

# ORBITER Planetary Textures

Copyright (c) 2000-2017 Martin Schweiger  
Orbiter home: [orbit.medphys.ucl.ac.uk/](http://orbit.medphys.ucl.ac.uk/)

Saturday, September 14, 2019

## Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>1</b>
<b>2</b>	<b>THE FILE LAYOUT.....</b>	<b>2</b>
<b>3</b>	<b>THE QUADTREE STRUCTURE.....</b>	<b>3</b>
<b>4</b>	<b>LAYER TYPES.....</b>	<b>4</b>
4.1	The Surf layer.....	4
4.2	The Mask layer.....	4
4.3	The Elev layer.....	4
4.4	The Elev_mod layer.....	5
4.5	The Cloud layer.....	5
4.6	The Label layer.....	5
<b>5</b>	<b>PLANET CONFIGURATION.....</b>	<b>5</b>
<b>6</b>	<b>THE COMPRESSED ARCHIVE LAYER FORMAT.....</b>	<b>5</b>
6.1	The .tree file format.....	6
<b>7</b>	<b>UTILITIES.....</b>	<b>7</b>
7.1	tileedit.....	7
7.2	plsplit64.....	8
7.3	texpack.....	9
<b>8</b>	<b>THE ELEVATION TILE FILE FORMAT.....</b>	<b>9</b>
<b>9</b>	<b>THE LABEL TILE FORMAT.....</b>	<b>10</b>
<b>10</b>	<b>CONVERTING SURFACE BASE TILES.....</b>	<b>11</b>

## 1 Introduction

This document describes the format Orbiter 2016 uses to define the surface textures, surface elevations and cloud layers for planetary bodies (planets, moons, asteroids, etc.)

Orbiter 2016 introduces a more powerful planetary texturing mechanism than previous releases that provides

- higher resolution levels
- support for surface elevation modelling
- a consistent hierarchical quad-tree implementation for level-of-detail (LOD) mapping over a large resolution range

The legacy texture mapping used in previous Orbiter releases is still supported by Orbiter 2016 for backward compatibility, but its use in new planet texture projects is

discouraged. The legacy model is described in `Orbitersdk\doc\APIGuide.pdf`, Chapter 2.1 “Planet texture maps”.

## 2 The file layout

The files which describe the surface and cloud layer of a planetary body are located in a subdirectory

`Textures\<planet name>`

relative to the Orbiter root directory, where `<planet name>` is a place holder for the name of the celestial body, e. g. `Textures\Earth`.

The folder for each celestial body contains one or several sub-folders for various *layers*. Currently Orbiter supports the following layers:

<code>Surf</code>	Surface textures
<code>Mask</code>	Water masks and night light textures
<code>Elev</code>	Surface elevation maps
<code>Elev_mod</code>	Surface elevation modifiers
<code>Label</code>	Labels for surface features
<code>Cloud</code>	Cloud textures

Only the `Surf` layer is required. All other layers are optional.

The files for each layer are arranged in a hierarchical quad-tree that is reflected by the folder structure within each layer.

Each layer can be represented by multiple resolution levels ( $\geq 1$ ) up to a maximum level of currently 19. The different resolution levels are split into subdirectories represented by 2-digit folder names. For example,

`Textures\Earth\Surf\08`

contains the level-8 surface textures for Earth.

Each resolution level is further subdivided into latitude bands ( $\geq 0$ ), represented by 6-digit subdirectories, e.g.

`Textures\Earth\Surf\08\000005`

contains the surface textures for latitude band 5 at resolution 8. The range of latitude bands `ilat` depends on the resolution level  $n$ :

$$0 \leq ilat \leq nlat-1 \text{ with } nlat = 2^{n-4} \ (n \geq 4)$$

where `ilat` = 0 contains the northernmost latitude band, and `ilat` = `nlat`-1 contains the southernmost latitude band.

Each latitude band subfolder contains the files for the tiles of that latitude band. Each file has a 6-digit file name representing the longitude index of the file, and an extension depending on the layer type. The range of the longitude indices (`ilng`) also depends on the resolution level:

$$0 \leq ilng \leq nlng-1 \text{ with } nlng = 2^{n-3} \ (n \geq 4)$$

where  $ilng = 0$  contains the westernmost longitude tile (left edge at longitude 180°W), and  $ilng = nlng-1$  contains the easternmost longitude tile (right edge at longitude 180°E).

For example, the Earth surface texture tile for resolution level 8, latitude index 5 and longitude index 7 is located in

Textures\Earth\Surf\08\000005\000007.dds

### 3 The quadtree structure

The planet surface is divided into *tiles* along latitude and longitude boundaries. Each tile at resolution level  $n$  can be divided into  $2 \times 2$  sub-tiles at resolution level  $n+1$ . Each tile can therefore be represented as a *node* in a quadtree structure with one parent (except for the root tiles) and up to four children.

The root of the tile quadtree consists of two tiles at resolution level 4:

Textures\<planet name>\<layer>\04\000000\000000.<ext>  
Textures\<planet name>\<layer>\04\000000\000001.<ext>

where file 04\000000\000000.<ext> contains the western hemisphere ( $-180^\circ \leq lng \leq 0^\circ$  and  $-90^\circ \leq lat \leq 90^\circ$ ), and file 04\000000\000001.<ext> contains the eastern hemisphere ( $0^\circ \leq lng \leq 180^\circ$  and  $-90^\circ \leq lat \leq 90^\circ$ ).

The area covered by tile 04\000000\000000.<ext> can then be split into four sub-tiles at level 5:

Textures\<planet name>\<layer>\05\000000\000000.<ext>  
Textures\<planet name>\<layer>\05\000000\000001.<ext>  
Textures\<planet name>\<layer>\05\000001\000000.<ext>  
Textures\<planet name>\<layer>\05\000001\000001.<ext>

In general, a tile with latitude and longitude indices  $[n, ilat, ilng]$  at resolution level  $n$  has subtiles  $[n+1, 2ilat, 2ilng]$ ,  $[n+1, 2ilat, 2ilng+1]$ ,  $[n+1, 2ilat+1, 2ilng]$ ,  $[n+1, 2ilat+1, 2ilng+1]$  at resolution level  $n+1$ .

The latitude and longitude range covered by tile  $[n, ilat, ilng]$  is given by

$$-180^\circ + 360^\circ/nlng*ilng \leq lng \leq -180^\circ + 360^\circ/nlng*(ilng+1) \text{ with } nlng = 2^{n-3}$$

$$90^\circ - 180^\circ/nlat*(ilat+1) \leq lat \leq 90^\circ - 180^\circ/nlat*ilat \text{ with } nlat = 2^{n-4}$$

For example, tile  $[n=8, ilat=5, ilng=7]$  spans the area

$$22.5^\circ \leq lat \leq 33.75^\circ, -101.25^\circ \leq lng \leq -90^\circ$$

Resolution levels 1, 2 and 3 are not part of the quadtree. They cover the entire planet surface at different resolution levels. For example, for the Surf layer, tile 01\000000\000000.dds contains the planet surfaces as a 128×128 pixel texture, 02\000000\000000.dds contains the planet surface as a 256×256 pixel texture, and 03\000000\000000.dds contains the planet surface as a 512×512 pixel texture.

The quadtree doesn't need to be populated completely. Missing tiles are interpolated by Orbiter from a sub-area of an ancestor tile. The quadtree is also allowed to contain gaps, that is, it can contain high-resolution tiles of an area (e.g. a surface base) even if

some of the ancestor trees are missing, so that there is no unbroken chain to the quadtree root.

The minimum requirement is the presence of all tiles for resolution levels 1 to 4 for a given layer. That is, the global tiles (levels 1-3) and the quadtree root (level 4) must be present. All higher resolution quadtree tiles are optional. The maximum currently supported resolution level is 17 for elevation layers (Elev and Elev\_mod), and level 19 for all other layers.

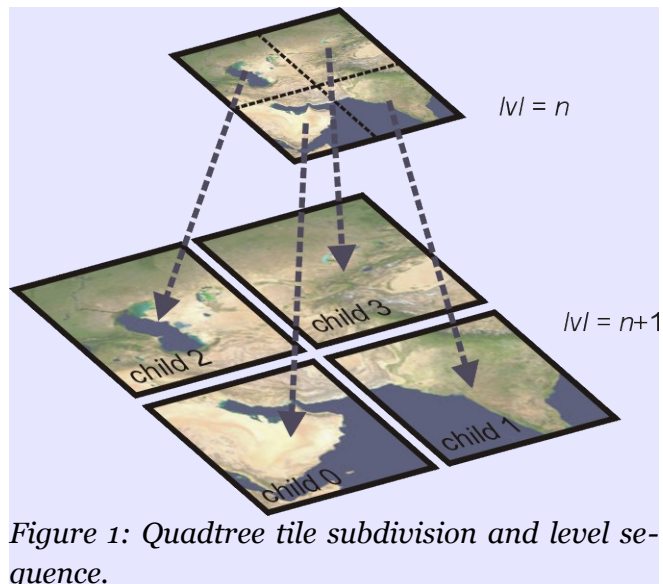


Figure 1: Quadtree tile subdivision and level sequence.

## 4 Layer types

The following layer types are currently supported by Orbiter:

### 4.1 The Surf layer

The Surf layer contains the surface texture of a planet, where the file for each tile is a 512×512 pixel bitmap in DXT1 format (no transparency). You can for example use the `Utils\DxTex.exe` (DirectX Texture Tool) utility to convert a bitmap to DXT1 format. You can also use the `Utils\p1split64.exe` utility to convert a planetary map in cylindrical projection to a set of tile files at a given resolution level.

### 4.2 The Mask layer

The Mask layer contains the planet water mask (for rendering specular reflection of water surfaces) and the night light texture, if applicable, in a combined 512×512 pixel DXT1 texture file (1-bit transparency). The RGB channels of the texture provide the night light information, while the 1-bit alpha channel provides the water mask (alpha=0: water, alpha=1: land). Due to a limitation of the DXT1 format, pixels with alpha=0 do not contain RGB information. Therefore, night lights cannot be defined on water surfaces.

A planet with neither night lights nor specular reflection surfaces does not need to provide a Mask layer.

### 4.3 The Elev layer

The Elev layer contains files defining the surface elevation of a tile in a custom format. (\*.elv, see Section 8). Planets which do not define an Elev layer are represented as spheres.

#### 4.4 The Elev\_mod layer

The Elev\_mod layer allows to modify tile elevations without the need to modify the original Elev files. The Elev\_mod layer can be used e.g. for editing the elevations at surface bases (flattening runways, etc.). The files are in a custom format (\*.elv, see Section 8.) A tile in the Elev\_mod layer can only modify an existing tile in the Elev layer, not create a new tile. Therefore, for each tile in the Elev\_mod layer, the corresponding tile in the Elev layer must exist.

#### 4.5 The Cloud layer

The Cloud layer contains the tiles for a global cloud layer in 512×512 pixel DXT5 format, containing colour and transparency information. Planets without clouds don't define this layer.

#### 4.6 The Label layer

The Label layer contains positions and labels for surface features such as cities, mountains, craters, landing sites, etc. See Section 9 for details.

### 5 Planet configuration

To configure a planet for use of the new surface texture format, its configuration file

`Config\<planet name>.cfg`

must contain the following line:

```
TileFormat = 2
```

If missing, the legacy format is assumed. Likewise, if a cloud layer is present and is defined in the new format, the configuration file must contain

```
CloudFormat = 2
```

otherwise a cloud layer defined in the legacy format is assumed.

The `MaxPatchResolution` tag can be used to specify the maximum resolution level for surface tiles. This can be higher than the maximum level defined in the quadtree, in which case Orbiter interpolates the missing high-resolution tiles. This can be useful for providing a smoother surface representation (in particular for elevation data). The maximum supported value is 19. If the `MaxPatchResolution` value is set to less than the highest quadtree level, those tiles are not used for rendering.

Likewise, the `MaxCloudResolution` tag can be used to specify the maximum resolution level for cloud tiles (the `MinCloudResolution` tag should always be set to 1).

### 6 The compressed archive layer format

The quadtree tile format described in Sections 2 and 3 contains individual files for each tile. This can lead to a very large number of files and a complex directory structure, which is often not desirable. Therefore, Orbiter alternatively supports a compressed archive layer format, where all files for a given planet and layer are compressed into a single file.

The archive files can exist alongside the directory tree containing the individual tile files (the “cache”). You can configure Orbiter to read only from the cache, only from the archive, or from both (first trying the cache, then the archive). Use the

Extra | Visualisation parameters | Planet rendering options | Tile sources

controls for the inline graphics client (orbiter.exe), or the corresponding client-specific controls for external graphics clients.

Compressed archive files have a custom format. They consist of a header, a table of contents (TOC) representing the tree structure, and the compressed and concatenated tile files. Archive files have the extension `.tree` (to signify that they contain quadtree data). They are located in

Textures\<planet name>\Archive

with

Surf.tree	Surf layer
Mask.tree	Water mask and night light layer
Elev.tree	Elevation layer
Elev_mod.tree	Elevation modification layer
Cloud.tree	Cloud layer

## 6.1 The .tree file format

A layer archive (.tree) file consists of a header, TOC and compressed node data. The file header has the following format:

```
struct ZtreeHeader {
    BYTE magic[4];           // file ID and version {'T','X',1,0}
    DWORD size;              // header size [bytes] (48)
    DWORD flags;             // bit flags (currently ignored)
    DWORD dataOfs;           // data block offset (header + TOC)
    __int64 dataLength;       // total length of compressed data block
    DWORD nodeCount;         // total number of tree nodes
    DWORD rootPos1;          // index of level-1 tile
    DWORD rootPos2;          // index of level-2 tile
    DWORD rootPos3;          // index of level-3 tile
    DWORD rootPos4[2];       // index of the level-4 tiles (quadtree roots)
};
```

where `dataOfs` specifies the offset of the data block from the file start (size of header + table of contents), `dataLength` is the size of the data block (= file size – `dataOfs`) and `nodeCount` is the number of entries in the TOC.

`rootPos1`, `rootPos2`, `rootPos3` are the indices of the level 1, 2, and 3 tiles in the TOC, respectively, and `rootPos4` are the indices of the two level 4 quadtree roots in the TOC. If any of these is not present in the TOC, the value is set to (DWORD) -1.

The header is followed by the TOC, consisting of a list of `nodeCount` quadtree node descriptors. A descriptor is given by

```
struct TreeNode {
    __int64 pos;             // file position of node data
    DWORD size;              // data block size [bytes]
    DWORD child[4];          // array index positions of the children
};
```

where `pos` is the node's data position in the file relative to the beginning of the data block, `size` is the node's decompressed data size, and `child` contains the indices of the four descendants, where `(DWORD)-1` terminates the branch.

The compressed data size of node  $i$  can be inferred from `node[i+1].pos - node[i].pos`, where `node[i+1].pos` must be replaced by `dataLength` for the last node.

Note that the TOC may contain nodes without data block, if they are required to provide connectivity across gaps to higher-resolution descendants. The uncompressed size of such dummy nodes is set to 0.

The TOC is followed by the compressed node data, following the same order as the TOC. The data for each node are compressed individually, using the zlib “deflate” method with `Z_DEFAULT_COMPRESSION` setting, essentially providing a gz compression format for each individual data block.

## 7 Utilities

### 7.1 tileedit

`Utils\tileedit.exe` is a tile viewer that is useful for navigating the tile quadtree. It can display multiple layers side by side, and has some basic editing capabilities for elevation tiles.

Tileedit is a Matlab application and requires the Matlab 2015a runtimes installed. The runtimes can be found at <http://uk.mathworks.com/products/compiler/mcr/>

**Important:** Currently, tileedit only works with tile trees represented by individual files, *not* with compressed archive files.

After launching tileedit, select a planet with

File | Open

and navigate to the root folder of a planet texture directory, for example

`Textures\Earth`

Then click “Select Folder”.

This will open the Earth textures at the lowest resolution (level 1). You will see the Surface, Water mask and night light layers side by side.

**Navigation:** Move the mouse over one of the tile images until it is surrounded by a red square. Click on it to proceed to the next higher level. At level 3, you can click on the western or eastern hemisphere to progress to one of the level 4 tiles. At higher levels, you can click on one of the four quadrants to proceed to a child tile.

Moving the mouse to the centre of the image until an “X” appears and clicking will return to the parent tile.

Moving the mouse toward one of the edges until an arrow appears and clicking will move the view to the corresponding neighbour tile.

You can also select a tile directly by entering the resolution level, latitude and longitude index in the input fields at the top left.



Note that tileedit will display a representation of the selected tile even if there is no file for that tile. Tileedit then interpolates a sub-region of an ancestor tile. You can see the source file for a displayed tile below the image. If this differs from the logical tile designation shown in the top left, then the tile has been synthesised from an ancestor.

**Layer selection:** You can select different layers from the drop-down box above an image.

**Elevation data editing:** tileedit allows to modify elevation data.

- Navigate to the tile you want to modify
- Make sure that one of the images shows the elevation layer
- pick an option under “Mouse option”
- Adjust the option-specific parameters
- Move the mouse over the elevation image and click

This will not modify the original Elev layer, but write into the Elev\_mod layer. Note that the modifications are only written to the Elev\_mod file when you navigate away from the modified tile.

You can undo all modifications to an elevation tile by deleting the corresponding file in the Elev\_mod layer.

## 7.2 plspllit64

Utils\plspllit64.exe is an interactive command line utility that allows a planetary surface bitmap to be split into individual tile files at a specific resolution level.

The user must provide the surface bitmap in BMP (24-bit) format at the correct resolution. To generate multiple resolution levels, plspllit64 must be run multiple times with surface bitmaps at the corresponding resolutions.

The surface bitmap must be provided in cylindrical projection (longitude along x-axis, latitude along y-axis). If the entire planet surface is to be processed from a single bitmap, the bitmap must cover longitude from -180° (left edge) to +180° (right edge) and latitude -90° (bottom edge) to +90° (top edge).

To create a level-4 tile set, the source bitmap for global coverage must be 1024×512 pixels in size. For each subsequent resolution level, the bitmap must double in size both in x and y.

In practice, start with the highest resolution supported by your source bitmap (if the size of the source bitmap is not a power of 2, use an image editing tool to interpolate to the closest power of two size).

Then repeat reducing the size of the bitmap by a factor of 2 using an image editing tool to create the next lower level all the way to level 4 (1024×512).

Levels 3, 2, and 1 must be created from source bitmaps 512×512, 256×256 and 128×128, respectively.

If the planet surface contains water areas with specular reflections or night lights, the corresponding source bitmaps for those must also be provided to plspllit64.



### 7.3 texpack

Utils\texpack.exe is a command line utility that packs the individual tile files in a directory tree into a compressed archive and stores it in the Archive subfolder of the planet texture directory. Please be aware that for very large tile trees, the packing operation can take a long time (several hours).

## 8 The elevation tile file format

Orbiter uses a custom file format (\*.elv) for encoding tile elevation data. The data file uses a binary format containing a header and the elevation data.

Each tile file represents a 259×259 grid of elevation points, where column 1 of the grid point matrix corresponds to the left (minimum longitude) edge of the tile, column 257 corresponds to the right edge. Columns 0 and 258 are padding and are used for computing surface normals at the edges. This means that columns 0, 1 and 2 are shared with the left neighbour, and columns 256, 257 and 258 are shared with the right neighbour. The same is true for the top and bottom neighbours. The values stored for the shared nodes must be consistent between neighbouring tiles, otherwise edge artefacts can occur.

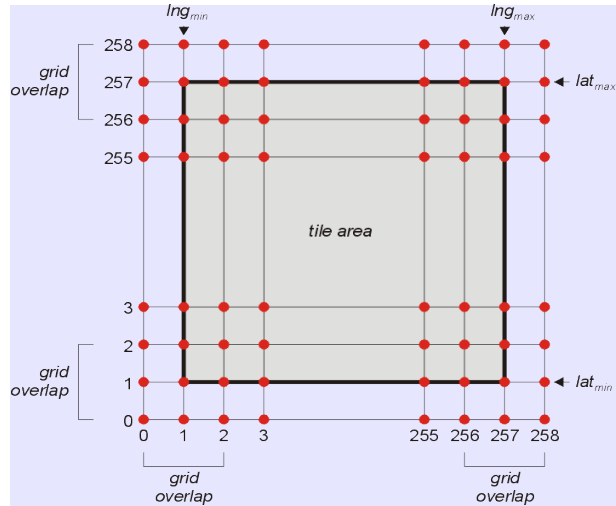


Figure 2: Tile elevation grid and surface cover

The file header consists of the following fields:

```
struct ELEVFILEHEADER { // file header for patch elevation data file
    char id[4];           // ID string + version ('E','L','E',1)
    int hdrsize;          // header size (100 expected)
    int dtype;            // data format (0=flat, no data block;
                        // 8=uint8; -8=int8; 16=uint16; -16=int16)
    int xgrd,ygrd;        // data grid size (259 x 259 expected)
    int xpad,ypad;        // horizontal, vertical padding width
                        // (1, 1 expected)
    double scale;         // data scaling factor (1.0 expected)
    double offset;        // data offset (elevation =
                        // raw value * scale + offset)
    double latmin, latmax; // latitude range [rad]
    double lngmin, lngmax; // longitude range [rad]
    double emin, emax, emean; // min, max, mean elevation [m]
};
```

This is followed by the 259×259 node data in row format, starting from the bottom left corner (min longitude, min latitude). The representation of the data entries depends on the dtype entry in the header file:

dtype = 0: file contains no node data. All nodes have the elevation specified by the offset value.

dtype = 8: node data are provided in unsigned char (8-bit) values

`dtype = -16`: node data are provided in signed short (16-bit little-endian) values  
`dtype = -8` and `16` are not currently used.

In all cases, the data are in units of meters [m]. The elevation resolution is therefore 1m. (The scale value must currently be set to 1.0).

The values represent elevation relative to the planet mean radius. The offset value is added to the node values to compute the elevation value.

The files in the `Elev_mod` layer use the same file format, except that the data block can contain mask flags for individual pixels to signify that this pixel is not to be modified. The mask flag is given by

```
dtype= 8: mask=UCHAR_MAX
dtype=-16: mask=SHRT_MAX
```

Note that the 259×259 pixel elevation tile resolution is higher than the mesh patch resolution used by Orbiter for that tile (which can be user-defined). If the patch resolution has been set to 64×64 (corresponding to 67×67 including edges and padding) then Orbiter will query a subset of a tile's grandparent node to retrieve the mesh elevation data for that tile.

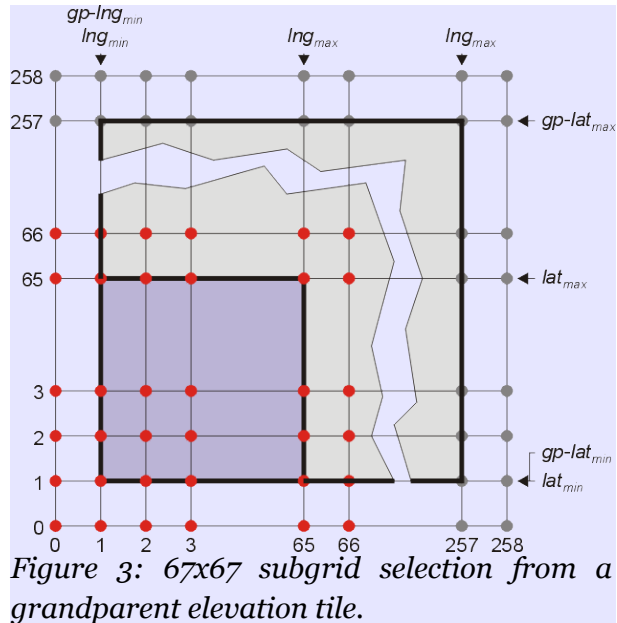


Figure 3: 67x67 subgrid selection from a grandparent elevation tile.

## 9 The Label tile format

Files in the *Label* quadtree folder structure are text files in UTF-8 format and have file extension `.lab`.

Each label file contains the positions and labels of surface features within the confines of the corresponding tile to be displayed at a given resolution level when the user enables surface labels in *Planetarium* mode (Ctrl-F9).

Each line in the file describes one label. The format for each label entry is given by

```
<type-id> <latitude> <longitude> <elevation> <label>
```

where `<typeid>` is a character defining the feature type, `<latitude>` and `<longitude>` are the feature's centre coordinates in degrees, `<elevation>` is the feature's elevation relative to the reference sphere in metres, and `<label>` is a string describing the feature.

`<elevation>` can be set to `NaN`, in which case Orbiter computes the elevation from its terrain elevation data. This should be avoided if possible to reduce computational cost.

The resolution level at which a label appears in the quadtree determines the camera distance at which it appears in the render window. By moving a label to a lower reso-

lution level, it will be visible at a greater distance. For example, large cities should be entered at a lower resolution level than small villages.

Note that in order to reduce the number of quadtree nodes, Orbiter shifts the resolution visibility by four levels. That is, a label entered in a level-2 tile will become visible when the corresponding level-6 tile is rendered.

A label tile file should only contain labels at positions covered by the tile. For example, `Label\05\000000\000001.lab` should only contain features with positions  $0^\circ \leq lat \leq 90^\circ$  and  $-90^\circ \leq lng \leq 0^\circ$ .

To enable the Label layer, the planet's config file

`Config\<planet>.cfg`

should contain the line

`LabelFormat = 2`

otherwise the legacy label format is assumed.

A label layer for a planet should be accompanied by a label legend in file

`Config\<planet>\Label.cfg`

where `<planet>` is the planet's name. `Label.cfg` is a text file in UTF-8 format, where each line describes a label type, with the format

`<type-id> <default-visible> <marker-id> <r> <g> <b> <typename>`

where `<type-id>` is a character defining the type, corresponding to the IDs in the quadtree files, `<default-visible>` is one of characters 'x' (visible by default) or 'o' (not visible by default), `<marker-id>` is a character defining the marker shape: 's' (square), 'o' (circle), 'D' (Delta) or 'N' (Nabla). `<r>`, `<g>` and `<b>` are the red, green and blue components (0-255) of the colour with which the labels are displayed, and `<typename>` is a string describing the label type.

The user can activate and deactivate the display of specific label types by opening the Planetarium dialog (Ctrl-F9), enabling Surface Markers, and opening the Surface Marker Config list to select label types.

Note that Surface base and Navaid markers are currently not included in the quadtree Label layer. They are generated automatically from parsed Base configuration files and the navaid list in the planet's configuration file.

## 10 Converting surface base tiles

The main difference in the definition of surface bases between Orbiter releases 2010 and 2016 is the fact that base tiles (surface textures that provide high-resolution texture content in the vicinity of the base) are no longer supported. Base tiles must now be defined directly in the planet's tile quadtree structure.

The legacy base tile format consisted of individual tile files in the `Textures` folder with the naming format

`<planet name>_<res>_<lngid>_<latid>.dds`

e.g. `Earth_2_W0462_N0161.dds`

where  $\langle res \rangle$  defines the base tile resolution  $m (\geq 1)$ ,  $\langle lngid \rangle$  is the longitude identifier starting with W (west) or E (east) followed by an index  $\geq 0$ , and  $\langle latid \rangle$  is the latitude identifier starting with N (north) or S (south) followed by an index  $\geq 0$ .

The relationship between the base tile resolution designation  $m$  and the quadtree resolution designation  $n$  is given by

$$n = m + 12$$

For base tile resolution level  $m$ , the surface is divided into  $2^{m+9}$  longitude strips and  $2^{m+8}$  latitude strips, with longitude designations

$[W2^{m+8}, \dots, W0, E0, \dots, E2^{m+8}-1]$

and latitude designations

$[S2^{m+7}, \dots, S0, N0, \dots, N2^{m+7}-1]$

Therefore, mapping longitude designations from base tile to quadtree longitude format is given by:

Western hemisphere:  $Wx \rightarrow 2^{m+8} - x$

Eastern hemisphere:  $Ex \rightarrow 2^{m+8} + 1 + x$

and mapping latitude designations from base tile to quadtree format is given by:

Northern hemisphere:  $Ny \rightarrow 2^{m+7} - 1 - y$

Southern hemisphere:  $Sy \rightarrow 2^{m+7} + y$

For example, base tile `Earth_2_W0462_N0161.dds` would translate to quadtree tile `Earth\Surf\14\000350\000562.dds`

Note that the surface tiles in the quadtree are expected to be size 512×512. The base tiles may be present in various resolutions. This has to be taken into account during the conversion. For example, if base tile `Earth_2_W0462_N0161.dds` was provided as a 1024×1024 texture, for conversion to the quadtree format this must be split into four 512×512 tiles and stored in the quadtree as

`Earth\Surf\15\000700\001124.dds`  
`Earth\Surf\15\000700\001125.dds`  
`Earth\Surf\15\000701\001124.dds`  
`Earth\Surf\15\000701\001125.dds`

If `Earth_2_W0462_N0161.dds` was provided as a 256×256 texture, then it can't be directly converted into a quadtree tile. The following options are available:

a) If the appropriate neighbour base textures are present, (in this case,

`Earth_2_W0463_N0161.dds`,  
`Earth_2_W0462_N0160.dds`,  
`Earth_2_W0463_N0160.dds`

they can be combined into a single 512×512 texture and saved as

`Earth\Surf\13\000175\000281.dds`.

b) The texture can be interpolated to size 512×512 using an image editing utility and then saved as a quadtree tile.

c) The 256×256 base tile can be copied into the appropriate quadrant of the original `Earth\Surf\13\000175\000281.dds` parent quadtree tile, if it exists.

Note that the base tiles may need to be edited to colour-match the underlying lower-resolution quadtree tiles to provide a seamless transition. Histogram-matching algorithms can be useful here.