

6. We're done, so click on **OK**.

Voila - tUME has created a new composite tileset made of 2x2 tiles created from 8x8 pixel tiles. These composite tiles may be used in exactly the same manner as a tileset made of 16x16 tiles. You may freely mix composite tiles and source tiles in an edit room, as long as they are the same size (e.g., a source room made of 16x16 pixel tiles would be the same size as a composite tileset made of 2x2 8x8 pixel tiles).

When you created a composite tileset, it appears in a locked room (status bar says **CL** to the right of the room name).

Now create a new room, grab some of your 16x16 pixel composite tiles and paste them into the new room. You may even create composite tile out of the 16x16 composite tile (though I don't know why you'd want to do that).

Save...

Choose this option to save all tilesets in the TMGX chunk in a map file. See the *tUME Programmer's Guide* for more information about TMGX chunks. If you want to save the tilesets in the TMGC chunk instead of TMGX, choose **Project|Save...|Save+TMGC**.

Delete...

Choose this option to delete a tileset. Generally speaking, you will not want to delete a tileset. But occasionally, say perhaps you accidentally loaded a tileset that you don't need, you'll want to delete the unnecessary tileset.

To delete a tileset, make the source room with the tileset to delete visible. Select one of the tiles in the tileset to delete, and choose **Tiles|Delete...**. tUME will ask you if you are sure you want to

delete the tileset and all references to that tileset. Click **Yes**.

Note that any tiles from the deleted tileset occurring in any edit rooms will be set to NULL tiles after the delete tileset operation.

Set Info...

Choose this option to change a tileset's name, tileset type, tileset number, or its comments.

The type and number have no significance or meaning except the meanings you give them for your project (which is usually the meaning assigned by your particular version of tUMEPack).

The long box that runs the entire length of the dialog box along the top is the tileset name. This is DPaint file that was loaded to create the tiles for this tileset.

The tileset type (**User Type**) can be entered by either typing a number in the box or (better yet), by selecting a choice from the tileset types list on the left side of the dialog box.

The two comment lines at the bottom are descriptions that you want associated with the current room. Some tUMEPacks use these lines to list additional switches specifying how the current tileset should be processed.

Count (keyboard equivalent: [Alt]-c)

Choose this option to count characters. This feature is user-definable, and may be used to count the total number of characters in your source tileset, the number of characters used in a specific room, or the number of characters used in all rooms.

As supplied, tUME will count the number of 8x8 image characters (TilesetType = 0) loaded. It counts SNES mode 2 characters, so it only looks at the lowest four bits in determining whether two tiles have the same image. It will check for X-flip, Y-flip, and XY-flip in determining whether two characters are the same. Tiles that also appear in Table Rooms (RoomType = 1) get count separately, without merging duplicate tiles nor checking for flipped tiles.

To change how tUME counts characters, see the **Configuring Character Counting** section in the *tUME Configuration Guide*.

Highlight Tile (keyboard equivalent: [Alt]-h)

Sometimes, when you have many source tiles, it becomes difficult to see where a particular tile is defined. Choose this option to locate the tile within the source room where the tile is defined.

To highlight a tile in a source room, first select a tile-brush that includes the tile you wish to highlight in the upper left corner. Choose **Tiles**|**Highlight Tile** (or press [Alt]-h), and tUME will change the view to show the source room and the highlighted tile.

If there are more tiles in the tile-brush then subsequent presses of [Alt]-h will cycle through the tiles in the tile-brush from left to right, top to bottom, floor layer to ceiling (wrapping at end).

❑Show Tile Usage (keyboard equivalent: [Alt]-u)

Check this box to make a grid of tile usage numbers to appear over every floor tile in the room. The number shows how many times the tile in the floor layer is used.

The numbers have different meanings, depending on whether the room is an edit or a source room. In an edit room, the tile usage number indicates how many times the floor tile appears in the **current** edit room only. In a source room, the number indicates how many times the floor tile appears in **all** edit rooms.

If the tile usage count is too large to be displayed in the current tile, +'s will displayed instead of the tile usage count. In this case, zoom in to make each tile larger and to show more of the tile usage count.

You can limit the largest tile usage count displayed by choosing **Tiles|Set Usage Limit...**

Set Usage Limit...

Choose this option to bring up a dialog box that will allow you to set the largest tile usage count that will be displayed. Enter a new maximum tile usage count, press [Enter], and tUME will display all tile usage counts that are less than or equal to your limit value.

E.g., you want to see tiles that are "infrequently used". You have arbitrarily decided that a tile is infrequently used if it appears five times or less. Choose **Tiles|Set Usage Limit...**, enter 5, and press [Enter]. tUME will now display tile usage numbers only for the tiles that are used five times or less.

Export...

Choose this option to save every tile of every tileset as a brush. For every tileset loaded that has its ❑**SAVE GRAFX** box checked (found in the **Tiles|Set Info...** dialog box), tUME will bring up a file requester asking you to enter the base name of the sub-directory to create. tUME will append '.TBI' to the entered base name to create the sub-directory name, and then it will save every tile in the tileset as a brush within that sub-directory.

E.g., there is a tileset named CONTOUR.LBM. Choose **Tiles|Export...**, and tUME brings up a file requester that asks "TBI for CONTOUR". You enter TEST. tUME will create a sub-directory TEST.TBI, and then it will save all tiles as IFF brushes in TEST.TBI as CONTOUR.001, CONTOUR.002, etc.

Room Menu

Create...

Choose this option to create a new edit room. A dialog box will appear, allowing you the enter the name, room type, room number, size, and comments for the new room. Please see **Room|Set Info...**, below, for a description of the fields in the dialog box.

Load...

Choose this option to load a room that was saved using the **Room|Save...** option into the current map.

Save...

Full
Choose this option to save the current room and all tilesets as a map.

Stripped
Choose this option to save the current room and only those tilesets that are actually used by the room as a map.

Clear...

Tiles
Choose this option to clear all the layers of the room to NULL tiles. Note that tUME remembers the size of tiles that was pasted into this room.

Complete
Choose this option to delete all layers but one and to clear the remaining layer to NULL tiles. Note that the tile size is reset (you may paste any size tile into this room), and that any tile-brush from this room will also be cleared.

Delete...
Choose this option to delete the current room. Note: If the brush is from the current room, it will be deleted as well.

Set Info...
Choose this option to change a room's name, room type, room number, size, or its comments.

The name, type and number have no significance or meaning except the meanings you give them for your project (which is usually the meaning assigned by your particular version of tUMEPack).

The long box that runs the entire length of the dialog box along the top is the room name. This is used by some tUMEPacks (such as tPMCKid2) to specify the DOS filename to use for the room.

The room type (**User Type**) can be entered by either typing a number in the box or (better yet), by selecting a choice from the room types list on the left of the dialog box.

Set the width and height of the room in tiles by changing the numbers in the Room Width and Room Height boxes.

The two comment lines at the bottom are descriptions that you want associated with the current room. Some tUMEPacks use these lines to list additional switches specifying how the current room should be processed.

❑Lock...
Check this box to lock a room and prevent brushes from being pasted into the room. Locking a room makes it easier to select tiles from the room; you can drag-select tiles from it without first pressing b. When a room is locked it will now longer show up on the edit pane, instead it will

show up on the source pane.

Copy Color

The copy color options allow you to copy colors from one room to another. Select a tile-brush from the source room, move to the destination room, then select one of the following three options.

All

Choose this option to copy the color cycling information and the palette from the tile-brush to the current room.

Palette

Choose this option to copy just the palette (256 color) from the tile-brush to the current room.

Color Cycles

Choose this option to copy the color cycling information from the tile-brush to the current room.

Set Palette... (keyboard equivalent: p)

Choose this option to bring up the palette requester for the current room. See **Palette Requester**, above.

❑Live Palette

Check this box will cause the palette to be sent to an attached development system every time you make a change in the palette requester. This will allow you to adjust colors and see the results on the target machine immediately. See **Download to a Development System**, above.

❑Color Cycle (keyboard equivalent: [Tab])

Check this box to show any color cycles you have defined for the current room. See **Palette Requester**, above, for information about defining a color cycle range.

Export...

Choose this option to export the current room as an IFF picture. See **Exporting Rooms**, below.

Print...

Choose this option to print the current room to a HP LaserJet. See **Printing Rooms**, above.

Layer Menu

Add

Choose this option to add a layer to the topmost layer of the current room.

Insert
Choose this option to insert a new layer at the floor layer of the current room. The previous floor layer and all layers above the floor get pushed up by one.

Load...
Choose this option to load a saved layer and insert it as the current floor layer of the current room. The previous floor layer and all layers above the floor get pushed up by one.

Append...
Choose this option to load a saved layer and append it as the topmost layer of the current room.

Save...
Choose this option to save the floor layer of the current room.

Delete...
Choose this option to delete the floor layer of the current room.

Move Up (keyboard equivalent: [Alt]-)
Choose this option to move the floor up one layer. Note that you may also press [Alt]-[Shift]- to move to the topmost layer.

Move Down (keyboard equivalent: [Alt]-↓)
Choose this option to move the floor down one layer. Note that you may also press [Alt]-[Shift]-↓ to move to the bottommost layer.

❑**EditOnlyFloor** (keyboard equivalent: [Alt]-f)
Check this option to display only the floor layer, and to limit editing to only the floor layer. **EditOnlyFloor** is a global operation. This means that it affects the view of all rooms until you turn it off. A lower-case f will appear in the extreme right of the status bar to indicate you are in **EditOnlyFloor** mode.

❑**Invisible**
Check this option to make the floor layer invisible. An invisible layer is still editable.

❑**Lock** (keyboard equivalent: l)
Check this option to lock the floor layer. A locked layer is not editable.

Invis+Lock (keyboard equivalent: i)
Choose this option to make the floor layer locked and invisible. A locked layer is not editable.

Download

❑One Screen

Check this option to download only one screen of data at a time (instead of the entire layer size). This affects the next two menu options. See **Download to a Development System**, above.

16-Color Chars

Choose this option to download the floor using 16-color characters. If **Download|❑One Screen** is selected, only the screen around the pointer position will be downloaded. See **Download to a Development System**, above.

256-Color Chars

Choose this option to download the floor using 256-color characters. If **Download|❑One Screen** is selected, only the screen around the pointer position will be downloaded. See **Download to a Development System**, above. This option has the same effect as **Download|16-Color Chars** on SEGA Genesis systems.

Brush Menu

Undo (keyboard equivalent: u)

Choose this option to removes the last tiles stamped. Choose this option again to restore the last tiles stamped.

Select Block (keyboard equivalent: b)

Choose this option to select a tile-brush from the floor layer and every layer above it. After pressing b, drag-select tiles using either the left or right mouse button. Drag-selecting using the left mouse button will leave a copy of the image in the current room, while drag-selecting using the right mouse button will "cut" out the image and leave NULL tiles. See **Editing Layers**, above.

Select Plane (keyboard equivalent: v)

Choose this option to select a tile-brush from the current floor layer. After pressing v, drag-select tiles using either the left or right mouse button. Drag-selecting using the left mouse button will leave a copy of the image in the current room, while drag-selecting using the right mouse button will "cut" out the image and leave NULL tiles. See **Editing Layers**, above.

Strip Brush (keyboard equivalent: [Shift]-X)

Choose this option to remove all layers except the bottommost layer from the current tile-brush.

Search

tUME allows you search for a particular tile-brush, and to replace what you've found with another tile-brush.

Set Buffer (keyboard equivalent: [Ctrl]-s)

Choose this option to place the current tile-brush in the Search Buffer. tUME will search for whatever is in the Search Buffer.

Search (keyboard equivalent: s)

Choosing this option will search across, then down, starting at the current pointer location, for the next occurrence of the Search Buffer.

To find a particular group of tiles, follow these steps:

1. Select the tiles to search for: press v or b, then select the tiles you want to search for.

2. Place the tiles to search for in the Search Buffer by pressing [Ctrl]-S or choosing **Brush|Search|Set Buffer**.

3. Switch to the room you wish to search, then position the pointer to where you want to start searching from. Note that if you position the pointer directly over a match, search will find the next match.

4. Press s or choose **Brush|Search|Search** to find the first match. tUME will search across, then down starting from the current pointer location. The room will be re-positioned, and the pointer will be placed over the match, which will be surrounded by an outline.

5. Press s again to find the next match. Once a match has been found, subsequent searches will continue from the previous match (the search outline).

When searching, tUME matches tiles starting from the current floor and up. Invisible layers match everything. E.g., you place a three layer brush in the Search Buffer. In the edit room, you place the floor on the first layer, and you make the second layer invisible. tUME will match the first layer of the Search Buffer with the first layer of the edit room, and the third layer of the Search Buffer with the third layer of the edit room. Since the second layer of the edit room is invisible, it matches everything, and is not compared against the second layer of the Search Buffer.

Note: you may wish to press [Ctrl]-[Home] between steps 3 and 4 to move to the start of the room.

Note: when you place a tile-brush into the Search Buffer, the previous match (search outline) is removed.

Note: by default, searching tries to match tile-flipping, tile-priority, and tile-colorsets exactly. However, you may change this by modifying the [Search Options] section tUME.INI file. See **Configuring Search and Replace** section of the *tUME Configuration Guide*.

Replace (keyboard equivalent: r)

Choosing this option will search across, then down, starting at the current pointer location, for the next occurrence of the Search Buffer and bring up a dialog box asking if you want to replace what was found with the tile-brush.

To find a particular group of tiles, and replace the tiles with another group of tiles, follow these steps:

1. Select the tiles to search for: press v or b, then select the tiles you want to search for.

2. Place the tiles to search for in the Search Buffer by pressing [Ctrl]-S or choosing **Brush|Search|Set Buffer**.

3. Select the tiles to replace with: press v or b, then select the tiles you want to replace with.

4. Switch to the room you wish to search and replace, then position the pointer to where you want to start searching and replacing from. Note that if you position the pointer directly over a match, the replacing will start from the next match.

5. Press r or choose **Brush|Search|Replace** to find the first match. tUME will search across, then down starting from the current pointer location. The room will be re-positioned, and the pointer will be placed over the match, which will be surrounded by an outline.

6. tUME will bring up a dialog box that asks you, "Replace tiles?", with four options: **Yes**, **No**, **All**, and **Exit**.

A. Selecting **Yes** or pressing [Enter] will replace the matching tiles with the tiles in the tile-brush, and tUME will find the next matching tiles, and bring up the "Replace tiles?" dialog box again.
B. Selecting **No** will skip replacing this match, and tUME will find the next matching tiles, and bring up the "Replace tiles?" dialog box again.
C. Selecting **All** will cause tUME to find each and every matching group of tiles in the current room, and replace each and every one with the current tile-brush. Pressing [Spacebar] will interrupt this function.
D. Selecting **Exit** or pressing [Esc] will stop the replace function.
Notes about Search and Replace:

You may undo the last search and replace operation by pressing u.

Invisible layers match everything; they are not searched.

tUME only replaces tiles in unlocked layers. Search and replace uses the current brush paste mode setting (either paint mode or replace mode) when replacing tiles.

❍Paint (keyboard equivalent: [F1])
Check this option to set the paint mode. In this mode, stamping with the left mouse button will stamp only the non-NULL tiles in tile-brush into the room. Stamping with the right mouse button will erase tiles in the room to NULL only if the corresponding tile in the tile-brush is non-NULL.

❍Replace (keyboard equivalent: [F3])
Check this option to set the replace mode. In this mode, stamping with the left mouse button will stamp both NULL and non-NULL tiles in tile-brush into the room. Stamping with the right mouse button will erase the entire rectangular tile-brush area to NULL tiles.

❑Stratify Paste
Check this option to enable the Stratify Paste mode.

If you are editing several layers, it can be troublesome to have to keep moving up and down

layers to paste to the right layer. tUME has a feature called **Stratify Paste** which automatically sends tiles to the right layer when you paste. To make this feature work, check **Brush|❑Stratify Paste**. Now when you paste tiles into an edit room, the tile will go to the layer defined by its Tileset Type.

Image Tiles, **4-Color Tiles**, **256-Color Tiles**, and **Mode 7 Tiles** all get pasted into the first layer. **Contour Tiles** are pasted into the second layer, **Special Tiles** into the third layer, and **Object Tiles** into the fourth layer.

The layers that tilesets are pasted into may be changed by editing the tUME.INI file; see the **Redefining Tileset Types** section of the *tUME Configuration Guide*.

Set Brush Mode

Normal (keyboard equivalent: [keypad .])
Choose the option to exit the colorset coloring mode. See **Colorsets**, above.

Recolor (keyboard equivalent: [keypad 0])
Choose the option to enter the colorset color-erase mode. See **Colorsets**, above.

Count (keyboard equivalent: h)
Choose this option to count the occurrences of a tile-brush in the current room.

tUME can tell you how often the tile-brush appear in the current layer of the current room. Press v to select a single layer brush, then drag-select the tiles you wish to count. Choose **Brush|Count** to count the number of occurrences of the brush in the current layer.

This function also extends across layers. If you press b to select your brush, then all layers in the brush must match, starting with the current floor layer and proceeding up. Thus the first brush layer must match the tiles in the floor layer, AND the second brush layer must match the tiles at the same X, Y coordinates in the next layer above the floor layer, and so forth.

If you make a layer invisible, then tUME will not match the tiles in that layer. E.g., you have a three layer room, and you are on the first layer. The first and third layers are visible, and the second layer is invisible. You press b to select a brush. Now when you choose **Brush|Count**, tUME will look at only the first and third layers in determining if the brush matches.

❑Show Tile-brush
Check this option to enable the Stratify Paste mode.

Check this option to display the floor layer of the current tile-brush you are holding as you drag it around. If this option is not checked, only an outline of the tile-brush will be shown. Showing just the outline causes the display to update slightly faster, and also allows you to see under the tile-brush to the room below.

❑Hide Cursor (keyboard equivalent: [F8])
Choose this option to hide the current mouse pointer.

Export...
Choose this option to export the current brush as an IFF picture. Note that the zoom and tile-spacing setting of the current room affects the exported image.

View Menu

Flip Panes (keyboard equivalent: [Space] or j)
Choose this option to display the other pane. Each window has two panes, a source pane and an edit pane.

Prev Room (keyboard equivalent: 2)
Choose this option to display the previous room in the current pane's list of rooms.

Next Room (keyboard equivalent: 1)
Choose this option to display the next room in the current pane's list of rooms.

Zoom

❑Toggle Zoom (keyboard equivalent: m)
Check this option to enlarge or reduce the display of tiles in the current pane.

Zoom Out (keyboard equivalent: <)
Choose this option to reduce the display of the current pane by one zoom setting.

Zoom In (keyboard equivalent: >)
Choose this option to enlarge the display of the current pane by one zoom setting.

Grid
Setting a drawing grid limits where you can draw tiles. If you set the grid to 4x4 then you will only be able to draw every 4 tiles across and 4 tiles down. You can toggle the grid on and off by pressing g (or choosing **View|Grid|❑Use Grid**). Grids have two attributes, the grid size, and the grid origin (where the upper left corner of the grid resides).

To set the grid size, use **View|Grid|Set Grid Size** or **View|Grid|Get Brush Size**. The grid size defaults to 2x2 tiles if you don't set it. To set the grid origin, move the cursor over the tile that you want to use as the grid origin, then press [Shift]-G.

❑Use Grid (keyboard equivalent: g)
Check this option to enable the grid.

Set Grid Size...
Choose this option to set the size of the grid. A dialog box will appear, allowing you to enter the grid size.

Get Brush Size (keyboard equivalent: [Alt]-g)

Choose this option to set the size of the grid to the same size as the width and height of the current tile-brush.

Guide

Setting a guide show a vertical line every X tiles and a horizontal line every Y tiles on screen. These guides have absolutely no effect on the room itself; they merely provide a visual point of reference. You can toggle the guides on and off by pressing o (or choosing **View|Guide|❑Show Guide**). Guides have two attributes, the guide size, and the guide origin (where the upper left corner of the guide resides).

To set the guide size, use **View|Guide|Set Guide Size** or **View|Guide|Get Brush Size**. The guide size defaults to 16x16 tiles if you don't set it (this is one SNES screenful if you are using 16x16 pixel tiles). To set the guide origin, move the cursor over the tile that you want to use as the guide origin, then press [Shift]-O.

❑Show Guide (keyboard equivalent: o)

Check this option to display the guide.

Set Guide Size...

Choose this option to set the size of the guide. A dialog box will appear, allowing you to enter the guide size.

Get Brush Size (keyboard equivalent: [Alt]-o)

Choose this option to set the size of the guide to the same size as the width and height of the current tile-brush.

❑Spaced Toggle (keyboard equivalent: \)

Check this option to cause a single pixel "spacer" to be drawn between each tile in a room. This makes it easier to see where the tiles are.

❑EditColorsOnly (keyboard equivalent: [Alt]-e)

Check this option to cause source rooms be displayed using the last edit room's palette.

Disable

By default, tUME shows the tile priority, flip and colorset settings. You may disable the display of priority, flip, or colorsets through one of the following three options:

❑Priority

Check this option to disable the display of tile priorities.

❑Flip

Check this option to disable the display of flipped tile.

❑Colorsets
Check this option to disable the display of tile colorsets.

❑Scroll Lock
Check this option to prevents tUME from scrolling the room.

Pane
By default, the source pane in a window displays only source rooms (check **View|Pane|❍Only Source**), and the edit pane in a window displays only edit rooms (check **View|Pane|❍Only Edit**). You may change a pane to show both edit and source rooms by checking **View|Pane|❍Allow All**, or lock a pane to show only rooms that are the same type as the current room by checking **View|Pane|❍Only Same**.

❍Allow All (keyboard equivalent: [Alt]-a)
Check this option to display both source and edit rooms in the current pane.

❍Only Source
Check this option to display only source rooms in the current pane.

❍Only Edit
Check this option to display only edit rooms in the current pane.

❍Only Same
Check this option to display only rooms of the same type as the currently viewed room in the current pane.

Bkgnd Color
Set the color register to use to drawing NULL tiles.

Next Color (keyboard equivalent: [Ctrl]-])
Choose this option to use the next color register in the palette to drawn NULL tiles.

Prev Color (keyboard equivalent: [Ctrl]-[)
Choose this option to use the previous color register in the palette to drawn NULL tiles.

Zero Color (keyboard equivalent: [Ctrl]-\)
Choose this option to use color register zero to drawn NULL tiles.

oToggleSmartFlip
Check this option to change the behavior of **View|Flip Panes**. Normally, **View|Flip Panes** switches between the edit and source panes. However, when **View|o**__**ToggleSmartFlip**__ is active, choosing **View|Flip Panes** in an edit room will find and make visible the source room that with

tiles that belong in the edit room's floor layer.

If you have more than one source room whose tiles can appear in the current floor layer, then **Smart Flip** will show the source room that you most recently selected a tile-brush from.

The tileset chosen for a given layer are specified through the tileset user types and are defined in the tUME.INI file; see the **Redefining Layer Types** section of the *tUME Configuration Guide*.

Export Screen...

Choose this option to export the current screen as an IFF picture.

Troubleshooting

tUME says I'm out of memory, and I know I have plenty of memory!

When tUME tells you it has run out of memory, you should choose **Project|About tUME** to see if you are low on main memory or EMS/XMS memory.

If you are running out of EMS/XMS memory, your options are limited. If you are happen to be running tUME under Windows, you may be able to configure Windows to supply more EMS/XMS memory. Otherwise, you'll have to upgrade the amount of memory in your machine.

However, it is more likely that you have run out of main memory. Here are some strategies for maximizing the amount of main memory available to tUME:

1. Use EMS instead of XMS memory. Due to the way tUME deals with EMS/XMS memory, you will typically gain about 64K by switching from XMS to EMS memory (and tUME will run faster as well!)

2. Load your device drivers high to maximize the largest executable program size reported when you type mem at the DOS prompt. See the manual for your memory manager for more information.

I can't flip (or set the priority of) my tiles!

1. Make sure that **View|Disable|❑Flip** (or **View|Disable|❑Priority**) is not checked.

2. Make sure that the tiles have not been set to a type that cannot be flipped (or prioritized). Please see **Redefining Tileset Types** in the *tUME Configuration Guide*.

I can't select a brush anymore!

1. Are you on the correct floor layer?

2. Currently, tUME only displays the floor layer of the current brush. Thus, if you have selected four layers, only the tiles in the bottommost layer will be shown. Stamp the brush to see which tiles have been selected.

I can't stamp my brush anymore!

You should check the following when you expect to be able to stamp your brush, yet when you stamp with the left mouse button, nothing happens:

1. Make sure that you are not in a brush coloring mode or priority setting mode. Press the . (period) key on the numeric keypad.

2. Make sure that the current room is not locked. Choose **Room|Lock** to unlock a locked room.

3. Make sure that the current layer is not locked. Choose **Layer|Lock** to unlock a lock layer.

Why does the word 'COLOR' appear attached to the pointer?

tUME has entered into color setting mode or priority setting mode, most likely through pressing of one of the keys on the numeric keypad. This feature is useful for placing tiles into different colorsets on target machines such as the Super Nintendo or the SEGA Genesis. Please see the **Colorsets** section, above.

My tilesets don't load correctly anymore!

If you are editing your DPaint tileset pictures, and quite mysteriously, tUME stops loading your tilesets properly, chances are you have changed the background color. Start DPaint, load your tileset picture, and set the background color by pressing **,** (comma), then clicking the right mouse button on the background color.

The tileset load formats **Full-Tiled**, **Tiles-Blanks**, **As Brushes**, and **Boxed** are all affected by the background color setting.

The Brush|Count function is not working!

Are you in the right layer? This function matches the number of layers in the tile-brush, starting with the floor layer and continuing with the layers above that.

Are some of the tiles to be matched flipped? The **Brush|Count** function is sensitive to flipped tiles. If some of the tiles are in a different flipped orientation, they will not be counted.

Are some of the tiles to be matched in a different colorset or have a different priority set? You can configure the feature to match flipped tiles, prioritized tiles, and different colorset tiles by changing the [Search Options] section of tUME.INI. See **Configuring Search and Replace** section of the *tUME Configuration Guide*.

Are the layers you want to match made visible?

Specifications

Hardware Requirements

IBM PC with 640K of memory, MCGA display, hard disk drive, and Microsoft-compatible mouse.

tUME will utilize additional memory, either EMS 4.0 memory (preferred), or XMS 2.0 memory.

Memory Requirements

Each pixel in a tileset requires one byte to represent, with some additional overhead for each tileset. Thus 1024 8x8 pixel characters would require slightly more than 1024x8x8, or 65536 bytes to represent.

Each tile in a map requires four bytes to represent, with approximately 2K overhead (to store the palette) for each room. Thus to store a 40x25 tile map would require 40x25x4+2K, or approximately 6K, while a 512x512 tile map would require 512x512x4+2K, or approximately 1Mb! Thus to edit a 512x512 tile map, you would need at least 1Mb of free EMS or XMS memory.

Making tUME Run Faster

If you are running tUME under Windows, and tUME runs very slowly (you can see each individual row in your map redraw), Windows is most likely swapping tUME's EMS or XMS memory to disk. This is bad. Try changing tUME's .PIF by selecting the **EMS Memory Locked** check box and the **XMS Memory Locked** check box in the **Advanced Options** dialog box in the .PIF editor to stop windows from swapping EMS or XMS memory to disk.

If you have a 386 or better machine, make sure plenty of EMS memory is available. Under MS-DOS 5.0 or later, you can type **MEM** at the DOS prompt to find out how much EMS and XMS memory is available. If no EMS memory is available, consider reconfiguring some of XMS memory as EMS memory by adding EMM386.SYS to your CONFIG.SYS file (see MS-DOS manual). If you are running tUME under Windows 3.1, use the supplied .PIF file. If you launch tUME from within a DOS box under Windows, make sure EMS memory is available.

Perhaps the least expensive way to increase tUME's operating speed is to upgrade to a fast VGA card. We recommend video cards that incorporate the Tseng ET4000 chip running at zero wait states, such as the Diamond SpeedStar. We do NOT recommend cards based on the S3 chip, such as the Diamond Stealth, as while these card significantly boost Windows performance, their performance with non-Windows applications is abysmal.

If you have a 286 or lesser machine, consider obtaining a 386 or faster machine. If you are purchasing a new machine, consider obtaining one with local-bus video technology. If a new machine is not an option, then consider an EMS memory board. Make sure that the card you purchase supplies expanded memory, not extended.

tUME Limitations

Currently tUME is limited to tiles that are less than or equal to 32768 pixels in size.

Key Assignments

Here are the default key assignments. Note that key assignments may be changed by modifying the tUME.INI file, see the *tUME Configuration Guide*.

Project Menu

[Alt]-L.....**Load...**
[Alt]-S.....**Save...|Normal**
[Alt]-X, q or Q.....**Quit**
[F5].....**Show Status|Tiles**
[F6].....**Show Status|Room Info**
[F7].....**Show Status|User Info**
[F9].....**Show Status|Coordinates**
[F10].....❑**TitleBar**

Tile Menu

[Alt]-C.....**Count**
[Alt]-H.....**Highlight Tile**
[Alt]-U.....❑**Show Tile Usage**
[Ctrl]-L.....**Load...|All Tiled**

Room Menu

p.....**Set Palette**
[Tab].....❑**Color Cycle**

Layer Menu

i.....**Invis+Lock**
l.....❑**Lock**
[Alt]-D.....**Download|16-color Chars**
[Alt]-F.....❑**EditOnlyFloor**
[Alt]- or 3.....**Move Up**
[Alt]-↓ or 4.....**Move Down**
[Alt]-[Shift]-.....move to topmost layer (not on menus)
[Alt]-[Shift]-↓.....move to bottommost layer (not on menus)
[Ctrl]-D.....**Download|256-color Chars**

Brush Menu

b.....**Select Block**
h.....**Count** (mnemonic "**h**ow many")
s.....**Search|Search**
r.....**Search|Replace**
u.....**Undo**
v.....**Select Pane**
x.....flip brush left to right (not on menus)
X.....**Strip Brush**
y.....flip brush top to bottom (not on menus)
[Ctrl]-S.....**Search|Set Buffer**
[Keypad]-0.....**Set Brush Mode|ReColor**
[Keypad]-.....**Set Brush Mode|Normal**
[F1].....❍**Paint**
[F3].....❍**Replace**
[F8].....❑**Hide Cursor**

View Menu

1 or [Alt]-→.....**Next Room**
2 or [Alt]-←.....**Prev Room**
\.....❑**Spaced Toggle**
<.....**Zoom|Zoom Out**
>.....**Zoom|Zoom In**
g.....**Grid|**❑**Use Grid**
G.....set grid origin (not on menus)
m.....**Zoom|**❑**Toggle Zoom**
o.....**Guide|**❑**Show Guide**
O.....set guide origin (not on menus)
[Spacebar] or j.....**Flip Panes**
[Alt]-A.....**Pane|**❍**Allow All**
[Alt]-E.....❑**EditColorsOnly**
[Alt]-G.....**Grid|Get Brush Size**
[Alt]-O.....**Guide|Get Brush Size**
[Alt]-,.....zoom out maximum (not on menus)
[Alt]-.....zoom in maximum (not on menus)
[Ctrl]-[.....**Bkgnd Color|Prev Color**
[Ctrl]-\.....**Bkgnd Color|Zero Color**
[Ctrl]-].....**Bkgnd Color|Next Color**

Set Colorset Keys

[Keypad]-.....disable set colorset mode
[Keypad]-0.....restore original colorset
[Keypad]-1.....set tiles to colorset 0
[Keypad]-2.....set tiles to colorset 1
[Keypad]-3.....set tiles to colorset 2
[Keypad]-4.....set tiles to colorset 3
[Keypad]-5.....set tiles to colorset 4
[Keypad]-6.....set tiles to colorset 5
[Keypad]-7.....set tiles to colorset 6
[Keypad]-8.....set tiles to colorset 7
[Keypad]-+.....enable tile priority

Special Keys

a....."again"; repeat last menu operation
n.....center display over tile under the cursor
[Alt]-P.....toggle palette to make status bar visible
←.....scroll left
.....scroll up
→.....scroll right
↓.....scroll down
[Ctrl]-←.....scroll left several tiles
[Ctrl]-.....scroll up several tiles
[Ctrl]-→.....scroll right several tiles
[Ctrl]-↓.....scroll down several tiles
[Ctrl]-[Home].....move to the upper-left corner of the room
[Ctrl]-[End].....move to the lower-right corner of the room

File Requester Special Keys

[Spacebar].....complete filename using highlighted name
[Baksp] on empty line move to parent directory
\ on empty line.....move to root directory
\.....move into sub-directory
[Enter].....enter sub-directory or choose file

Palette Requester Special Keys

b.....**BLEND**
c.....**COPY**
d.....**DELETE**
i.....**INSERT**
s.....**SWAP**
[Spacebar].....cancels **COPY**, **SWAP**, **BLEND** and **HSVSPREAD** command
[Enter].....leave palette requester and save all changes made to the palette
[Esc].....leave palette requester and abandon all changes made to the palette

Importing and Exporting Rooms

To convert a room ("level") saved as an IFF picture into a tUME room, use CUTTILES.EXE. To convert a tUME room to an IFF picture choose **Room|Export...**

To convert artwork from other file formats to IFF file format, or to scale artwork in either axis, or reduce colors, we recommend Handmade Software, Incorporated's Image Alchemy™. Contact HSI voice at (408) 358-1292, or HSI fax at (408) 356-4143, or download a shareware version from the HSI BBS at (408) 356-3297.

Importing Rooms: CUTTILES.EXE

If you have a room saved as a large IFF picture, you can use CUTTILES.EXE to convert it into a tUME room. CUTTILES finds duplicate tiles in the IFF picture, and creates a tUME project file that contains all unique tiles found, and a room that corresponds to the IFF picture.

To use CUTTILES, you need lots of EMS memory. If you don't have EMS memory, you can use XMS, but the program will run much slower. The command-line syntax is:

CUTTILES <*IFF_picture*> <*IFF_tiles*> {<*tUME_map*>} {*switches*}
<*IFF_picture*>.....the input IFF picture to convert to a tUME map
<*IFF_tiles*>.....the output filename to use to save the IFF picture with all unique tiles
<*tUME_map*>.....the output filename to use to save the tUME room in
{*switches*} are case-sensitive, and may be one or more of the following:

-W<*width*>.....create tile that are <*width*> pixels wide (default = 16)
-H<*height*>.....create tile that are <*height*> pixels high (default = 16)
+M.....force colors 252..255 to Menu colors (default ON)
+N.....remap NES colors: 0..3->0..3, 4..7->16..19, ... (default OFF)
+S.....<*IFF_picture*> is a SNES 16 color picture (default OFF)
+X.....merge X-flipped tiles when creating <*IFF_tiles*> (default ON)
+Y.....merge Y-flipped tiles when creating <*IFF_tiles*> (default ON)
+Z.....merge XY-flipped tiles when creating <*IFF_tiles*> (default ON)
Note that switches preceded with + are turned on with +*switch* and turned off with -*switch*.

Exporting Rooms

To export the current room as an IFF picture choose **Room|Export...**. The following settings affect how the exported image will appear:

- the layers currently visible (choose **Layer|❑Invisible** to change);
- the current **View|❑Toggle Zoom** and **View|❑Show Guide** settings;
- the current **Layer|❑EditOnlyFloor**, **View|❑Spaced Toggle**, and **View|❑EditColorsOnly** settings;
- the current **View|Disable|❑Priority**, **View|Disable|❑Flip**, and **View|Disable|❑Colorsets** settings; and
- the current **View|Background Color** setting.

To export the floor layer in a form that can re-imported back into tUME, use the following settings:

- uncheck **Layer|❑Invisible**;
- uncheck **View|❑Toggle Zoom** and uncheck **View|❑Show Guide**;
- check **Layer|❑EditOnlyFloor** and uncheck **View|❑Spaced Toggle**;
- check **View|Disable|❑Priority** and check **View|Disable|❑Colorsets**;
- set the **View|Background Color** to the same color as the transparent color.

Unfortunately, priority and colorset information do not get re-imported properly by CUTTILES.EXE.

Exporting Rooms: MAP2PIC.EXE

To use MAP2PIC, you need lots of EMS memory (about as much as the size of your tUME map file). If you don't have EMS memory, you can use XMS, but the program will run much slower. The command-line syntax is:

MAP2PIC <*tUME_map*> {*switches*}
<*tUME_map*>.....the input tUME map to convert into IFF picture.
{*switches*} may be one or more of the following:

NOPRI.....ignore the priority bit when writing the output IFF picture
NOEXT.....don't add .LBM extension to room names when creating output IFF picture filenames
NES.....move colors 4,5,6,7 to 16,17,18,19 ...

Since output filenames are the name of each room with the extension .LBM tacked onto the end, you should use **Room|Set Info...** to make sure your room names are valid DOS filenames.

Note that your tUME map file must include either a TMGC or TMGX chunk. This chunk is included when you select **Project|Save...|Save+TMGC** or **Project|Save...|Save+TMGX**.

Glossary

Here are some terms we use while talking about tUME.

Composite Tileset.....A tileset composed of tiles that are created from other tiles grouped together. E.g., a 16x16 tile created from four 8x8 tiles.

Drag-Select.....To choose several on-screen objects by placing the mouse pointer at one corner of a group of objects, pressing the mouse button, and while keeping the mouse button depressed, move the mouse to the opposite corner of the group of objects, then releasing the mouse button.

Edit Room.....A room that you created in tUME. You can edit an Edit Room.

Map.....This is what tUME ultimately loads and saves. A map contains one or more **tileset**s and one or more **room**s. **Project** is a synonym for map.

NULL Tile.....Any position in a room that contains no tiles. Nothing has been stamped at that position. Though most map editors fill a new room with tile #0, tUME fills a new room with NULL tiles. Since tUME can load multiple tilesets, it has no concept of tile #0. Instead it has the NULL tile in the absence of a tile.

Pane.....Every tUME window has two panes. Each pane may be set to show only certain types of rooms, either source rooms or edit rooms. The default tUME setup shows Source Rooms in one pane and Edit Rooms in the other pane.

Room.....A Room is a rectangular grid of **tile**s, and in tUME it can also be a certain number of tiles 'deep'. tUME has **source room**s, which is where tiles appear when they are loaded into tUME. tUME also has **edit room**s, which are the rooms you create. An edit room may be a single level in a video game. In addition to being used to represent a video game level or a fantasy role-playing map, tUME edit rooms may also be used to define tile attributes.

Room User Number..Room User Numbers are usually used for your own personal reference. In your game you may need to reference a particular room. Sometimes you might do this by using the room's name and other times you might find it more useful to reference a room by its Room User Number.

Room User Type.....Every room may have a User Type. Room User Types are used to tell tUMEPack how to interpret a particular room. Some rooms may be converted into levels for a video game. Others may be converted into attribute tables or collision maps.

Select.....To choose an on-screen object by clicking (pressing and releasing) the mouse button while the mouse pointer is over it.

Source Room.....A room created to show a **tileset**. You cannot edit a Source Room.

Tile.....A **tile** is the smallest unit of graphics you work with. Tiles are rectangular patches of pixels. Tiles graphic may be literal, where what you see is what will appear in the level, or iconic, where it may represent a monster, an attribute, or something else entirely. A collections of tiles is called a **tileset**. tUME can also create **composite tilesets** that are created from other tilesets.

Tile-Brush.....The tiles that are attached to your mouse pointer that you may use to draw into a room or affect the tiles in a room in various ways.

Tileset.....A **Tileset** in tUME is a set of graphics images ('**tiles**') loaded from a DPaint file that can be used to create or fill a room.

Tileset User Number.Every tileset may have a User Number. User Numbers are usually used as a reference for your program or to tell tUMEPack what order to sort your tilesets. See TPNES and M.C. Kids below.

Tileset User Type.....Every tileset may have a User Type. The Tileset User Type is used to tell tUMEPack how to interpret the graphics in a particular tileset. For example, Tileset User Type 0 might tell tUMEPack to convert this tileset into a character font for the SEGA Genesis or the Super Nintendo Entertainment System. Tileset User Type 1 might mean ignore this tileset. Tileset User Type 2 might mean convert these tiles to collision masks. The interpretation of the Tileset User Types is dependent on the particular version of tUMEPack you are using.

tUMEPack.....An external program that takes the maps you create with tUME and converts them to a data format that is usable in your end product. If you are writing a SNES game then tUMEPack would take your maps and write out files that you could assemble with your SNES assembler. If you are writing a MS-DOS based game then a different version of tUMEPack would convert your maps to something suitable for your MS-DOS game. tUMEPack can also create character fonts, collision tables, collision maps, sprite object lists and many, many other data structures.

Window.....One view of a room in tUME.

Error Messages

tUME error messages appear both in the status bar and in dialog boxes. Sometimes, two error messages will appear at once (one in the status bar, the other in the dialog box)! In this case, the message in the status bar will generally be more specific, and will reveal more about the nature of the error.

Sometimes your palette will be set such that you will be unable to read an error message (the letters in the status bar are the same color as the background). In these cases, you may not be able to tell that an error has occurred. Pressing [Alt]-p will toggle between your palette colors and special colors that make the status bar visible.

Can't add layer to a composite room.

Appears when you try to add a layer to a composite room. Composite source rooms may have only one layer. See **Composite Tiles**.

Can't add layer to a locked room.

Appears when you try to add a layer to a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't add layer to a tileset room.

Appears when you try to add a layer to a source room. Source rooms may have only one layer.

Can't change locked composite room stats.

Appears when you choose **Room|Set Info...** on a locked composite room. You must first unlock the room before you can change the composite room size.

Can't change stats on a tileset room.

Appears when you choose **Room|Set Info...** on a source room. Choose **Tiles|Set Info...** instead to change source room information.

Can't clear a locked room.

Appears when you try to clear a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't count characters.

Appears when you try to count characters. There is not enough memory to complete this operation.

Can't delete a composite room.

Appears when you choose **Room|Delete** on a source composite tile room. You are not allowed to do this. If you want to delete the composite tileset, and all references to the composite tiles, then choose **Tiles|Delete...**. See **Composite Tiles**.

Can't delete a locked room.
Appears when you choose **Room|Delete...** on a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't delete a tileset room.
Appears when you choose **Room|Delete...** on a source room. You are not allowed to do this. If you want to delete the source tileset, and all occurrences of the source tiles, then choose **Tiles|Delete...**. See **Deleting a Tileset**.

Can't delete layer from a composite room.
Appears when you try to delete a layer from a composite room. Composite source rooms must have one layer. See **Composite Tiles**.

Can't delete layer from a locked room.
Appears when you try to delete a layer from a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't delete layer from a tileset room.
Appears when you try to delete a layer from a source room. Source rooms must have one layer.

Can't delete locked layer.
Appears when you try to delete a locked layer. Make sure the layer is unlocked by choosing **Layer|Lock** or pressing l.

Can't delete the last layer.
Appears when you try to delete the last layer in a room. Rooms must have at least one layer.

Can't find/make 'tumepack' directory.
Appears when you try to Xave a map. tUME was unable to create the directory to place all the files to Xave. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in the Xave dialog box.

Can't find or make tileset directory.
Appears when you try to Xave a map. tUME was unable to create the directory to save the individual tileset images. A DOS error may have occurred (e.g., the drive may be full), or you may have changed your source tileset filename to an invalid DOS path name.

Can't find target hardware.
Appears when you try to download a room to a target development system. Make sure the development system is connected to the PC, make sure the map downloader program is running on the target system, and make sure the target system is receiving data correctly (you may need to reset the target system). See **Downloading to a Development System**.

Can't find tileset 'picture_name'.
Appears when you try to load a tileset and the file does not exist.

Can't GridRoomAsTiles on a tileset room.
Appears when you try to turn a source room into a composite room. You must first create an edit room, fill it with the tiles you wish to composite, and then choose **Tiles|GridRoomAsTiles...** on your edit room. See **Composite Tiles**.

Can't initialize this room!
Appears when you try to clear a room. tUME ran out of memory while trying to clear this room.

Can't insert layer to a composite room.
Appears when you try to insert a layer to a composite room. Composite source rooms may have only one layer. See **Composite Tiles**.

Can't insert layer to a locked room.
Appears when you try to insert a layer to a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't insert layer to a tileset room.
Appears when you try to insert a layer to a source room. Source rooms may have only one layer.

Can't load layer to a composite room.
Appears when you try to load or append a layer to a composite room. Composite source rooms may have only one layer. See **Composite Tiles**.

Can't load layer to a locked room.
Appears when you try to load or append a layer to a locked room. Make sure the room is unlocked by choosing **Room|Lock**.

Can't load layer to a tileset room.
Appears when you try to load or append a layer to a source room. Source rooms may have only one layer.

Can't load '*tileset*'! Would you like to try a different file?
Appears when tUME cannot find a tileset file. Click **Yes** to specify a different file. See **Configuring Tileset Search Path** section of the *tUME Configuration Guide*.

Can't save a null room.
Appears when you try to save a null room. tUME won't save non-existent rooms.

Can't seem to load map '*filename*'.
Appears when you try to load a map. Most likely, tUME cannot find *filename*.

Can't seem to put this tileset in a room.
Appears after a tileset has been loaded. Most likely, tUME has run out of memory.

Can't seem to save map.
Appears when tUME is unable to save the map. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in the save map dialog box.

Can't seem to save tile images.
Appears when you try to export tiles as brushes, or when you try to Xave a map. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in the save dialog box.

Couldn't create palette file.
Appears when you try to save a palette or palette range. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in the save dialog box.

Couldn't open printer file!
Appears when you try to print a room. The path you have specified on the line '**PrintTo=**' in the group **[Print Maps]** of the tUME.INI file is invalid.

[COLOR MENU] section not found in .INI file.
The **[COLOR MENU]** section is missing from the tUME.INI. Please see *tUME Configuration Guide*.

Color requester not active.
Appears you you try to load/save a palette/palette range and the color requester is not active. Please activate the color requester first.

Composite tile height must be less than or equal to room height.
Appears when you try to make a composite tileset where the tile height of a composite tile is taller than the current room height. Either re-size the room so it is taller, or make your composite tiles less tall.

Composite tile width must be less than or equal to room width.
Appears when you try to make a composite tileset where the tile width of a composite tile is wider than the current room width. Either re-size the room so it is wider, or make your composite tiles less wide.

Downloading disabled.
Appears when you try to download a room to a target development system. Make sure that downloading has been enabled in the tUME.INI file (i.e., there is a line that says '**Enable=1**' in the group **[Download]**). See **Downloading to a Development System**.

Error saving tile image.
Appears when you try to Xave a map. tUME was unable to save the individual tile images as ILBM files. A DOS error has probably occurred (e.g., the drive may be full).

layer types: '=' not found.

A line in the layer types section of tUME.INI does not contain the symbol "=". See **Redefining Layer Types** in the *tUME Configuration Guide*.

layer types: bad syntax.

A line in the layer types section of tUME.INI is not formed properly. See **Redefining Layer Types** in the *tUME Configuration Guide*.

layer types: keyword 'Layer' not found.

A line in the layer types section of tUME.INI does not contain the keyword "Layer". See **Redefining Layer Types** in the *tUME Configuration Guide*.

Memory critical, reverting to last display.

Appears when you try to view the next or the previous room. You are dangerously low on memory.

No room exists here.

Appears when you try to copy palettes. You must select a room for tUME to copy the palette to. See **Room Colors**.

No Search Buffer specified.

Appears when you try to search or search and replace without first specifying a Search Buffer. See **Search and Replace**.

No tileset selected for deletion.

Appears when you try to delete a tileset, and you have not selected a tileset to delete. First, make the room displaying the source tileset visible, then select a tile-brush that includes a tile from the tileset you want to delete in the upper left corner of the brush, then choose **Tiles|Delete...**.

No tiles for composite tileset.

Appears when you try to convert a room without any tiles in it into a composite room. You must fill your edit room with the tiles you wish to composite before choosing **Tiles|GridRoomAsTiles...** See **Composite Tiles**.

No tiles selected for setting tileset info.

Appears when you try to set tileset information, and you have not selected a tileset to apply the function to. First, select a tile-brush that includes a tile from the tileset you want to set in the upper left corner of the brush, then choose **Tiles|Set Info...**

Not enough memory to load map.

Appears when you try to load a map. tUME has run out of memory. See **Troubleshooting: tUME says I'm out of memory, and I know I have plenty of memory**.

Out of memory loading tileset.

Appears when you try to load a tileset. tUME has run out of memory.

Please enter a unique room name.

Appears when you are trying to create a composite room. Either you have failed to give your new composite tileset a name (the topmost long box is empty), or the name you have given to your new composite tileset is the same as another source or edit room. Change the name so it is unique. See **Composite Tiles**.

Please select a block first.

Appears when you try to copy palettes. You must first create a tile-brush (press b and drag-select some tiles) to indicate to tUME which room to copy the palette from. See **Room Colors**.

Please select a palette range.

Appears when you try to save/load a palette range. You must select a range from the palette first. Note that you may not select a color cycle range.

Range is out of bounds.

Appears when you try to load a palette range. You have specified a invalid initial color number.

Room is already a composite tileset.

Appears when you try to convert a composite room into a composite room. To create a composite tileset made of smaller composite tiles, you must first create an edit room, fill it with the tiles you wish to composite, and then choose **Tiles|GridRoomAsTiles...** on your edit room. See **Composite Tiles**.

Room may only have one layer.

Appears when you try to convert an edit room with several layers into a composite room. You may convert single-layered edit rooms into composite rooms. See **Composite Tiles**.

Search Buffer has different size tile.

Appears when you try to search or search and replace and the Search Buffer contains different size tiles than the current room. See **Search and Replace**.

Tile image is too large.

Appears when you try to load a tileset. The width times height size you have selected is too large for tUME to handle. Consider using a smaller tile size in conjunction with composite tiles.

Trouble allocating new layer.

Appears when you try to add a layer to a room. Most likely, tUME has run out of memory.

Trouble loading tileset.

Appears when you try to load a tileset. Either a DOS error has occurred, or tUME has run out of memory.

Trouble saving picture.

Appears when tUME is unable to save the current screen as an IFF picture. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in

the save picture dialog box.

Trouble saving room.
Appears when tUME is unable to save the room. A DOS error may have occurred (e.g., the drive may be full), or you may have entered an invalid DOS path name in the save room dialog box.

Try to load other 'problem' tilesets from here?
tUME is asking if it should add the sub-directory of the last tileset it loaded to the list of sub-directories to search when loading the other tilesets. See **Configuring Tileset Search Path** section of the *tUME Configuration Guide*.

Unable to create room.
tUME has run out of memory while trying to create a new room.

Unable to re-size room.
tUME has run out of memory while trying to re-size the room.

Unable to select block.
Appears when you try to select a tile-brush. tUME has run out of memory.

Unable to set Search Buffer.
tUME has run out of memory while trying to set the Search Buffer.

tUME

the Universal Map Editor

Source Code Overview

Confidential Proprietary Information

Dan Chang

Introduction

This overview of the tUME source code is designed to aid someone in modifying and enhancing tUME. It also describes enough of the workings of tUME to enable someone to add a new feature to tUME.

Building tUME

To build tUME, you need the Borland C++ 3.1 package; use the command make -a to invoke Borland's MAKE on the file MAKEFILE. Note that there are some environment variables that need to be set by calling the batch file TUMEVARS.BAT. Alternatively, use the batch file BUILD.BAT to delete all the old object modules, set the environment variables, and re-build tUME.

There are three header files that define compile-time switches. The file switches.h contains mostly debugging switches and such; for the most part we leave it alone.

The file switch1.h controls whether to build a demonstration version or a normal version.

The file switch2.h controls whether or not to include the downloading code, and which SNASM to check for.

The file license.h contains the tUME licensee's name.

The file version.c contains the version number. The version number is automatically incremented by BUMP.EXE everytime you build tUME.

The batch file SAVEIT.BAT will save the tUME source code in a zip file. The batch file SAVETOOL.BAT will create the zip file tUMETOOL.ZIP found on the tUME executables disk.

The batch file BPIT.BAT calls BRUSHPAK to process all the graphics in tUME (such as radio buttons, check buttons, graphic fonts, mouse pointers, etc.) to create the files EUIBPI.BPI and TUMEBPI.BPI. BRUSHPAK converts *.LBM files to some data structure that tUME understands.

The batch file FPIT.BAT calls FILEPACK to process the *.BPI files to create the file tUME.FPF which is loaded by tUME.EXE. FILEPACK collects all the various data bits used by a program, packs the data bits, and puts them together in one file.

Object Modules

tUME is built using Borland's VROOM technology. All object modules required to build tUME are listed in the MAKEFILE. There are four object module suffixes used in the MAKEFILE; they control how the object module is compiled:

*.OBJ.....Compile for smallest size; also assembly language module

*.OBF.....Compile for best speed
*.OVJ.....Compile overlaid module for smallest size
*.OVF.....Compile overlaid module for best speed

Note that all overlaid modules should be listed in the group OVDEP, and all non-overlaid modules should be listed in the group PDEP.

Main Event Loop

tUME is an event-driven program. The main loop is found in the file tUME.c, in the function main(), and looks like:

```
while (!ExitProgram) {
        ...
}
```

Inside this loop, the program calls ReadMouse() to read the mouse, calls HandleMenus() to process the menus, calls HandleKeys() to process any key presses, and calls FN_DontWait() to process mouse movements and mouse button presses on screen areas other than the menu bar.

The menus are defined in tUME.INI, and the code to display the menus and process the menus is found in the EGGS (Echidna Game Generation System) library EUILIB. When the user chooses an item on the menus, it generates the corresponding event specified in tUME.INI. We look up this event in the first column of events.e to determine the corresponding C function to call, which is listed in the second column. The menu code performs the lookup in events.c.; we don't need to concern ourselves with events.c, as it is generated by the event compiler EVNTCOMP.EXE when we pass events.e as the input file.

E.g., the user chooses **Project|Load...** to load in a tUME map. By examining the tUME.INI file, we determine that the event generated by this menu choice is LoadMap. We search the events.e file for the LoadMap event, and determine that the corresponding C function to be called is LMap(). Thus, when the user chooses **Project|Load...**, the event-handler function LMap()gets called by HandleMenus().

tUME events are documented in the *tUME Configuration Guide*.

Most of the event-handler functions are found in menuitem.c, though a few of them are found in other modules.

Key presses are treated in a similar fashion: the keys and their corresponding events are defined in tUME.INI. We look up the event in events.e to find the corresponding event-handler function to call. E.g., the user presses [Spacebar]. Examining the tUME.INI file, we find that the corresponding event generated is FlipPanes. We search events.e for the corresponding event-handler for FlipPanes, which is WFRoom(). Thus, when the user pressed [Spacebar], the event-handler function WFRoom() gets called by HandleKeys().

FN_DontWait is a pointer to function. It may be NULL, or it may point at Drawing(), Hovering(), Selecting(), or Tracking(). When the user is not holding a tile-brush, Hovering() gets to process the mouse events; when the user is holding a tile-brush, Tracking() gets to process the mouse events; when the user is drag-selecting a tile-brush, Selecting() gets to process the mouse events; and when the user is drawing with a tile-brush, Drawing() gets to process the mouse events.

Initialization

The tUME.INI file is processed by ProcessINI() to set certain global variables, thus defining some of tUME's behavior. The events in the [Initial Events] section are processed by ProcessInitialINIEvents(). Some of the EGGS library need to be initialized as well; these initializations are performed by calling OpenEUI(), OpenDBufGraphics(), and InitFileReqs().

Some sections of the tUME.INI file, such as [Zoom Events] and [Cursor Movement Events], define new events. These events are dynamically allocated, and added to the list of events by AddEmptyEvents(). All events of a class point to one event handler; e.g., all zoom events point to SetZoomEvent(), and all cursor movement events point to MoveCursorEvent(). It is the responsibility of the class event handler to determine which specific zoom (or cursor, or...) event occurred.

Note that the *tUME Configuration Guide* is useful in understanding the contents of the tUME.INI file.

Data Structures

The majority of tUME's data structures are defined in tudef.h. We consider each object in tUME, and talk about the data structures used to represent it.

Tiles

Tiles are represented by the data structure PlotType, and have three attributes associated with them:

UBYTE Plot_Flags...which contains flip, priority, and colorset information;
UBYTE TileSet_ID..which tileset this tile belongs to; and
UWORD Tile_ID.….which tile in the tileset is this tile.

TilesSet_IDs are number starting from 1, and Tile_IDs are also numbered starting from 1.

Layers

A layer is represented by the data structure LayerType. In a linear memory machine (such as the Amiga), a layer is basically an array of PlotTypes. However, in paged memory architectures, a layer is basically a RGRGPLT. A RGRGPLT, also defined in tudef.h, is basically a MPYTMPXTPT, along with the width and height of the layer. A MPYTMPXTPT is a pointer to an array of MPXTPT. A MPXTPT is just a synonym for an XTRAPntr. An XTRAPntr is a pointer to XTRA memory; see the MEMLIB docs.

The layers of a room are stored in a linked list Layers. The RoomType element LayerType *FloorLayer points to the node in the linked list that represents the current floor.

Layers are created by roomio#AddLayer(), and destroyed by roomio#DeAllocateLayer() and roomio#DeAllocateLayers().

Rooms

A room is represented by the data structure RoomType. The linked list Layers contains the layers in the room; the structure ColorInfo *R_ColorInfo contains the palette information for the room.

All the rooms in tUME are stored in the linked-list contained in the data structure MapType *GlobalMap.

Tile-Brush

The tile-brush is represented by the data structure BlockCopyType. The linked list Layers contains the layers of the tile-brush; the elements RoomWindowType *SourceRW, RoomStuffType *SourceStuff, RoomType *SourceRoom, WORD SourceX, and WORD SourceY define where the tile-brush was selected.

While the user is dragging the tile-brush around, the elements RoomWindowType *DestRW, RoomStuffType *DestStuff, WORD DestX, and WORD DestY define where the tile-brush is about to be pasted.

Tilesets

A tileset is represented by the data structure TileSetType.

The macro FAST_TILESET_PTR takes a Tileset_ID and returns a pointer to the TileSetType data structure that contains information for that particular tileset ID. The functional prototype for this macro would be something like this:

```
extern TileSetType *FAST_TILESET_PTR(UBYTE TileSet_ID);
```

To convert from a TileSetType pointer back to a TileSet_ID, look up the element WORD TS_id in the TileSetType data structure.

All the tilesets in tUME are stored in an array contained in the data structure TileSpaceType *GlobalTileSpace.

Palettes

A palette is represented by the data structure ColorInfo, which is found in colorseq.h. Since the color information structure is fairly large (about 6K!), and since there is one for every room, they are kept in XTRA memory. The palette requester is designed to work data structure in main memory, so when it starts, it makes a copy of the colors into main memory.

Adding an Event

Here are step by step instructions for adding a new event to tUME:

1. Give the event a name, e.g., ToggleCoolNewEvent.

2. Add a line to represent the event in the events.e file, e.g., "ToggleCoolNewEvent" ToggleCoolNewEventHandler ST=CoolNewEventSTATE CHECK TOGGLE;

The text in the first column in quotes is the name of this event ("ToggleCoolNewEvent"). The text in the second column is the name of the event handler (ToggleCoolNewEventHandler). This is the C function that gets called when this event is triggered. The rest of the line says that this event has a check box (CHECK), and that it TOGGLEs the state variable CoolNewEventSTATE between FALSE and TRUE.

3. Define the state variable in a C module. Either create a new module, or lump it into menuitem.c. E.g., short CoolNewEventSTATE = FALSE;. This sets the state variable initially FALSE.

4. Define the event handler function in a C module. Either create a new module, or lump it into menuitem.c. Make it a function return a short and taking no arguments, e.g., short ToggleCoolNewEventHandler(void) { .. }.

Look at the event SpaceToggle for an example of an event implement using the above four steps.

Note that it is possible to define mutually exclusive events with radio buttons in the menus. See the events SetStampPaint and SetStampReplace for an example of how to do this. If you want to define mutually exclusive events, but you don't need the radio buttons, you only need to specify the event name and the C function event handler. E.g., the events RoomStatus, TileStatus, UserStatus, and CursorStatus can be considered mutually-exclusive, but they do not have radio buttons that reflect their mutually-exclusive status.

Modules Overview

Here we examine some of the algorithms used in some of the modules of tUME.

tumedraw.c

This is the heart of tUME.

The function StampTile() figures out the scaling and draws the actual tile. It will also recursively call itself to draw a composite tile. The routine calls one of the assembly language routines, either MCGA_ClippedMaskedCopyTransRect() or MCGA_ClippedScaledMaskedCopyTransRect() to actually draw the tile.

Note that rooms are redrawn one layer at a time. When Scroll() scrolls the screen, the unchanged area is block copied, and the new area is re-drawn by calling ShowRoomRectLayer().

download.c

The function InitDownloader() includes the calls to check for the presence of SNASM hardware.

The characters in the current layer are converted to Super Nintendo or Genesis characters. The characters are downloaded, the palette for the current room is downloaded, the map for the current layer is downloaded, and a header block defining the size of the map and other relevant information is downloaded.

The characters are collected into an array. The characters are collected by spiralling out from the current pointer position on the map. It is performed in this fashion to ensure the the map immediately surrounding the pointer utilizes the majority of the characters, and the map far away may be less detailed (as the character set may be filled by then).

If you wanted to speed up this routine, you should modify it so that it scans the layer from left to right, then from top to bottom. Activate an entire row of XTRA memory at once, the process all tiles in that row before proceeding to another row.

EGGS Library Overview

The EGGS (Echidna Game Generation System) library contains routines and data structures that are useful to the development of games on the IBM PC. The libraries are divided by function, and the library files are found in the C:\EGGS\LIB\ECHIDNA directory. The source for each library is found in a sub-directory with the same name (e.g., C:\EGGS\LIB\ECHIDNA\EUILIB). The header files for each EGGS library are found in the directory C:\EGGS\INCLUDE\ECHIDNA. The EGGS libraries used in tUME are:

BEIFFLIB.....IFF readers
EUILIB.....menu and keyboard support routines
GFXLIB.....MCGA routines
LDSLIB.....linked list routines
MEMLIB.....EMS and XMS support routines
MISCLIB.....miscellaneous support routines
TIMERLIB.....periodic interrupt support routines
XPAKLIB.....packed data loading routines

To use an EGGS library, merely link it into your program. tUME modifies some of the functionality of some of the EGGS libraries; this is accomplished by placing a local copy of the appropriate C source file from the library in the tUME directory, making the modifications to the local copy of the source file, and adding the C file to the list of modules to compile in the MAKE file.

To build an EGGS library, enter the sub-directory that contains the source code for that library, and execute the batch file MAKEIT.BAT found therein.

Some libraries, such as BEIFFLIB and EUILIB, are "monolithic"; they contain lots of code just to do one or two things. Other libraries, such as LDSLIB and MEMLIB, and more "granular"; they contains lots of little functions.

To understand the tUME source code, and to effectively make additions and modifications to tUME, it is important to have a firm understanding of LDSLIB and MEMLIB. Please refer to the documentation for those two EGGS libraries. While it is nice to study the other EGGS libraries as well, it is not required; study them on an "as needed" basis.

BEIFFLIB

The EGGS BEIFFLIB (Big, Easy IFF library) contains the support code necessary to load IFF ILBM files and IFF tUME files. The Big part of the name refers to the fact that this library loads the data into EMS or XMS memory provided by MEMLIB.

EUILIB

The EGGS EUILIB (Easy User Interface library) contains the support code necessary to display

the windows, display the control gadgets (list boxes and elevators), display the menus, process the user's menu choices, and process the user's key presses. EUILIB also contains the standardized file requester.

GFXLIB

The EGGS GFXLIB (Graphics library) contains the support code necessary to draw on the IBM PC MCGA screen.

LDSLIB

The EGGS LDSLIB contains the linked list and binary tree support code.

MEMLIB

The EGGS MEMLIB contains the EMS and XMS memory support code.

MISCLIB

The EGGS MISCLIB contains miscellaneous support routines, such as the error reporting routines, the easy input/output routines, the INI file reading routines, the easy C strings routines, and the exit clean-up routines.

TIMERLIB

The EGGS TIMERLIB contains routines to re-program the IBM PC periodic interrupt, and provides for several concurrent interrupts at different periods.

XPAKLIB

The EGGS XPAKLIB contains routines to load FPF (FilePack'ed) files, and other support routines.

Function Locations and Brief Descriptions

The following is a list are some of the key functions in tUME and in the EGGS library, the source file where they are found, and a brief description of each function. Non-inclusion in this list does not mean the function is un-important. Use Borland's GREP.EXE to find other functions that are not included in the following list. Remember: GREP is your friend when it comes to understanding tUME.

AddLayer().....roomio.c: create a new layer and add it to a list of layers
DeAllocateLayer()….roomio.c: free memory used by a single layer
DeAllocateLayers()...roomio.c: free memory used by all layers & free layers linked-list
Drawing().....mitems.c: process events when user is drawing with the tile-brush
HandleKeys().....euilib\keyevent.c: reads keys and calls appropriate event-handler functions
HandleMenus().....euilib\menus.c: processes and calls appropriate event-handler functions
Hovering().....mitems.c: process cursor moving around with no tile-brush attached
FN_DontWait()……mitems.c, points to Drawing(), Tracking(), Selecting(), and Hovering()
LMap().....menuitem.c: loads in a tUME map
ParseINI().....parseini.c: read and process the tUME.INI file
ProcessInitialINIEvents() parseini.c: process [Initial Events] in the tUME.INI file
ReadMouse().....ibmmouse.c: read the mouse position and mouse buttons
Selecting().....mitems.c: process user drag-selecting a new tile-brush
Scroll().....tumedraw.c: re-draws screen when user scrolls around
ShowRoom().....tumedraw.c: re-draws screen to show current room
ShowRoomRectLayer() tumedraw.c: draws part of one room layer on screen
StampTile().....tumedraw.c: draws one properly scaled tile on screen
Tracking().....mitems.c: process tile-brush moving around, mouse buttons not pressed
WFRoom().....menuitem.c: display the other pane on screen

Compile Time Switches and Brief Descriptions

Here are some of the compile time switches used in tUME and what they mean:

PLOTARRAY.....defined if linear memory, currently left undefined (paged EMS & XMS memory)
fDemoBanner.....switch1.h: set to 1 if you want to build a demo version of tUME
fDoSaveRooms.....switch1.h: set to 0 if you want to build a demo version of tUME
dvpSNASM.....switch2.h: set to 1 to include SNASM downloading code
dvpNONE.....switch2.h: set to 1 to not include SNASM downloading code
fCheck.....switch2.h: set to fCheckNone, fCheckSNASM, fCheckGeneral, or fCheckBoth

tUME

the Universal Map Editor

Programmer's Guide

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.

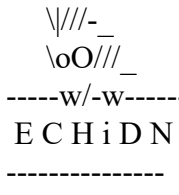
The final map is written to a binary file by using the name of the room and adding the extension '.BMP' for Binary MaP. The BLOCK descriptions are separated into two files. One file, the '.BLK' file contains the character part of the BLOCK descriptions, 4 characters per tile. The second file, the '.FLR'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



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}

tUME

the Universal Map Editor
Configuration Guide

Gregg A Tavares

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

Configuring tUME

tUME.INI

The tUME.INI file holds information for configuring tUME to your particular needs. Usually, we will supply a tUME.INI file that is setup for your particular needs, but, if for some reason you'd like to change the tUME.INI file for some other needs, or if you just find that you'd like to change the menus or the keys to something you are more comfortable with, then here is how to do it.

Setting Initial Global States

You may set the initial state of global variables, such as menu bars visible/invisible, tile-brush visible/invisible, stratify paste on/off, etc., by specifying a list of events that toggle the state of the particular global variable when tUME starts up.

The list of initial events are specified in the [Initial Events] section of tUME.INI. All global events default to the off state, so to turn them on, add them to this section.

E.g., to make tUME default to stratified pasting, add the following line to the [Initial Events] section:

```
ToggleStratifyPaste
```

Redefining Room Types

Room types define how tUMEPack interprets a particular room, whether to treat a particular room as a level graphics room, as a conversion room, etc. They are listed when you select **Room| Set Info...** They may be re-defined to suit your particular application. If you change the room types, you'll want to change your tUMEPack program to match.

In the tUME.INI file, there is a section called [Type Groups] with three lines in it:

```
TileType=MCKids 2 Tileset Types
RoomTypes=MCKids 2 Room Types
LayerTypes=MCKids 2 Layer Types
```

RoomTypes are described here, TileTypes are described in the section **Redefining Tileset Types**, below, and LayerTypes are described in the section **Redefining Layer Types**, below. The string following the RoomType= defines the name of the section to look in for the list of room types. You may have several room type lists in the tUME.INI file, and select one just by changing the label following the RoomType=. Continuing with this example, there is a section called [MCKids 2 Room Types] that looks like this:

```
[MCKids 2 Room Types]
Level Room=0
Table Room=1
Picture Room=2
```

Parallax Room=3
Mode 7 Room=4
Flat Conv Room=5
Layered Conv Rm=6

This section lists all the room types that are defined. The text that appears to the left of the '=' is the name of the room type. The number following the '=' is the room type. This is the number that tUMEPack looks at to determine how to deal with a particular room.

Redefining Layer Types

Layer Types are currently only used by the **Smart Flip** feature. Layer Types define which tileset user types are associated with each layer. Thus, when you press [Spacebar] while **Smart Flip** is enabled, tUME will find the last source room with the same tileset user type for the floor layer.

In the tUME.INI file, there is a section called [Type Groups] with three lines in it:

TileType=MCKids 2 Tileset Types
RoomTypes=MCKids 2 Room Types
LayerTypes=MCKids 2 Layer Types

LayerTypes are described here, TileTypes are described in the section **Redefining Tileset Types**, below, and RoomTypes are described in the section **Redefining Room Types**, above. The string following the LayerType= defines the name of the section to look in for the list of layer types. You may have several layer type lists in the tUME.INI file, and select one just by changing the label following the TileType=. Continuing with this example, there is a section called [MCKids 2 Layer Types] that looks like this:

[MCKids 2 Layer Types]
Layer 1=0,4,5,6,7,8
Layer 2=1
Layer 3=2
Layer 4=3
Layer 5=0,4,5,6,7,8
Layer 6=1
Layer 7=2
Layer 4=3

This section lists all the layers that are defined. The text to the left of the '=' is the layer number being defined. The comma-separated list following the '=' lists all tileset user types that may be pasted into the layer on the left.

Thus when **Smart Flip** is enabled, and the floor is layer 2 of an edit room, pressing [Spacebar] will display the source room that contains the most recently selected tiles of user type 1 (defined as **Contour Tiles** in the next section). Or if the floor is layer 1 of an edit room, will display the most recently selected tiles in a source room containing tiles of user type 0, 4, 5, 6, 7, or 8.

Redefining Tileset Types

Tileset types define how tUMEPack interprets a particular tileset, whether to treat a particular tile as graphics, as contour information, or as object type, etc. They are listed when you select **Tiles| Set Info...** They may be re-defined to suit your particular application. If you change the tileset types, you'll want to change your tUMEPack program to match.

In the tUME.INI file, there is a section called [Type Groups] with three lines in it:

TileType=MCKids 2 Tileset Types
RoomTypes=MCKids 2 Room Types
LayerTypes=MCKids 2 Layer Types

TileTypes are described here, RoomTypes are described in the section **Redefining Room Types**, above, and LayerTypes are described in the section **Redefining Layer Types**, above. The string following the TileType= defines the name of the section to look in for the list of tileset types. You may have several tileset type lists in the tUME.INI file, and select one just by changing the label following the TileType=. Continuing with this example, there is a section called [MCKids 2 Tileset Types] that looks like this:

[MCKids 2 Tileset Types]
Image Tiles=0,1,11111111
Contour Tiles=1,2,xxx11111
Special Tiles=2,3,xxx11111
Object Tiles=3,4,xxx11111
4-Color Tiles=4,1,11111111
256-Color Tiles=5,1,11111111
Mode 7 Tiles=6,1,xxxxxxxx
Global 16 Tiles=7,1,11111111
Logic Tiles=8,1,xxxxxxxx

This section lists all the tileset types that are defined. The text that appears to the left of the '=' is the name of the tileset type. Three fields follow the '=' separated by commas; the first field is required, the other two are optional. The first field following the '=' is the Tileset Type. This is the number that tUMEPack looks at to determine how to deal with a particular tileset.

The second comma-separated field following the '=' specifies the stratify paste layer. Thus, in the above example, if stratify paste is turned on, then Image Tiles, 4-Color Tiles, 256-color Tiles, and Mode 7 Tiles will be pasted into the first layer, Contour Tiles will be pasted into the second layer, Special Tiles into the third layer, and Object Tiles into the fourth layer.

The third comma-separated field following the '=' specifies which tile flag bits are enabled for this particular tileset. In the above example, with the exception of Contour Tiles, all tile flag bits are enabled. For the Contour Tiles, the x's in bit 5, 6 & 7 positions mean that bits 5, 6 & 7 do not get set for that tileset. The current definition for bit 7 is priority bit, bit 6 is X Flip tile, and the definition for bit 5 is Y Flip tile. Please see **Redefining Tile Flip Bits**, below.

Redefining Tile Flip Bits

Every tile in the map has a tile flag byte associated with it. These eight bits may be redefined for each individual tile in the map. Currently, they are used to represent tile X-flipping, Y-flipping, and colorset information. By default, bit 6 is used to represent an X-flipped tile, and bit 5 is used to represent a Y-flipped tile. The relevant configuration lines look like this:

```
[X Flip Bit]
Enable=x1xxxxxx

[Y Flip Bit]
Enable=xx1xxxxx
```

If you change the enable so there are no 1 bits, like this:

```
[X Flip Bit]
Enable=xxxxxxxx
```

then X-flip is globally disabled.

Redefining Colorsets

Different target hardware platforms have different colorset definitions. You can change the way tUME deals with colorsets. Associated with every tile in the map is a tile flag byte. These eight bits may be redefined for each individual tile in the map. Currently, they are used to represent tile X-flipping, Y-flipping, and colorset information. Below are the relevant lines from the tUME.INI file that deal with colorsets.

Example tUME.INI colorset sections

```
[Color Mask Groups]
16-Bit Nintendo Color
16-Bit Nintendo Priority

[Tile Mask Events]
ClearColor=0xxxx000,0xxxx000
 SetColor0=1xxxx000,0xxxx000
 SetColor1=1xxxx001,0xxxx000
 SetColor2=1xxxx010,0xxxx000
 SetColor3=1xxxx011,0xxxx000
 SetColor4=1xxxx100,0xxxx000
 SetColor5=1xxxx101,0xxxx000
 SetColor6=1xxxx110,0xxxx000
 SetColor7=1xxxx111,0xxxx000
   SetPri=xxx1xxxx,xxx0xxxx
 ClearPri=xxx0xxxx,xxx0xxxx
```

```
[8-Bit Nintendo Color]
Enable=1xxxxxxx
xxxxxxx0:xxxxx0xx
xxxxxxx1:xxxxx1xx
xxxxxx0x:xxxx0xxx
xxxxxx1x:xxxx1xxx

[8-Bit Sega Color]
Enable=1xxxxxxx
xxxxxxx0:xxx0xxxx
xxxxxxx1:xxx1xxxx
xxxxxx0x:xx0xxxxx
xxxxxx1x:xx1xxxxx
xxxxx0xx:x0xxxxxx
xxxxx1xx:x1xxxxxx
xxxx0xxx:0xxxxxxx
xxxx1xxx:1xxxxxxx

[16-Bit Nintendo Color]
Enable=1xxxxxxx
xxxxxxx0:xxx0xxxx
xxxxxxx1:xxx1xxxx
xxxxxx0x:xx0xxxxx
xxxxxx1x:xx1xxxxx
xxxxx0xx:x0xxxxxx
xxxxx1xx:x1xxxxxx

[16-Bit Nintendo Priority]
Enable=xxx1xxxx
xxx1xxxx:1xxxxxxx
```

The first section [Color Mask Groups] specifies which Color Mask Groups are active. The next two lines are the name of the sections to use, and are names chosen by the user to describe each color mask group. The above example says that both the '16 bit Nintendo Color' group and the '16-bit Nintendo Priority' group are active.

Now, look down at the [16-bit Nintendo Color] section. The first line

```
ENABLE=1xxxxxxx
```

means that if a particular tile's flags have bit 7 set then affect them with this Color Mask Group.

To understand how colorsets work you need to know somethings about tUME. Every tile in a tUME map has a one byte flag. These flags are used to track colorsets, priority and X and Y flipping. Also, tUME draws it's graphics in a 256 color mode. This mode uses one byte per pixel.

In the above example, if a particular tile's flags has bit 7 set then when each pixel of that tile is

drawn on the screen in tUME each pixel's value will be affected by the '16-Bit Nintendo Color' group. The way the pixels will be affected is defined by the following lines:

```
xxxxxxx0:xxx0xxxx
xxxxxxx1:xxx1xxxx
xxxxxx0x:xx0xxxxx
xxxxxx1x:xx1xxxxx
xxxxx0xx:x0xxxxxx
xxxxx1xx:x1xxxxxx
```

The first line says, if bit 0 of the tile's flags is 0 then bit 4 of the tile's pixels should be set to 0. The second line says, if bit 0 of the tile's flags is set to 1 then bit 4 of the tile's pixels should be set to 1. Putting all these lines together says set bits 4 thru 6 of every pixel in this tile to the same bits as bits 0 thru 2 of this tile's flags. This means a particular tile will be drawn in either the first 16 colors, the second 16 colors, the third 16 colors on up to the eighth set of 16 colors.

Now, in our example we have two Color Mask Groups active. The other group '16-Bit Nintendo Priority' says:

```
Enable=xxx1xxxx
xxx1xxxx:1xxxxxxx
```

The first line says: if bit 4 of this tile's flags is 1, then (the second line says) if bit 4 of this tile's flags is 1, then set bit 7 of this tile's pixels to 1. (No, I did not stutter in the last sentence. In this Color Mask Group, the same bit (number 4) is used for two things.)

This is all fine and dandy, but how do we affect the actual tile flags? Well, you create Tile Mask Events in the [Tile Mask Events] section. Look at the one below. The First line

```
ClearColor=0xxxx000,0xxxx000
```

defines a new Event. This is an event just like all the Events listed at the end of this document. You may assign it to a key or to a menu item. When you choose this event you put tUME into Color mode. In Color mode, when you draw in a room the tiles you draw over will have their tile flag bits affected, and nothing else. They will be affected as specified by the Event in the TUME.INI. The two bitmasks defines the action for the left mouse button, followed by the action for the right mouse button. For the ClearColor event, the left and right mouse button have the same effect; each tile's flag will have its bit 7, bit 2, bit 1 and bit 0 set to 0 and the other bits will be left unchanged.

```
SetColor0=1xxxx000,0xxxx000
```

For the above SetColor0 event, pressing the left mouse button will set each tile's flag bit 7 to 1 and reset bits 2, 1 and 0 to 0. Pressing the right mouse button will reset each tile's flag bit 7, 2, 1, and 0 to 0.

These new tile flag bits will now be used when drawing the tiles. Of course as always it is up to tUMEPACK to interpret these bits and turn them into something useful for your project.

Configuring Search and Replace

As supplied, when tUME is counting brushes, searching, or replacing, it tries to match tiles exactly; i.e., it searches for tiles with the same flipping, same priority, and same colorset. The FlagBits= line in the group [Search Options] defines which tile flag bits to match. Thus if the tile flag bit 4 is the priority bit, then the following lines will ignore tile priorities when searching:

```
[Search Options]
FlagBits=111x1111
```

Configuring Character Counting

As supplied, tUME will count the number of 8x8 image characters (TilesetType = 0) loaded, whether or not they appear in an edit room. It counts SNES mode 2 characters, so it only looks at the lowest four bits in determining whether two tiles have the same image. It will check for X-flip, Y-flip, and XY-flip in determing whether two characters are the same. Tiles that also appear in Table Rooms (RoomType = 1) get count separately, without merging duplicate tiles nor checking for flipped tiles.

The relevant lines in the tUME.INI files look like this:

Example tUME.INI sections to count all characters:

```
[Count Characters]
CharacterWidth=8
CharacterHeight=8
CompareMask=xxxx1111
Groups=Character Groups

[Character Groups]
CharGroup1=MergeDuplicates,MergeXFlips,MergeYFlips,MergeXYFlips
CharGroup2=NoMerge

[CharGroup1]
Tileset=0,Room=none
Tileset=0,Room=0..65535

[CharGroup2]
Tileset=0,Room=1
```

The [Count Characters] section describes the basic parameters of the characters to count. CharacterWidth and CharacterHeight says to count 8x8 pixel characters. If both are zero, it counts all tiles, irregardless of size.

CompareMask says to only examine the lowest four bits in each pixel when determining which characters are duplicates. Groups says the section name of the different characters to count is [Character Groups].

The [Character Groups] section says there are two logical groups of characters to count,

CharGroup1 and CharGroup2. Both names ares user-defined, and refer to sections by the same name.

Following each character group name is one of five keywords in a list, separated by commas. The keywords and their meanings are:

NoMerge	treat every character as unique
MergeDuplicates	count characters with same image as one tile
MergeXFlips	count characters that are flipped about the X-axis as one tile
MergeYFlips	count characters that are flipped about the Y-axis as one tile
MergeXYFlips	count characters that are flipped about the X and Y-axis as one tile

The [CharGroup1] section determines how to count characters for the first logical character group. The first line says to count all characters of tileset type 0 that have not been placed in any edit rooms (none). The second line says to count all characters of tileset type 0 that have been placed in any edit rooms (rooms with a room type in the range of 0..65535). Note that when we count the characters in this logical group, we MergeDuplicates, MergeXFlips, MergeYFlips, and MergeXYFlips.

The [CharGroup2] section determines how to count characters for the second logical character group. The line says to count all characters of tileset type 0 that appear in an edit room of room type 1. Note that when we count the chracters in this logical group, we do NoMerge.

Other sample lines and their meanings:

Tileset=0..2,Room=none	count characters of tileset type 0, 1 or 2 that do not appear in an edit room
Tileset=0..2,Room=0	count characters of tileset type 0, 1 or 2 that appear in an edit room of type 0
Tileset=0..255,Rooms=0..65535	count all characters (0..255) that do appear in some edit room (0..65535)

Each character will only be counted once in a given logical character group, irregardless of how many lines it satisfies within that group. Thus, any given line may appear multiple times in a given logical character group, and tUME will not count the characters more than once (it's the union of characters in each line and not the sum of all characters in each line). However, if any given line appears in *n* different logical character groups, each time a tile satisfies the condition on that line, it will be counted *n* times.

Counting Only Characters That Appear in Edit Rooms

Here are the lines you want to change in your tUME.INI file to make tUME count only the characters that are actually used in edit rooms:

Example tUME.INI sections to count only the characters that appear in an edit room:

```
[Count Characters]
CharacterWidth=8
```

```
CharacterHeight=8
CompareMask=xxxx1111
Groups=Character Groups

[Character Groups]
CharGroup1=MergeDuplicates,MergeXFlips,MergeYFlips,MergeXYFlips

[CharGroup1]
Tileset=0,Room=0..65535
```

Notes that these commands count ALL tiles of tileset type 0 in ALL edit rooms. If you only want to count a specific edit room, you could give that room a unique room type, and change the expression Room=0..65535 in the [CharGroup1] section to Room=<*your unique room type*>.

Configuring the File Requester

You may configure the file requester to either display the directories and files alphabetically intermixed or display all the directories first, then all the files.

To configure the file requester to list all the directories first, then all the files, set DirsAtTop= in the [File Requester] section of the tUME.INI file to 1; to configure the file requester to list the directories and files intermixed alphabetically, set DirsAtTop= in the [File Requester] section of the tUME.INI file to 0:

```
[File Requester]
DirsAtTop=0
```

Note that if you set DirsAtTop=0, the [Files] and [Dirs] buttons do not appear in the file requester.

Configuring Mouse Sensitivity

You may set the mouse sensitivity by changing the numbers in the [Mouse] section of the tUME.INI file:

```
[Mouse]
MouseXRes=640
MouseYRes=400
```

The MouseXRes=sets the X-axis sensitivity; the default value is 640. The MouseYRes= sets the Y-axis sensitivity; the default value is 400. In both instances, setting the numbers to a larger number will make the mouse less sensitive, and setting the numbers to a smaller number will make the mouse more sensitive.

Configuring Printing

You may specify which port to print to and page numbering [Print Maps] section of the

tUME.INI file:

```
[Print Maps]
PrintTo=lpt1
NumberPages=1
```

To print to a different port, say LPT2, change the command to PrintTo=lpt2. To print to a file, specify a DOS filename instead of lpt1. To disable page numbering, change the command to read NumberPages=0.

Configuring Tileset Search Path

By default, tUME loads tilesets using the directory and filename specified in the tUME map file. This can be a problem if you move tilesets to another sub-directory (perhaps on a different machine). To adress this problem, tUME will use the following search strategy in loading tilesets:

1. [tUME.INI] Load tileset using the directory saved in the tUME map file, if SearchAsSpecified=1. If not found, then
2. [USER-SPECIFIED] Search sub-directories specified by user, if any. If not found, then
3. [tUME.INI] Search sub-directories specified by SearchDir's, if any. If not found, then
4. [tUME.INI] Search current sub-directory, if SearchCurrentDir=1. If not found, then
5. [ASK USER] **Can't load 'tileset'! Would you like to try a different file?**

Option 2 sub-directories are added by the user when they click **Yes** in response to the question **"Try to load other 'problem' tilesets from here?"**

Option 3 is useful when you copy maps from another machine and you want to place all tilesets in a separate sub-directory from the tUME map. E.g., if you want tUME to search the sub-directory Z:\PROJECT\ART for tileset files, add the following lines to the tUME.INI file:

```
[Load Options]
SearchDir=Z:\PROJECT\ART
```

If you want to search the sub-directory C:\GOOD\ART as well, simply add another line:

```
SearchDir=C:\GOOD\ART
```

Option 4 is useful when you copy maps from another machine and you want to place the tUME map and its tileset files in the same sub-directory. To enable this option, add the following lines to the tUME.INI file:

```
[Load Options]
SearchCurrentDir=1
```

Option 5 is useful if you want to rename a tileset. Change the tileset name before you start tUME. When tUME tries to load the renamed tileset, it won't find it, and then it will give you a chance to specify the new filename.

Thus to make tUME search the current sub-directory of the map file, set SearchAsSpecified=0 and SearchCurrentDir=1. Do not specify any SearchDir's.

Redefining Keys

To define a key, edit the tUME.INI file with your favorite text editor. Move down through the file until you see a lot of lines that start with the word 'KeyEvent.' This will be in the [KEYS] section. Notice the syntax of the lines:

```
KeyEvent    'Key'    Event
```

You must put the word 'KeyEvent' at the beginning of the line. This tells tUME what the line is for. Next you specify the key you are assigning. All keys must by surrounded by single quotes and all key combinations and special keys must be surrounded by angle brackets '<', '>'. Some example keys might be:

'q'
'p'
'2'

Some key combinations are:

'<Alt-p>'
'<Home>'
'<Ctrl-PgUp>'
'<Keypad-6>'
'<Alt-Ctrl-t>'

Then you specify which event to assign to this key. All the events are listed below. For example if you wanted to be able to save your map by pressing Alt and the 's' key you would put

KeyEvent '<Alt-s>' SaveMap

Key List

Most alphabetic, numeric and symbol keys may be specified by putting the symbol, letter or number in single quotes. If the specification requires more than one character you must also use angle brackets '<>' in your key specification. NOTE: Capital letter will require you to press <SHIFT>. In other words there is a difference between assigning something to 'a' and 'A'.

Below is a list of special keys.

'<F1>'	'<Up>'	'<Del>'
'<F2>'	'<Down>'	'<Delete>'
'<F3>'	'<Left>'	'<Ins>'
'<F4>'	'<Right>'	'<Insert>'
'<F5>'	'<BkSp>'	'<Home>'
'<F6>'	'<Esc>'	'<End>'
'<F7>'	'<Tab>'	'<PgUp>'
'<F8>'	'<Space>'	'<PgDn>'
'<F9>'	'<Return>'	'<Grey-+>'
'<F10>'	'<Enter>'	'<Grey-->'

'<Grey-*>' '<Grey-/>'

Below is the list of Qualifiers that may be added to a key specification. More than one may be added separated by a dash '-' for example '<Alt-Shift-p>'.

Alt
Shift
Ctrl
Keypad

Redefining Menus

All menus may be redefined. To do this edit the TUME.INI file with your favorite text editor and look at the [MENU] section. There are four menu keywords:

POPUP
POPDOWN
MENUITEM
STICKYMENUS

µ §
Look at the [MENU] section and compare it against the menus you see in tUME. A **POPUP** line displays a choice that brings up more menu choices. A **POPUP** has one optional parameter which is the text you want to appear on the **POPUP**. You may also use any of the switches (see below). For every **POPUP** statement you MUST have a corresponding **POPDOWN** to mark the end of the current **POPUP**. **POPDOWN** takes no parameters.

The **MENUITEM** keyword specifies a selection or choice on the menus. **MENUITEM** has two optional parameters. The first is the text that appears for this **MENUITEM** and the second is the **event** that happens when this **MENUITEM** is chosen (see **List of "Events"**, below). If you don't specify an event, then this menu item will appear on the menus but it won't have any effect. You may use any of the **switches** with **MENUITEM** (see below).

The keyword **STICKYMENUS** affects the entire menu. **STICKYMENUS** makes the menus work like Microsoft Windows in this sense: if you bring the menus down with the mouse and then let off the mouse the menus will stay active. This means you can click once to bring the menus down and click again to choose the menu item you want. If you don't use the **STICKYMENUS** option then menus always disappear when you let off the mouse, which is more Macintosh / Amiga like.

When you specify text for a **POPUP** or **MENUITEM** you can put an '&' symbol before any letter and then the following letter will be 'underlined' and will also become a key you can use to choose the **POPUP** or **MENUITEM** when using the menus from the keyboard. You may also put the characters '\t' once in the text. All text to the left of the '\t' will appear left justified in the menus and all text to the right of '\t' will appear right justified.

For all **MENUITEM**s with an associated event, if the event has any keys assigned to it (see

Redefining Keys, above) the first assigned key will appear right justified in the menus.

Switches

Finally there are two switches, **SEPARATOR** and **MENUBREAK**. **SEPARATOR** adds a horizontal line just before the current **MENUITEM** or **POPUP**. **MENUBREAK** will start another column of menu items for the current popup.

[COLOR MENU]

In addition to the main [MENU] section, tUME also expects to find a [COLOR MENU] section in tUME.INI. This defines the menu commands available while the palette requester is active. The default [COLOR MENU] is:

```
[COLOR MENU]
POPUP       "&Colors"
        MENUITEM  "&Load..."          LoadPalette
        MENUITEM  "&Save..."          SavePalette    SEPARATOR
        POPDOWN
POPUP       "&Range"
        MENUITEM  "&Load..."          LoadPaletteRange
        MENUITEM  "&Save..."          SavePaletteRange     SEPARATOR
        POPDOWN
```

Redefining Scrolling

Scrolling events and the keys they are attached to are user defined in the tUME.INI file. The two steps involved are first, create a new event that specifies how far and in which direction to scroll the room, and second, attach a key to this new event.

All cursor movement events are creating by listing them in the [Cursor Movement Events] in the tUME.INI file. As supplied, there are eight events in this section:

```
RightScroll=-1,0
RightScrollMultiple=-5,0
LeftScroll=1,0
LeftScrollMultiple=5,0
DownScroll=0,-1
DownScrollMultiple=0,-5
UpScroll=0,1
UpScrollMultiple=0,5
```

The two numbers that follow specify how far <x>,<y> to move the screen. So if we wanted to add a new event that scrolled the screen up and right one tile time, we would add the following line:

```
UpRightScroll=-1,1
```

The name (UpRightScroll) we select is totally arbitrary; you may call the new event Foobar if you wish.

After you define a new cursor scrolling event, you need to attach a key to it, otherwise you won't be able to access it. We skip down to the [KEYS] section in the tUME.INI file, and add this line:

```
KeyEvent        '<Keypad-9>'  UpRightScroll
```

Now whenever we press 9 on the numeric keypad, we will scroll right and up one tile. See the section Redefining Keys above for a more detailed description of attaching keys to events.

List of "Events"

Below is a list of 'Events'. These events can be assigned to almost any key combination and to any menu item.

Some of the events listed below, such as **LeftScroll**, **LeftScrollMultiple**, **RightScroll**, etc. are **user-defined** events rather than built-in events; this means the name of the event has be created by the user and entered in the **tUME.INI** file. In the case of the aforementioned scrolling events, they are defined in the **[Cursor Movement Events]** section in tUME.INI file.

The keys listed following each event is what is currently attached to the event as defined in tUME.INI. These keys are completely user redefinable.

About

Displays the About tUME box.

AppendLayer

Load a layer and append it as the topmost layer of the current room.

AppendMap

Load a map without clearing out the previous map. If the filespecs of the tilesets in the new map match the tilesets filespecs of the currently loaded map then the new map will use the currently loaded tiles. If the filespecs do not match but the file names do then you will be asked if tUME should load a new tileset or use the one it already has loaded. You usually don't want it to load the new tileset.

CenterRoomOnCursor KEY = 'n'

Moves the tile under the cursor to the center of the display or window. Same as DPaint.

ClearMap

Clears all rooms and tilesets from memory.

ClearRoom

Clears all tiles from a room.

ClearRoom3

Clears the room and it's tile size so that a new tile size may be stamped into it.

CopyColors1

Copies the color palette and all color cycles from the room the current tile-brush was grabbed from to the current room.

CopyColors2

Copies just the color palette from the room the current tile-brush was grabbed from to the current room.

CopyColors3

Copies just the color cycles from the room the current tile-brush was grabbed from to the current room.

CopyrightStatus

Displays the copyright message in the status bar. This is what you usually see in the status bar when you first run tUME.

CountChars KEY = '<Alt-C>'

Count the characters used in the map.

CreateRoom

Creates a new room by first asking for the width and height of the new room.

CursorStatus KEY = '<F9>'

Sets the Status bar to Coordinates mode. See 'Coordinates' in the Status Bar section.

DecBackColor KEY = '<Ctrl-[>'

Changes to background color tUME uses to draw areas where there are no tiles.

DeleteLayer

Deletes the current floor layer.

DeleteRoom

Deletes a room from memory.

DeleteTileset

Deletes a source tileset from memory.

Download16 KEY = '<Alt-D>'

Download current layer of current room as 16-color characters to target development system.

Download256 KEY = '<Ctrl-D>'

Download current layer of current room as 256-color characters to target development system.

DownScroll KEY = '<Down>'

User-defined event; see tUME.INI. Scrolls the current room down one tile.

DownScrollMultiple **KEY = '<Ctrl-Down>'**

User-defined event; see tUME.INI. Scrolls down multiple tiles. How many is defined in tUME.INI.

EditOnlyFloor **KEY = '<Alt-F>'**

Makes only the floor layer visible, and make only the floor layer editable.

EditRoom **KEY = '<Keypad-.>'**

Put brush stamping in Edit Room Mode (the default mode). This as opposed to the colorset modes. See **Colorsets** in the *tUME User's Guide* and **Redefining Colorsets** above.

ExportBrush

Writes the current brush as an IFF picture.

ExportRoom

Writes the current room as an IFF picture.

ExportScreen

Writes the current screen as an IFF picture.

FlipPanes **KEY = '<space>', 'j'**

Flips between the two panes of a window.

FloorDown **KEY = '<Alt-Down>', '4'**

Moves the floor down one layer.

FloorUp **KEY = '<Alt-Up>', '3'**

Moves the floor up one layer.

GetGridFromBrush **KEY = '<Alt-G>'**

Set the size of the grid based on the size of the tile-brush.

GetGuideFromBrush **KEY = '<Alt-O>'**

Set the size of the guide based on the size of the tile-brush.

GoEnd **KEY = '<Ctrl-End>'**

Show the lower-right corner of the current room.

GoHome **KEY = '<Ctrl-Home>'**

Show the upper-left corner of the current room.

GroupSaveTiles

Saves the current map just like SaveMap but also saves the tile graphics into the map in the TMGX format.

HideCursor KEY = '<F8>'

Turns the mouse pointer off.

HilightTile KEY = '<Alt-h>'

Show the source room that contains the tile contained in the tile-brush. See **Highlight Tile in Source Room** in the *tUME User's Guide*.

IncBackColor KEY = '<Ctrl-]>'

Changes the background color tUME uses to draw areas where there are no tiles.

InsertLayer

Inserts a new layer to the current room, and pushes the previous floor layer and layers above it up by one.

KeepDownloadPalette

Toggles whether to keep updating the palette on the target development machine or not.

LastMenuEvent KEY = 'a'

Repeats the last menu command. Same as DPaint.

LeftScroll KEY = '<Left>'

User-defined event; see tUME.INI. Scrolls the current room left one tile.

LeftScrollMultiple KEY = '<Ctrl-Left>'

User-defined event; see tUME.INI. Scrolls multiple tiles to the left. How many is defined in tUME.INI.

LoadLayer

Load a layer to the floor of current room, and pushes previous floor layer and layers above it up by one.

LoadMap KEY = '<Alt-L>'

Loads a map after first clearing any old map from memory.

LoadPalette

Loads an IFF palette into the currently selected palette. You may not attach a key to this event.

LoadPaletteRange

Loads an IFF palette range into the selected palette range. You may not attach a key to this event.

LoadRoom

Loads a room, same as AppendMap.

LoadTilesAllTiled KEY = '<Ctrl-L>'

Loads a tileset from a DPaint picture by cutting out every tile on the grid. See **All Tiled** in the **Tilesets** section in the *tUME User's Guide*.

LoadTilesBoxed

Loads a tileset from a DPaint picture using the boxed method. See **Boxed** in the **Tilesets** section in the *tUME User's Guide*.

LoadTilesCookieCutter

Loads a tileset from a DPaint picture using the Cookie Cutter method. See **As Brushes** in the **Tilesets** section in the *tUME User's Guide*.

LoadTilesFullTiled

Loads a tileset from a DPaint picture using the Full Tiled method. See **Full-Tiled** in the **Tilesets** section in the *tUME User's Guide*.

LoadTilesFullTiledNoBlank

Loads a tileset from a DPaint picture using the Tile-Blanks method. See **Tile-Blanks** in the Tilesets section in the *tUME User's Guide*.

MakeCompositeTiles

Create a composite tileset from an edit room. See Composite Tiles section in the *tUME User's Guide*.

NextRoom KEY = '2'

Switches to the next room.

OpenLayer

Adds a new layer to the current room. The new layer becomes the topmost layer.

PreviousRoom KEY = '1'

Switches to the previous room.

PrintMap

Prints current room to HP LaserJet compatible printer.

QuitAndExit KEY = '<Shift-Q>', '<Alt-x>'

Exits tUME.

Replace KEY = 'r'

Find the next occurrence of the Search Buffer, and replace it with the current tile-brush.

RightScroll KEY = '<right>'

User-defined event; see tUME.INI. Scrolls the current room to the right one tile.

RightScrollMultiple KEY = '<Ctrl-right>'

User-defined event; see tUME.INI. Scrolls the current room to the right multiple tiles. How many is defined in tUME.INI.

RoomStatus KEY = '<F6>'

Sets the status bar to Room Info mode. See **Room Info** in the *tUME User's Guide*.

SaveLayer

Saves the floor layer of the current room into a map file and only those tilesets that are used in the layer.

SaveMap KEY = '<Alt-s>'

Saves all the current rooms and tilesets into a map file.

SavePalette

Saves the currently selected palette to an IFF file. You may not attach a key to this event.

SavePaletteRange

Saves the currently selected palette range to an IFF file. You may not attach a key to this event.

SaveRoom

Saves the current room into a map file and only those tilesets that are used in this room.

SaveRoomAll

Saves the current room into a map file and also saves all the tilesets that are currently loaded.

SaveTilesAsBrushes

Saves all the tiles as DPaint brushes in a directory you specify.

ScrollLock

Prevents tUME from scrolling around a room.

ScrollLockKEY

Prevents tUME from scrolling around a room.

SearchNext KEY = 's'

Find the next occurrence of the Search Buffer in the current room.

SelectBlock KEY = 'b'

Removes the current tile-brush from your pointer, and set tUME so next left mouse button press select a new tile-brush.

SelectSquare KEY = 'v'

Removes the current tile-brush from your pointer, and set tUME up so that you may select another. When you select another it will only be one layer deep regardless of how many layers deep the current room is.

SetGridOrigin KEY = '<Shift-G>'

Sets the upper left corner of the grid.

SetGridSize

Brings up dialog box that asks user to set the grid size.

SetGuideOrigin KEY = '<Shift-O>'

Sets the upper left corner of the guide.

SetGuideSize

Brings up dialog box that asks user to set the guide size.

SetMaxTileUsage

Sets the largest tile usage count that will be displayed.

SetRoomInfo

Brings up a dialog box that allows you to edit a room's name, User Type and User Number.

SetSearchBuffer KEY = '<Ctrl-S>'

Copies the current tile-brush to the Search Buffer. Used with search, and with search and replace.

SetStampPaint KEY = '<F1>'

Set the tile brush so it does not stamp NULL tiles.

SetStampReplace KEY = '<F3>'

Set the tile brush so it does stamp NULL tiles.

SetTheColors **KEY = 'p'**

Brings up the Palette Requester allowing you to edit the palette of the current room.

SetTilesetInfo

Brings up a dialog box that allows you to edit a tileset's name, User Type and User Number. The tileset is determined by the tile in the top left corner of your current tile-brush.

ShowBrushCount **KEY = 'h'**

Count and display how many times the current brush occurs in the floor and above layers of the current room.

SpaceToggle **KEY = '\'**

Toggles whether or not a single pixel space is drawn between each tile in a room or not. By default Source Panes display tiles with the space and Edit Panes do not.

StripDownBlockCopy **KEY = '<Shift-X>'**

Strips the current tile brush to just one layer.

TileStatus **KEY = '<F5>'**

Sets the status bar to Tiles mode. See Tiles in the Status Bar section above.

TMGCSaveTiles

Saves the current map just like SaveMap but also saves the tile graphics into the map in the TMGC format.

ToggleCycleColors **KEY = '<Tab>'**

Toggle Cycling Colors on and off.

ToggleDownloadOneScreen

Toggle between downloading one screenful and downloading entire room.

ToggleJamPalette **KEY = '<Alt-p>'**

tUME will force the last 4 colors of the palette to something you can read the status bar with. This event toggles that feature on and off. The default is off.

ToggleLAll **KEY = 'i'**

Toggles the Invisible and Locked flags of the floor layer on or off.

ToggleLInvisi

Toggles the Invisible flag of the floor layer on or off.

ToggleLLock KEY = 'l'

Toggles the Locked flag of the floor layer on or off.

ToggleLockRoom

Locks or Unlocks the current room. A locked room acts like a source room meaning when you click on a locked room you are picking up tiles and not placing them.

ToggleShowBrush

Toggles whether or not you see the tiles you are carrying in your current tile-brush to just a rectangular outline.

ToggleShowGuide KEY = 'o'

Toggles the guide on and off.

ToggleShowTileUsage KEY = '<Alt-u>'

Toggles the display of tile usage numbers on and off.

ToggleSmartFlip

Modify FlipPanes so it flips to the appropriate source room for the current edit room floor layer. See **Smart Flip** in the *tUME User's Guide*, and **Redefining Layer Types**, above.

ToggleStratifyPaste

Enable paste mode where tiles in brush gets pasted into correct layer as defined by the tileset type.

ToggleSwankyMode

Does nothing. Really!

ToggleTitleBar KEY = '<F10>'

Turns the Status Bar on or off.

ToggleUseGrid KEY = 'g'

Toggles the grid on and off.

ToggleZoom KEY = 'm'

Toggles between last zoom setting and zoom off.

Undo KEY = 'u'

Undoes the last edit you made to a room.

UpScroll KEY = '<Up>'

User-defined event; see tUME.INI. Scrolls the current room up one tile.

UpScrollMultiple KEY = '<Ctrl-Up>'

User-defined event; see tUME.INI. Scrolls the current room up multiple tiles. How many is defined in tUME.INI.

UseEditPalette KEY = '<Alt-e>'

This event displays all subsequent source rooms using the last edit room's palette.

UserStatus KEY = '<F7>'

Sets the status bar to User Info mode. See **User Info** in the **tUME User's Guide**.

VersionStatus

Shows the current version of tUME in the Status bar.

WClose

Closes the current Window.

WCreate

Opens a new Window.

WLockClear KEY = '<Alt-a>'

Set the current Window's current pane to show both edit rooms and source rooms.

WLockToEdit

Sets the current Window's current pane to show edit rooms only.

WLockToSame

Sets the current Window's current pane to show rooms that are the same type as the current room.

WLockToSource

Sets the current Window's current pane to show source rooms only.

XFlipBrush KEY = 'x'

Flips the current tile-brush horizontally.

YFlipBrush KEY = 'y'

Flips the current tile-brush vertically.

ZaveMap

Saves a map and saves every tile in every tileset as a separate DPaint brush (**Xave**).

ZeroBackColor KEY = '<Ctrl-'>'

Sets the background color to zero. tUME uses this color to draw all areas where there are no tiles.

ZoomIn KEY = '<>>'

Display the current room using the next available zoom in setting.

ZoomOut KEY = '<<>'

Display the current room using the next available zoom out setting.

tPBin

User's Guide

Gregg A Tavares & Dan Chang

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

tPBin

tPBin is a *tUMEPack* designed for programmers that want some kind of “raw” or “binary” dump of the tUME data so that they may create their own tUMEPACK-like programs that process these "raw" dump files instead of directly reading tUME files. When creating a tUMEPack to generate data for you games, we recommend that you modify the code for DUMPTUME to create your own tools, as it contains routines to read tUME maps directly; however, if you are not comfortable with modifying DUMPTUME or one of the other versions of tUMEPACK, then here is another solution you might be more comfortable with.

tPBin outputs following files:

<*RoomName*>.R*nn*, one for each layer *nn* in <*RoomName*>: two-dimensional array of tile indexes words
<*RoomName*>.F*nn*, one for each layer *nn* in <*RoomName*>: two-dimensional array of tile flag bytes
<*RoomName*>.PAL, one for each room: color information for this room
<*TilesetName*>.T*nn*, one for each tileset type: actual graphic images (one byte per pixel) of each tile

tPBin works as follows:

1. Read in all maps.

2. Find all tilesets with the same UserType, and "merge" them together. E.g., if you have two tilesets, SPLOON.LBM with 25 tiles and SPLAT.LBM with 17 then when they are merged SPLOON.LBM tiles will be numbered from 1 to 25 and SPLAT.LBM tile will be numbered 26 to 42.

3. Write out the tile graphics for each merged tileset to a binary file "*.T??".

4. For each room, write a file "*.PAL" with the palette of 256 colors.

4. For each room, sort tiles to make all tile of user type *n* appear on layer *x*. *n* and *x* are specified using the -m switch.

5. For each layer in each room, write a file "*.R??", which is a two-dimensional array of words, where each word is an index into the merged tileset.

6. For each layer in each room, write a file "*.F??", which is a two-dimensional array of bytes, where each byte is the tile flag byte at that location. Tile flags bits are defined in the tUME.INI file.

tPBin Command-Line Switches

Single letter switches with no arguments can be turned on with a '+', and turned off with a '-'. In

the following list, the '-' or '+' in front of the switch specifies the default setting of that switch, thus debugging (-d) is off by default, while (+d) would turn it on.

-d Disable debugging messages. +d to enable debugging messages.

-f<n> First Tile. When tiles are renumbered after like tilesets have been merged the first tile is usually numbered 0. NULL tiles show up in the .R?? files as zeros (0). This option lets your specify the number of the first tile in the merged tilesets. If you use '-f0' then you won't be able to tell the difference between the first tile and NULL tiles; however, this may be what you want.

-g Write Room Width/Height .R?? files. +g to write width and height information into the files.

-k Generate only tiles used in rooms. Normally all tiles are generated from the tilesets whether or not they are actually used in any room. +k will make *tPBin* generate only those tiles actually used and will renumber the tiles in the .R?? files so they point to the correct image.

-m<tileMap> **This option MUST BE SPECIFIED!** This option specifies which tiles should be put in which layers (or which .R?? files) You should account for all types of tiles that will appear in your map otherwise you will get a warning that a tile is being discarded. The mapping is specified by entering a tileset UserType followed by an '=' followed by a layer number. Layers start at layer 0 (zero). For example: -m0=0,1=1,3=2,2=3. This would mean tiles of UserType 0 are written to layer 0, tiles of usertype 1 to layer 1, tile of type 3 to layer 2 and tiles of type 2 to layer 3

-p Not implemented yet.

tPBin File Formats

NOTE: All words are in Little Endian format (low byte first)

.T?? files (Tileset Graphics)

A binary .T?? file, where ?? is the tileset UserType in hex, is generated for each 'merged' tileset (see above). The filename will be the name of the first tileset of this specific UserType tUME finds in the map. The data is stored 1 byte per pixel. If the tileset used 8x8 pixel tileset then the first tile will be the first 64 bytes of the file and the second tile will be the second 64 bytes. The size of the tiles is not encoded in this file.

.R??

A binary .R?? file, where ?? is the number of the layer in hex, is generated for each layer in a room. The data is stored as an array of words. The array is <room width> by <room height> of words. Each word indexes the corresponding merged tileset.

The file is a two-dimensional array of words:

WORD map_index[{room height}][{room width}]

A word value of zero indicates the absence of any tiles at that location. By default, tiles are numbered starting from 0, but you may use the -f switch to specify a different starting number.

If the +g option is specified then the file is preceded by the size of the room in blocks.

WORD room_width
WORD room_height

.F??

A BINARY .F?? file, where ?? is the number of the layer in hex, is generated for each layer in a room. The data is stored as an array of bytes. The array is <room width> by <room height> of words. Each bytes is the tileflags for the corresponding tile in the .R?? file. Tileflags are interpreted as specified in the tUME.INI file; the defaults are:

BITS	0-2	Colorset of tile
BIT	3	Unused
BIT	4	Tile has priority set.
BIT	5	Tile is flipped vertically
BIT	6	Tile is flipped horizontally
BIT	7	Colorset information (bits 0-2) are used

The file is a two-dimensional array of bytes:

UBYTE tileflags[{room height}][{room width}]

.PAL files (Palettes)

A Binary .PAL file is generated for each room. There are 256 entries each consisting on three bytes (red, green and blue). The red, green and blue values range from 0 to 255 with 255,255,255 being white.

PALETTE[256] (formatted as follows):

PALETTE consists of:	
UBYTE	Red color information
UBYTE	Green color information
UBYTE	Blue color information

tP16

User's Guide

Gregg A Tavares & Dan Chang

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

tP16

tP16 is a *tUMEPack* designed for 16-bit console products. The authors of *tUME* do not endorse the methods described in *tP16* as the "one true religion"; in fact, we disagree with many of the design decisions implicit in this *tUMEPack*. However, the product exists, and it works; so study it, and learn from it. Use it if you must, but we recommend that you adapt it to your needs.

Here is a description of the rooms and tilesets *tP16* expects, and what files are generated. *tP16* is designed to work with 8x8 tiles or 16x16 tiles. Here are the tileset types that *tP16* recognizes:

Tileset Type	*User Type*	*User Number*
Image Tiles	0	NA
Contour Tiles	1	NA
Special Tiles	2	NA
Object Tiles	3	NA
4 Color Tiles	4	NA
256 Color Tiles	5	NA
Mode7 Tiles	6	NA

The second column is the tileset number that *tP16* processes, and the first column describes what *tP16* considers that tileset user type to mean. *tP16* ignores the tileset user number.

Image tiles will be converted into 16 color SNES or Genesis characters, 4 color tiles will be converted into 4 color SNES characters, 256 color tiles will be converted into 256 color SNES characters. When *tP16* creates the .CHR font file it puts 4 color characters first, then 16 color characters and finally 256 color characeters.

It is best if all tilesets start with a blank tile in the top left corner in DPaint. This is because any NULL TILEs in your maps are converted to tile 0, colorset 0.

tP16 writes out only one .CHR file; if you create rooms using several different character types, note that all characters used in the tiles in a table room are added to the end of the character font consecutively such that you can use DMA to transfer all of them at once.

Contour tiles may be used in level rooms in the second layer. They generate .CON files. Special tiles may be used in level rooms in the third layer; they generate entries in the .FLR file. Object tiles may be used in level rooms in the fourth layer; they also generate entries in the .FLR file.

Mode 7 tiles are used in Mode 7 rooms.

Here are the room types that *tP16* recognizes:

Room Type	User Type	User Number
Level Room	0	NA
Table Room	1	NA
Picture Room	2	NA
Parallax Room	3	NA
Mode7 Room	4	NA
Flat Conversion Room	5	NA
Layered Conversion Room	6	NA
Animation Room	8	NA

The second column is the room user type that *tP16* processes, and the first column describes what *tP16* considers that room user type to mean. *tP16* ignores the room user number.

Level Rooms

Level rooms generate a .MAP and a .PAL file. Level rooms are treated as 16x16 tiles. The first layer is treated as follows. The program builds a tileset table (a .BLK file) which contains the index numbers of the four 8x8 pixel characters that make up the 16x16 pixel tile along with the flip, colorset and priority information for each character. The program also builds a map (.MAP) file. Each tile on the map is an index into the .BLK file.

The second layer of a level room contains the contour information. Each entry in the .MAP file also indexes a table (a .FLR file) that indexes the .CON file to show which contour tile is used at this map location. The third layer contains special tiles, which is stored in the .FLR file as well. The fourth layer contains object tiles, which is also stored in the .FLR file.

Second Playfields:

A second set of four layers may be added to a level room in which case *tP16* will generate a .MAP file as above for the first 4 layers and an .MP2 file which is in the same format as the .MAP file except for the second set of 4 layers. The layers would be as follows.

layer 1 Image Layer
layer 2 Contour Layer
layer 3 Special Layer
layer 4 Object Layer
layer 5 2nd Image Layer
layer 6 2nd Contour Layer
layer 7 2nd Special Layer
layer 8 2nd Object Layer.

Table Rooms

Table rooms generate a .TBL file. Any characters used by any tiles in the table rooms will have their own entry in the character set, and they won't be checked for duplication when they are added to the character set. The font position of each character is then written out to a .TBL file. The purpose of table rooms is to allow you can find where certain characters are in the font so you may re-define them on the fly to do character set animations.

Another use for table rooms is to force inclusion of contour tiles. This is of interest only if you have flipped contours (-e1) on, as with this option only the contour tiles that are "used" are included in the .CON file. (If flipped contours is not on, all contours get included always.) Including a contour tile in a table room meets the definition of "use". This is useful if you want to have one set contour tiles used for all levels in your game.

Picture Rooms

Picture rooms generate a .MAP file and a .PAL file. Picture rooms use 8x8 characters (the .CHR file), and the map (.MAP) file contains words that can be stored in screen memory directly (it contains the flip, colorset and priority information). One .MAP file is generated for each layer in the room.

Parallax Rooms

Parallax generate a .PAR file and a .PAL file. Parallax rooms use 8x8 characters (the .CHR file), and the map (.PAR) file contains words that can be stored in screen memory directly (it contains the flip, colorset and priority information). Parallax rooms are affected by the command-line -j<WxH> switch. If the (-j) option is specifed than each room is broken up into several screen dumps, each WxH characters in size. One .PAR file is generated for each layer in the room.

Example:

If the original Parallax room was 32x16 16x16 pixel tiles then that would equal 64x32 8x8 characters. Without the (-j) option the .PAR file would be arranged like this:

0	1	...	62	63
64	65	...	126	127
...	...	...	...	...
1920	1921	....	1982	1981
1984	1985	....	2046	2047

With the option (-j32x16) the .PAR file would be arranged as follows:

0	1	...	30	31	512	513	...	542	543
32	33	...	62	63	544	545	...	574	575
...	...	...	...	...	...	...	...	...	...
448	449	...	478	479	960	961	...	990	991
480	481	...	510	511	992	993	...	1022	1023
1024	1025	...	1054	1055	1536	1537	...	1566	1567
1056	1057	...	1086	1087	1568	1569	...	1598	1599
...	...	...	...	...	...	...	...	...	...
1472	1473	...	1502	1503	1984	1985	...	2014	2015
1504	1505	...	1534	1535	2016	2017	...	2046	2047

Mode 7 Rooms (SNES)

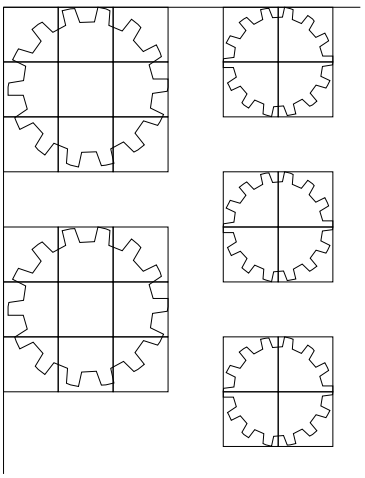
Mode 7 rooms generates a .M7R room file and a .PAL palette file. You should use tiles of type Mode 7 in this room. Since you can not mix Mode 7 with other modes it is best if you do not mix mode 7 tiles with other tiles. The tiles may be 8x8 or 16x16 or 8x8 2x2 composite. Note: All mode 7 limits are enforced. You will be warned if tiles have colorsets or are flipped or if they have priority because all of this is not supported in Mode 7 on the SNES. Mode 7 will only allow 256 characters!

Mode 7 Rooms use 1 byte per tile and are basically retangular screen dump. This means that to use a Mode 7 map you would need to know the width and height of your map. Example: your map is 50 characters wide and 65 characters tall. You would need to copy the first 50 bytes in the mode 7 map file into Video RAM skipping every other byte in Video RAM. Then you would skip 78 WORDS to get to the next mode 7 line in Video RAM and then copy the next 50 bytes and on and on until you have copied all 65 lines.

If you just want a 32K Mode 7 screen dump to can stuff into Video memory and display use the (+7) option below.

Animation Rooms

Animation rooms are used to generate background character animation data. You may place tiles in columns separated by NULL tiles in an animation room. Each NULL tile separates a 'frame' of animation in the column. For example say you had a two spinning gears that you wanted to animate. One gear is 2x2 tiles large and 3 frames and the other is 3x3 and 2 frames. You would place them in an animation room like this:



tP16 will then scan the room and generate animation data for the gears. *tP16* will write out a binary frame data file for the frames and an animation macro file describing how do download the frame data to the character set for animation. Animations that are the same number of frames are merged into one animation. One list of macros is written for each animation unless the (+i) option is given in which case the various animations are interleaved into one macro table. Animations may contain any type of tile (ie. 4 color, 16 color, 256 color, Mode 7) and *tP16* will deal will it. Note: Do not mix Mode 7 tile with other types.

Tiny Mode vs Non-tiny Mode

There are two major modes of operation in tP16, tiny mode and non-tiny mode. The mode affects the output of the .MAP, and the .BLK files. All other files remain the same.

In non-tiny mode, the word in the .MAP table directly indexes the .BLK table. Divide the word in the map by two to index the .FLR table. The .BLK file contains the four characters that should be displayed. All the colorset, flip, and priority information is already encoded into each word in the .BLK file, you can stuff them into screen memory directly.

In tiny mode, the lowest eleven bits of each word (ten bits for SNES) in the .MAP table is multiplied by eight to index the .BLK table, and multiplied by four to index the .FLR table. The upper five bits of each word contains the colorset, flip, and priority information. These need to be merged with the values in the .BLK file before they can be stored in screen memory.

The advantage of non-tiny mode are simplicity (the display code is simpler), and each tile can be made of four tiles that each have a different colorset. The disadvantage of non-tiny mode is a larger .BLK table size; if two locations in the map have the same image, contour, special, and object layer, but differ only in that one instance of the image tile is flipped, two entries will be generated in the .BLK file. Each entry takes 12 bytes. On one particular project, on just one level there were over 3100 entries, which multiplied by 12 bytes is 37200 bytes, just for the .BLK and .FLR files of one section of the game.

The advantages of tiny mode is smaller .BLK table size; if two locations in the map have the same image, contour, special, and object layer, but differ only in that one instance of the image tile is flipped, only one entry will be generated in the .BLK file, and the flipping is encoded in the .MAP file. The disadvantange is that each tile can use only one colorset four all four characters.

Currently, it takes 12 bytes to represent a tile (four words in the .BLK file, and four bytes in the .FLR file). In non-tiny mode, there is a limit of 8192 tiles. In tiny mode, there is a limit of 2048 tiles in the Genesis, and 1024 tiles in the SNES. However, recall that the definition of a 'tile' differs between tiny and non-tiny mode. If a tile is the same as another tile, except that the image is flipped or has a different colorset or priority in one of them, it is counted as two tiles in non-tiny mode, but as only one tile in tiny mode.

Note that *tP16* does not warn you if you generate too many tiles, you must watch the number yourself.

Decoding a Tiny Mode map

To decode a word in the .MAP file in the tiny mode, you must use the [low 11 bits] * 8 to index the .BLK table. (10 bits on SNES) With the four words you get from the .BLK table, you must OR in the colorset and priority. If the flip bits are set in the .MAP word, you must re-order the four words as appropriate. Finally, you must exclusive-or in the flip bits (as the .BLK words may have flip information in it as well), and store them on screen. In C, it looks something like this:

```
#if SEGA
      #define NDX_BITS        0x07FF
      #define FLIP_BITS       0x1800
      #define PRI_COLOR_BITS  0xE000
      #define XFLIP_BIT       0x0800
      #define YFLIP_BIT       0x1000
#elif SNES
      #define NDX_BITS        0x03FF
      #define FLIP_BITS       0xC000
      #define PRI_COLOR_BITS  0x3C00
      #define XFLIP_BIT       0x4000
      #define YFLIP_BIT       0x8000
#endif

UWORD tile;
UWORD ndx;
UWORD tlchar;
UWORD trchar;
UWORD blchar;
UWORD brchar;
UWORD temp;
UWORD flipbits;
UWORD pcbits;

tile = map[y * mapwidth + x];       // Get tile from map
ndx  = tile & NDX_BITS;             // get tile ndx (low 11 bits)
tlchar = blocktbl[ndx].tlchar;      // get top left char
trchar = blocktbl[ndx].trchar;      // get top right char
blchar = blocktbl[ndx].blchar;      // get bottom left char
brchar = blocktbl[ndx].brchar;      // get bottom right char

flipbits = tile & FLIP_BITS;        // extract flip bits.
pcbits = tile & PRI_COLOR_BITS;     // keep only color & pri

if (flipbits & XFLIP_BIT) {
```

```
      // if tile is xflipped then swap
      // left and right characters.
      temp = tlchar;    tlchar = trchar;  trchar = tlchar;
      temp = blchar;    blchar = brchar;  brchar = blchar;
}
if (flipbits & YFLIP_BIT) {

      // if tile is yflipped then swap
      // top and bottom characters.
      temp = tlchar;    tlchar = blchar;  blchar = temp;
      temp = trchar;    trchar = brchar;  brchar = temp;
}

tlchar = (tlchar ^ flipbits) | pcbits; // put
trchar = (trchar ^ flipbits) | pcbits; // it
blchar = (blchar ^ flipbits) | pcbits; // all
brchar = (brchar ^ flipbits) | pcbits; // together

// now the characters are ready to store
// into screen memory.
```

Object Tiles and Special Tiles

Currently, object tiles and special tiles are encoded into the .FLR file. An alternative approach is to generate a list of objects for each room. Typically, this can be encoded in three to five bytes, the x-coordinate, the y-coordinate, and the object number. *tP16* does not currently support generating a list of objects, however, it is an option we can easily add (call us!).

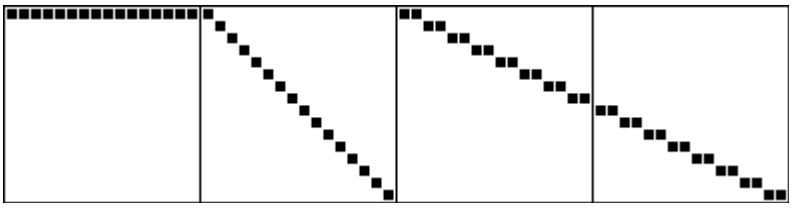
Contour Tiles

Contour tiles are 16x16 pixel tiles. Every tile in the contour tileset is always written to the output .CON file (unless you use the +e switch, in which case it only writes the contour tiles that are actually used). By default, tP16 does NOT process flipped contour tiles; you can configure *tUME* so it does not generated flipped contour tiles. Alternatively, if you decide to use flipped contour tiles, use the +e switch with *tP16*, and it will process flipped contours. It does not check to see if a flipped contour is the same as another non-flipped contour, however (we can easily add this, call us!).

For each pixel column in a tile, scan down the column. If a pixel is found then add the position (+1) of that pixel to the table and continue to the next column. If no pixel is found in the column then add a 0 (zero) to the table. Write out the table to file <*tileset name*>.CON.

Example:

Let say you have CONTOUR.LBM that looks like this:



tP16 will create CONTOUR.CON that looks like this:

```
dc.b  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1                ; 0
```

```
dc.b  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16          ; 1
dc.b  1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8                 ; 2
dc.b  9,9,10,10,11,11,12,12,13,13,14,14,15,15,16,16   ; 3
```

There is another way to deal with contours. Instead of having a separate contour layer, you can create a **conversion room** that associates a contour tile with every image tile. This advantage of doing contours this way is you save memory required to represent your contour. The disadvantage (a minor one) is that any given image will always use the same contour. *tP16* does not currently support contour conversion rooms; however, it is something that can be easily added if you would prefer (call us!).

tP16 Command-Line Switches

Single letter switches with no arguments can be turned on with a '+', and turned off with a '-'. In the following list, the '-' or '+' in front of the switch specifies the default setting of that switch, thus debugging (-d) is off by default, while contour (+c) is on by default.

-a FirstColor. All colorsets in the map (not the palette) are moved up by the amount specified here such that if a tile would normally be displayed in colorset 3, and you use the option -a2, then that tile will be displayed in colorset 5 (3+2). This option is global to all tiles and no error checking is done so that if you do something like have a tile using colorset 5, and you do a -a6, which would make that tile become colorset 11 (5+6), you lose.

-b Binary. Using +b this option will cause ALL files (except table .TBL files) to be written in binary so that they may be easily compressed.

+c Output contour information.

-d Disable debugging messages. +d to enable debugging messages.

-e<ct> Set the flipped contour mode type.

0 = No Flipped contours allowed.

1 = Generate Flipped Contours for Contours that are flipped Instead of converting each contour tile from your DPaint picture directly into a contour table, only those contours that are used are added to the contour table and if a contour is flipped it will be added 'flipped'. Contour #0 is assumed to be blank so make sure it is in your DPaint picture. If you need to know the position of other contours, add them to your TABLE room. Contour tiles in a TABLE room are added to the beginning of the contour table in the order found in your TABLE room, (left to right, top to bottom).

2 = Store flip bit flags in the contour field of the .FLR file to indicate a flipped tile.

bit 0 = X flip (ie. $0001)
bit 1 = Y flip (ie. $0002)

-f<n> Skip the first <n> characters when numbering characters. This option adds a constant to all characters in the block table (.BLK). This means that if a particular tile would normally use characters 10,11,25,27 for its top-left, top-right, bottom-left, bottom-right characters, with -f20 they would become 30,31,45,47. This is all find and dandy; however, if you have mixed tilesets in your map (i.e. 4 color tiles and 16 color tiles) then the -f option makes no sense because 4 color tiles are numbered differently than 16 color tiles and so adding the same constant to both would not work. There are two solutions:

1. Never use mixed tilesets. With this option if you needed to create a 16 color tile level and a 4 color parallax you would create them in separate maps and *tP16* them separately.

2. Use the -r option (see below).

-g Write Room Width/Height and table sizes into files. +g to write size information into the files.

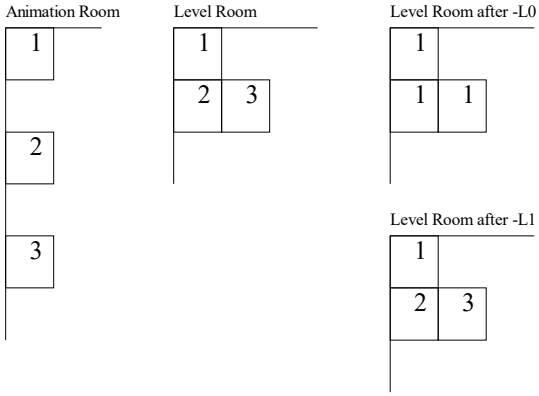
-h SNES 16x16 characters. +h to write SNES 16x16 characters (this is a total hack for a particular project and will not general work as expected)

-j<WxH> Write parallax rooms as several screens, each WxH characters in size.

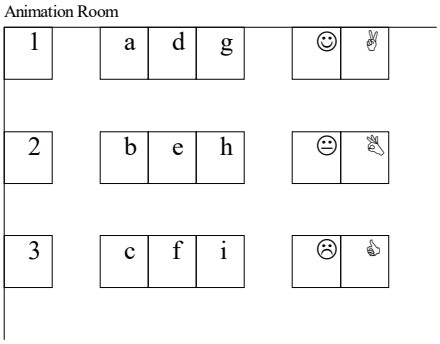
-k Generate only characters used in rooms. Normally all possible characters are generated from the image tilesets whether or not they are actually used in any room. +k will make *tP16* generate only those characters actually used.

-L<animmodes> This arguement takes several sub-parameters.

-TOP : Normally each column of animation converts to the same entry in the block table. In other words if you have a 1 column 4 frame animation in an animation room you may place any of the 4 tiles that make the frames in your LEVEL rooms and they will all be converted to the same animating block. If you specify -L+TOP then only the top tile will be animated. Placing the lower 3 tiles in your LEVEL rooms will generate a standard non-animated block.



+MRG : Normally all animations in one animation room that are the same number of frames in length are merged into one animation. If you specify -L-MRG then, if you separate animations by at least one column of NULL tiles they will generate a separate animation in the .ANI file



Example: Given the animation room above, with -L+MRG there would be one animation in the .ANI file. With -L-MRG there would be 3 animations.

-ZRO : Zero out animation area in character font. When an animation room is found and animation data is created the first frame of animation is also written to the character font since it must be available to initially display the level. With this option specified, the area in the character font where the animation is happening will be zeroed out so that it will compress better and an initialization table is added to the animation macro file specifying what animation data to copy into the character font at runtime to restore the missing characters.

-INT : Interleave animations of different frames. Normally animations of different lengths are written into the macro file as separate lists. -L+INT will write the different animations into one table but it will interleave them. For example if you had a two frame animation of frames ①and and a three frame animation of frames ❶,❷ and ❸ then with -L+INT the frames would be interleaved like this: ①, ❶, , ❷, ①, ❸, , ❶, ①, ❷, , ❸.

+CMP: Compress animations. Animations are compressed for example if two columns of characters look and animate exactly the same as another column of characters. -L-CMP turns this compression off.

In tUME, each room may have a comment. Animation use these comments to pass the above parameters so that you may change them from room to room. In otherwords, each room can be processed with a different set of arguments. You can put any of the -L option in the comment for the room. Do not put the '-L'. Just put the actual options (e.g. '+TOP-MRG+ZRO-INT+CMP') Note: if you use room comments to pass animation arguements it is best to specify all five flags for each room

+m MergeLayers. Using this option allows you to put contour, special and object tiles on ANY of the layers above layer 1 (ie. layer 2, 3, 4 ....) and *tP16* will still figure everything out.

-o<binlist> 0 = Palette File, 1 = Contour File, 2 = Font File, 3 = Mode7 Font File, 4 = Table Room File, 5 = Block Table File, 6 = Mode7 Room File, 7 = Level Room File, 8 = Parallax Room File, 9 = Picture Room File, 10 = Floor File, 11 = Animation Macro File. Default = -o0,1,2b,3b,4,5,6b,7b,8b,9b,10, 11. Character files (.CHR) and maps (.MAP & .PAR) are always written out in binary.

+p Pack Font; find duplicate characters including flipped characters and eliminate from character set.

-r<mem> Skip first <mem> bytes in font, rounded up to 16 byte boundary. *tP16* figures out first tile index based on <mem> bytes skipped. Thus, if you specify -r2048 then *tP16* would start numbering your characters appropriately, so that if you are using 16 color tiles, the first character would be numbered 64, or if you are using 4 color characters, the first character would be numbered 128. Using this option *tP16* can correctly number all of your characters even if you are using mixed tilesets.

-s SNES mode. +s to generate SNES instead of Genesis characters and also to make all WORD values be ordered for the 65816 instead of the 68000.

-t Tiny mode. Using this option, the XY Flip bits, the Priority Bit and the Colorset Bits are stored in the map data instead of the Block table.

+w Write files. Use (-w) if you just want *tP16* to report errors but not create any files.

+x Write tiles

+z	Write rooms
-2	2Bit colors. +2 causes tP16 to look at colors 0..3, 16..19, 32..35, etc., instead of colors 0..3, 4..7, 8..11, etc. when generating 4 Color Tiles.
-7	Imbedded font. Using +7 will create a 32K mode7 screen dump that can be copied directly into VRAM for displaying. The .MD7 (font) file will not be generated with this option.
IGNORE	This option allows you to specify which colors are to be ignored in determining the colorset of a particular tile/character. *tP16* used to assume colors 0 and 1 were to be ignored, but now you may specifiy which colors. Example: `TP16 MYLEVEL.MAP IGNORE 0 2 5 7` This will ignore colors 0, 2, 5 and 7

Making *tP16* Run Faster

tP16 will run 10 times faster if you use EMS memory instead of XMS memory. You can get EMS memory by running DOS 5 EMM386 or QEMM. The reason is EMS is up to 32000 times faster than XMS.

tP16 File Formats

.BLK files (Which Four Characters to Use, Non-tiny Mode)

tP16 will write out a ASCII file that describes which four SNES or Genesis characters make up each tile. Each tile generates a four word entry, which specifies the upper-left, upper-right, lower-left, and lower-right character to use. The flip bits, colorset bits and priority bits are already set in each word. You may stuff these into screen memory directly. Note that the number of tiles is not encoded anywhere in this file.

CHARINDEX4[{number of tiles}] (formatted as follows):

CHARINDEX4 consists of:	
WORD	upper-left_character_index
WORD	upper-right_character_index
WORD	lower_left_character_index
WORD	lower_right_character_index

In Genesis mode, each *x-y*_character_index is formatted as follows:

BITS 00..10	index character table
BIT 11	X flip information
BIT 12	Y flip information
BITS 13..14	colorset information
BIT 15	priority information

In SNES mode, each *x-y*_character_index is formatted as follows:

BITS 00..09	index character table
BITS 10..12	colorset information
BIT 13	priority information
BIT 14	X flip information
BIT 15	Y flip information

.BLK files (Which Four Characters to Use, Tiny Mode)

tP16 will write out a ASCII file that describes which four SNES or Genesis characters make up each tile. Each tile generates a four word entry, which specifies the upper-left, upper-right, lower-left, and lower-right character to use. In each word, only the flip bits are set. You must include the flip, colorset, and priority information from the .MAP before setting screen memory.

Note that each entry in the .BLK file corresponds directly to the same index entry in the .FLR file. Note that the number of tiles is not encoded anywhere in this file.

CHARINDEX4[{number of tiles}] (formatted as follows):

CHARINDEX4 consists of:	
WORD	upper-left_character_index
WORD	upper-right_character_index
WORD	lower_left_character_index
WORD	lower_right_character_index

In Genesis mode, each *x-y*_character_index is formatted as follows:

*x-y*_character_index consists of:	
BITS 00..10	index character table
BIT 11	X flip information
BIT 12	Y flip information
BITS 13..15	set to 0

In SNES mode, each *x-y*_character_index is formatted as follows:

*x-y*_character_index consists of:	
BITS 00..09	index character table
BITS 10..13	set to 0
BIT 14	X flip information
BIT 15	Y flip information

.CHR files (Character data)

tP16 will write out one large BINARY character file for all characters collected from all tilesets of UserTypes 0 (zero), 4 and 5. The name of the file is the same as the name of the map being *tP16*'ed with the extension '.CHR' appended. The format of each character is the native format of the SNES or Genesis; see appropriate documentation about that machine's character format for more details. Note that the number of characters is not encoded anywhere in this file; the first byte of the .CHR file is the first byte of the first character.

When *tP16* creates the .CHR font file it puts 4 color tiles first, then 16 color tiles and finally 256 color tiles. The file would look like this if it contained 4-colors, 16-color, and 256-color tiles:

BYTE[16]	4_color_characters[{all 4 color characters used}]
BYTE[32]	16_color_characters[{all 16 color characters used}]
BYTE[64]	256_color_characters[{all 256 color characters used}]

If the file contains mode 7 characters they need to be copied to every other byte of video memory because that's the way mode 7 works.

.CON files (Contour Tiles)

Please see the description of contour tiles, above. These are usually ASCII files that need to be INCLUDEd by an assembler. Each contour tile generates 16 bytes; each byte is the height of the contour at each of the 16 Y positions. Note that the number of contour tiles is not encoded anywhere in this file.

BYTE[16]	contour_information[{number of contours}]

.FLR files (Contour, Special, and Object Information)

An ASCII file that includes contour, special, and object definitions. For every tile, there is a word and two bytes of information. The word is the index into the contour (.CON) table for this tile. It is already multiplied by 16, so it indexes the .CON table directly. The next byte is set if there is a special tile in the special (third) layer, it will be set to the tile number less one. Similarly, the final byte is set if there is an object tile in the object (fourth) layer, it will be set to the tile number less one.

Note that any time an object tile appears, it will generate a unique entry in the .FLR (and corresponding .BLK) table, even if everything else (image, flip, special, contour, colorset, priority) is exactly the same.

Note that each entry in the .FLR file corresponds directly to the same index entry in the .BLK file. Note that the number of tiles is not encoded anywhere in this file.

CONINDEX_S_O[{number of tiles}] (formatted as follows):

CONINDEX_S_O consists of:	
WORD	index into contour table (.CON)
BYTE	{special tile number}-1
BYTE	{object tile number}-1

.MAP files (Level Room Maps, Non-tiny Mode)

A BINARY .MAP file is generated for each level room. Each is an array <room width> by <room height> of words. Each word indexes the .BLK table directly. Divide each word by two to index the .FLR table.

The file is a two-dimensional array of words:

WORD map_index[{room height}][{room width}]

If the +g option is specified then the file is preceded by the size of the room in blocks.

WORD room_width
WORD room_height

.MAP files (Level Room Maps, Tiny Mode)

A BINARY .MAP file is generated for each level room. Each is an array <room width> by <room height> of words. Each word is an encoded field. The <lowest 11 bits> * 8 indexes the .BLK table. The <lowest 11 bits> * 4 indexes the .FLR table. The upper five bits contain x-flip, y-flip, colorset, and priority information.

Note: on the SNES the <lowest 10 bits> * 8 indexes the .BLK table and the upper 6 bits contain x-flip, y-flip colorset and priority information.

WORD encoded_map_index[{room height}][{room width}]

If the +g option is specified then the file is preceded by the size of the room in blocks.

WORD room_width
WORD room_height

In Genesis mode, each encode_map_index word is formatted as follows:

encoded_map_index consists of:	
BITS 00..10	index into .BLK and .FLR tables
BIT 11	X flip information
BIT 12	Y flip information
BITS 13..15	priority and colorset information

In SNES mode, each encode_map_index word is formatted as follows:

encoded_map_index consists of:	
BITS 00..09	index into .BLK and .FLR tables
BITS 10..13	not defined
BIT 14	X flip information
BIT 15	Y flip information

.MAP files (Picture Room Maps)

A BINARY .MAP file is generated for each picture room. Each is an array <room width> by <room height> of words. Each word indexes the .CHR table directly, and includes encoded flip, colorset, and priority information. This room is basically a **screen dump**.

The file is a two-dimensional array of words:

WORD character_index[{room height}][{room width}]

If the +g option is specified then the file is preceded by the size of the room in characters.

WORD width_in_chars
WORD height_in_chars

In Genesis mode, each character_index is formatted as follows:

BITS 00..10	index character table
BIT 11	X flip information
BIT 12	Y flip information
BITS 13..15	priority and colorset information

In SNES mode, each character_index is formatted as follows:

BITS 00..09	index character table
BITS 10..12	colorset information
BIT 13	priority information
BIT 14	X flip information
BIT 15	Y flip information

.MD7 files (Mode 7 Character data)

Note: This file is no longer written. All mode 7 character data is written to the .CHR file unless the (+7) option is specifed. Do NOT mix mode 7 tiles with other types in the same map.

.M7R files (Mode 7 Room Maps)

A BINARY .MAP file is generated for each picture room. Each is an array <room width> by <room height> of bytes. Each byte indexes the .MD7 table directly. This room is basically a **screen dump**.

The file is a two-dimensional array of bytes:

BYTE character_index[{room height}][{room width}]

If the +g option is specified then the file is preceded by the size of the room in characters (unless the +7 option is specifed).

WORD width_in_chars
WORD height_in_chars

.PAL files (Palettes)

An ASCII .PAL file is generated for each room. In Sega Genesis mode, each entry is a PALETTE value that contains three nibbles, R, G, and B color information. You should define a macro which converts PALETTE into something that works on the Genesis:

PALETTE[{number of palette entries}] (formatted as follows):

PALETTE consists of:	
BITS 00.03	B color information
BITS 04..07	G color information
BITS 08..11	R color information
BITS 12..15	always 0

In SNES mode, each entry is a word value that may be directly loaded into the SNES palette registers:

WORD[{number of palette entries}] (formatted as follows):

WORD consists of:	
BITS 00.04	R color information
BITS 05..09	G color information
BITS 10..14	B color information
BIT 15	always 0

.PAR files (Parallax Room Maps without -j)

.PAR files without using the -j option are exactly the same as .MAP files for Picture Rooms as specified above (except with the +g option, see below).

.PAR files (Parallax Room Maps with -j)

A BINARY .PAR file with several arrays each WxH in size (see -j<WxH> command-line switch), is generated for each parallax room. Each is an array WxH of words. Each word indexes the .CHR table directly, and includes encoded flip, colorset, and priority information. This room is basically a bunch of small **screen dumps**.

The file is a several two-dimensional arrays of words:

WORD character_index[{-jW height}][{-jH width}]

How ever many arrays are required to cover the entire room are written to the file.

If the +g option is specified then the file is preceded by the size of the room in arrays and by the size of one array in characters.

WORD arrays_across_room
WORD arrays_down_room
WORD width_of_one_array

WORD	height_of_one_array

In Genesis mode, each character_index is formatted as follows:

BITS 00..10	index character table
BIT 11	X flip information
BIT 12	Y flip information
BITS 13..15	priority and colorset information

In SNES mode, each character_index is formatted as follows:

BITS 00..09	index character table
BITS 10..12	colorset information
BIT 13	priority information
BIT 14	X flip information
BIT 15	Y flip information

.TBL files (Table Room Tagged Characters)

tP16 will write out a ASCII file which contains equates that describe where the four SNES or Genesis characters that make up each tile appear in the character set. Each tile generates a four word entry, which specifies the upper-left, upper-right, lower-left, and lower-right character to use. Instead of actually generating data, currently the program generates an EQUATES file.

CHARINDEX4[{number of tiles}] (formatted as follows):

CHARINDEX4 consists of:	
WORD	upper-left_character_index
WORD	upper-right_character_index
WORD	lower_left_character_index
WORD	lower_right_character_index

.ADT files (Animation Data)

tP16 will write out a binary file which contains all the background character animation data. The data is setup so that one DMA will update an entire frame.

.ANI files (Animation Macros)

tP16 will write out an ASCII file which contains macro lists for animating background characters. The macros look like this:

ani_dma ANIROOMNAME_DATA+$<FOFFSET>,$<COFFSET>,$<SIZE>

where FOFFSET is the offset into the corresponding .ADT file. COFFSET is the offset into the character set where to copy the data in bytes. SIZE is the number of bytes to copy (DMA). ANIROOMNAME will be the name of the animation room.