

# Virtual Vellum: Technical Reference

Java version 1.2 has been used to develop Virtual Vellum and all of its custom components, including the multiple document interface and JPEG 2000 decoder. The interface is built using the abstract window toolkit (AWT) as opposed to Swing.

## 1 XML Metadata File

Virtual Vellum identifies collections of images to be displayed using an XML catalogue file. An example XML file is given below:

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <images thumbnailwidth="48" thumbnailheight="72">
3:    <image id="ms_864_flr" name="Besançon, Bibliothèque d'Etude et de
      Conservation, MS 864, f. 1 r" alias="BES MS 864, f. 1 r"
      colourbar="colourbar.jpg" scale="500ppi" offset="20px,20px"
      rotation="0degs" />
4:    <image id="001_t256,256_r20_n6.jp2" name="MS_865 f. 1 r - name"
      alias="MS 865 f. 1 r" colourbar="colourbar.jpg" />
5:    <image id="002_t256,256_r20_n6.jp2" name="MS_865 f. 1 v - name"
      alias="MS 865 f. 1 v" scale="500ppi" offset="20px,20px"
      rotation="0degs" />
6:    <image id="003_t256,256_r20_n6.jp2" name="MS_865 f. 2 r - name"
      alias="MS 865 f. 2 r" scale="500ppi" offset="20px,20px"
      rotation="0degs" />
7:    <image id="004_t256,256_r20_n6.jp2" name="MS_865 f. 2 v - name"
      alias="MS 865 f. 2 v" scale="500ppi" offset="20px,20px"
      rotation="0degs" />
8:  </images>
```

The first line in the XML file provides the parser with some metadata about the file itself. The parser expects UTF-8 encoding to be used. The metadata shown in line 1 can be faithfully copied to other XML metadata image collection files without being changed.

The second line in the XML file tells Virtual Vellum what size the thumbnails should be in the thumbnail preview region. These parameters are in pixel measurements and are also used to determine the visible height of the thumbnail preview region. Any thumbnails larger or smaller than this size are automatically scaled to meet the target size, which is given by the `thumbnailwidth` and `thumbnailheight` parameters. The `images` tag that line 2 opens is closed on line 8. Details about the individual images are enclosed between the opening and closing of the `images` tag, where, in the above example, each image has a separate line (lines 3 to 7).

The metadata about each image is encoded between an `image` tag, which is subsequently closed by the explicit `>` operator at the end of each `<image` line. Virtual Vellum can process the following named fields within the `image` tag:

- `id`: This field identifies the root directory name for a collection of tiled JPEG images or a specific JPEG 2000 image (see section 4 for a review of the expected file structuring).
- `name`: The name field provides a human-readable name for the image which is displayed to the user when they hover over the image in the thumbnail preview region.
- `alias`: A shorter version of the name field is provided by the alias which is used to label the multiple document windows and is also displayed in the bottom left corner of the status bar of Virtual Vellum.

`colourbar`: The `colourbar` key is used to identify a colour bar image which can be displayed when the user shows the colour bar window. The colour bar image is expected to be in the root directory for a collection of JPEG tiled images and in the same directory as the associated JPEG 2000 image. This key is optional.

`scale`: The `scale` value is used to calibrate the rulers. The only acceptable unit is pixels per inch (ppi). This key is optional.

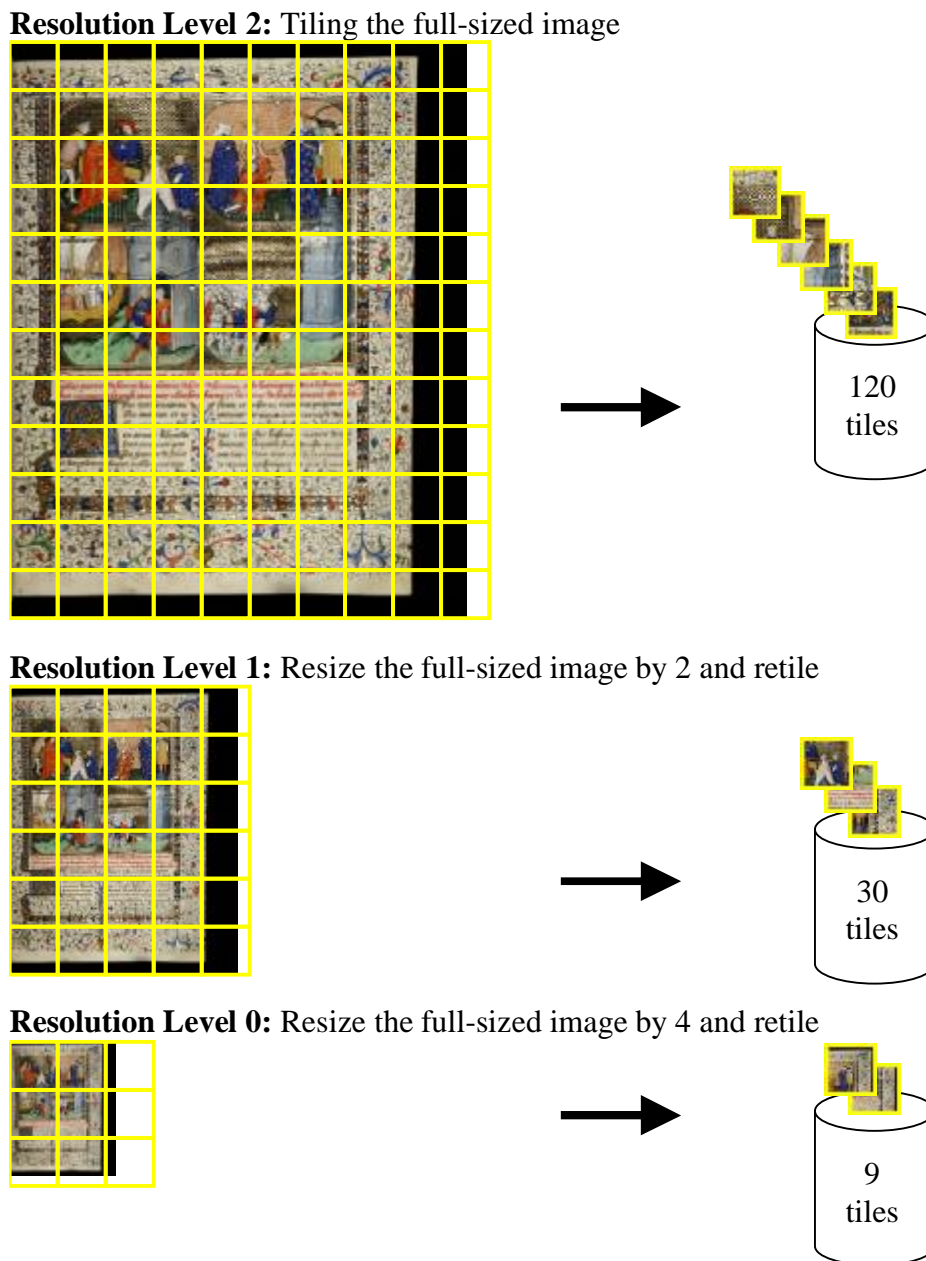
`offset`: This parameter is not currently implemented but reserved for future development to allow the (0, 0) ruler origin to be repositioning anywhere on the image file. The offset will be measured in pixels (px) and have an x and y component. This key is optional.

`rotation`: This parameter is not currently implemented but reserved for future development to allow an image to be rotated if the original is not perfectly axis aligned. The rotation will be measured in degrees (degs). This key is optional.

The ordering of the images in this file is used to order to the thumbnails in the thumbnail preview region of Virtual Vellum.

## **2 JPEG Tiled Images**

When viewing large image files the picture is slit into smaller parts call tiles. Therefore when an image is displayed at a high magnification the software only needs to load and display the tiles for the region that is visible on-screen. However, as the magnification decreases, increasingly more tiles are needed to reconstruct the original picture up to the point where the complete image is displayed and all the tiles need to be loaded and displayed. Therefore at low magnifications, tiling the full-sized image is inefficient. To increase the effectiveness of tiling the image, the full-sized image is rescale to half its original size and tiled again. This provides a collection of lower resolution tiles for the full-sized image which are more suitable for displaying at low magnification levels, especially owing to the fact that each tile needs to be resized to a smaller image anyway. The rescaling process can occur many times, thus producing collections of tiles for different image resolution levels, as illustrated in Figure 1.



**Figure 1: JPEG tiling and rescaling a full-sized image**

Using these collections of tiled images, Virtual Vellum chooses the most appropriate rescaled resolution band and displays the visible tiles from this collection.

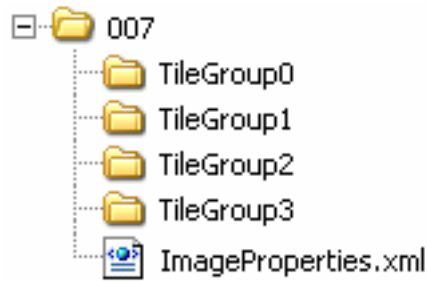
*TIP: For full-sized images that have a resolution of 6,000 x 8,000 pixels, we have found that 5 rescale levels are sufficient when using a tile size of 256 x 256 pixels.*

## 2.1 Data structure

The use of JPEG tiled images is designed to be compatible with the structure used by existing image tiling tools such as Zoomify. Two different JPEG tiling data structures are supported, which are described in the following subsections.

### 2.1.1 TileGroupX folders

The directory structuring expected for this collection of JPEG tiled images is illustrated in Figure 2. Each full-sized image will have an individual folder with a similar structure.



**Figure 2:** Directory structure for TileGroup JPEG tiled image collections

The root collection folder in Figure 2, 007, corresponds to the `id` key entry in the `image` tag's metadata (see section 1). The `ImageProperties.xml` file provides metadata about the tiling used for this image. The metadata is a single line `IMAGE_PROPERTIES` tag as illustrated below:

```
<IMAGE_PROPERTIES WIDTH="5750" HEIGHT="7883" NUMTILES="970" NUMIMAGES="1"
  VERSION="1.8" TILESIZE="256" />
```

The `WIDTH` and `HEIGHT` parameters indicate the dimensions of the full-sized image in pixels. The `NUMTILES` and `TILESIZE` fields give the number of tiles within the complete collection for this image and the tiling size in pixels respectively (the tiles are assumed to be square). Using this information the number of resolution levels (i.e. the number of times the image is resized and tiled) is automatically calculated<sup>1</sup>. The `NUMIMAGES` and `VERSION` keywords are unsupported within Virtual Vellum but exist for compatibility with other JPEG tile reading programs.

The naming of the tiled images should be consistent with the following format, where `resolutionlevel` gives the rescaling level (see Figure 1), `xpos` gives the zero-based index of the tile along the horizontal axis and `ypos` gives the zero-based index of the tile along the vertical axis (the origin is the top left).

```
resolutionlevel-xpos-ypos.jpg
```

The largest reduced full-sized image is mapped to a resolution level 0, where the non-scaled full-sized image maps to the largest resolution level used. Table 1 illustrates this mapping.

Resolution Level	Resize Factor	Resulting image size
5	1/1	6,000 x 8,000
4	1/2	3,000 x 4,000
3	1/4	1,500 x 2,000
2	1/8	750 x 1,000
1	1/16	375 x 500
0	1/32	187 x 250

**Table 1:** Relationship between resolution level and image size

For example, a filename of `0-0-0.jpg` represents the smallest rescaled image size and the top left corner of it, whereas `4-1-12.jpg` represents the second tile from the left and the 13<sup>th</sup> one down from the 4<sup>th</sup> resolution level resized image.

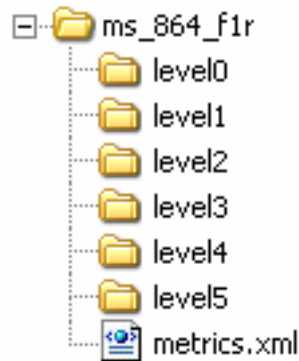
The tiled images are stored in the `TileGroup` folders in groups of 256 files. The grouping is achieved by storing the lower resolution bands first and when a resolution band is split across

<sup>1</sup> It is always assumed that the image resizing process progressively halves the width and height at each step

TileGroup folders it is done in a scanline order (i.e. rows are completed before columns). Therefore TileGroup0 will always contain the 0-0-0.jpg tile and the final TileGroup will always contain the bottom right tile of the full-sized image.

### 2.1.2 LevelX folders

The directory structuring expected for this collection of JPEG tiled images is illustrated in Figure 3. Each full-sized image will have an individual folder with a similar structure.



**Figure 3:** Directory structure for Level JPEG tiled image collections

The root collection folder in Figure 3, `ms_864_fr1`, corresponds to the `id` key entry in the image tag's metadata (see section 1). The `metrics.xml` file provides metadata about the tiling used for this image. The metadata is a single line `metrics` tag as illustrated below:

```
<metrics width="6000" height="7600" tilesize="256" />
```

The `width` and `height` fields indicate the dimensions of the full-sized image. The `tilesize` field indicates the size of each square tile in pixels.

The naming of the tiled images should be consistent with the following format, where `resolutionlevel` gives the rescaling level (see Figure 1), `xpos` gives the zero-based index of the tile along the horizontal axis and `ypos` gives the zero-based index of the tile along the vertical axis (the origin is the top left).

```
resolutionlevel_xpos_ypos.jpg
```

The largest reduced full-sized image is mapped to a resolution level 0, where the non-scaled full-sized image maps to the largest resolution level used. Table 1 illustrates this mapping. For example, a filename of `0_0_0.jpg` represents the smallest rescaled image size and the top left corner of it, whereas `4_1_12.jpg` represents the second tile from the left and the 13<sup>th</sup> one down from the 4<sup>th</sup> resolution level resized image.

The tiled images are stored in the `level` folders, where all tiles associated with a given level are stored in the same folder. Therefore all resolution level 0 JPEG tile images are stored in the `level0` folder whereas all resolution level 4 JPEG tile images are stored in the `level4` folder.

## 3 JPEG 2000

JPEG 2000 facilitates multiresolution tiling within a single file. Therefore there is no need to manually break a large image into many smaller JPEG tiles or provide redundant resolution levels. When a JPEG 2000 file is decoded only the information specific to the tile and resolution

level being decoded is extracted from the file. For a comprehensive review of the JPEG 2000 format, the reader is referred to <http://www.jpeg.org/jpeg2000>.

A JPEG 2000 image is only a single file and thus there is no need to store all of its components in a subfolder. The JPEG 2000 file therefore resides in the root of the `images` collection folder, which is the same folder that each of the tiled JPEG collection root folder resides (see section 4). Furthermore, the JPEG 2000 file contains all the decoding metadata required and so there is no need for a separate XML file.

*TIP: For full-sized images that have a resolution of 6,000 x 8,000 pixels, we have found that 5 resolution levels are sufficient using a tile size of 256 x 256 pixels.*

### 3.1 Virtual Vellum JPEG 2000 decoder limitations

In order to reduce the compiled and run-time memory footprint of the JPEG 2000 decoder, a number of assumptions have been made about the JPEG 2000 images which must hold for Virtual Vellum to successfully open an image<sup>2</sup>. The assumptions are as follows:

- There is no sub-precinct information
- There is only one layer
- There are no progression levels, i.e. there is only one layer and no additional layers that add further resolution of the image
- The decoder assumes LRCP order where  $L=P=1$  (layers=precincts=1)
- Each tile is encoded in exactly the same manner<sup>3</sup>
- Tile sizes should not exceed  $2^{15}=32,768$  in width or height
- Only the 5-3 wavelet scheme (integer based) is fully implemented

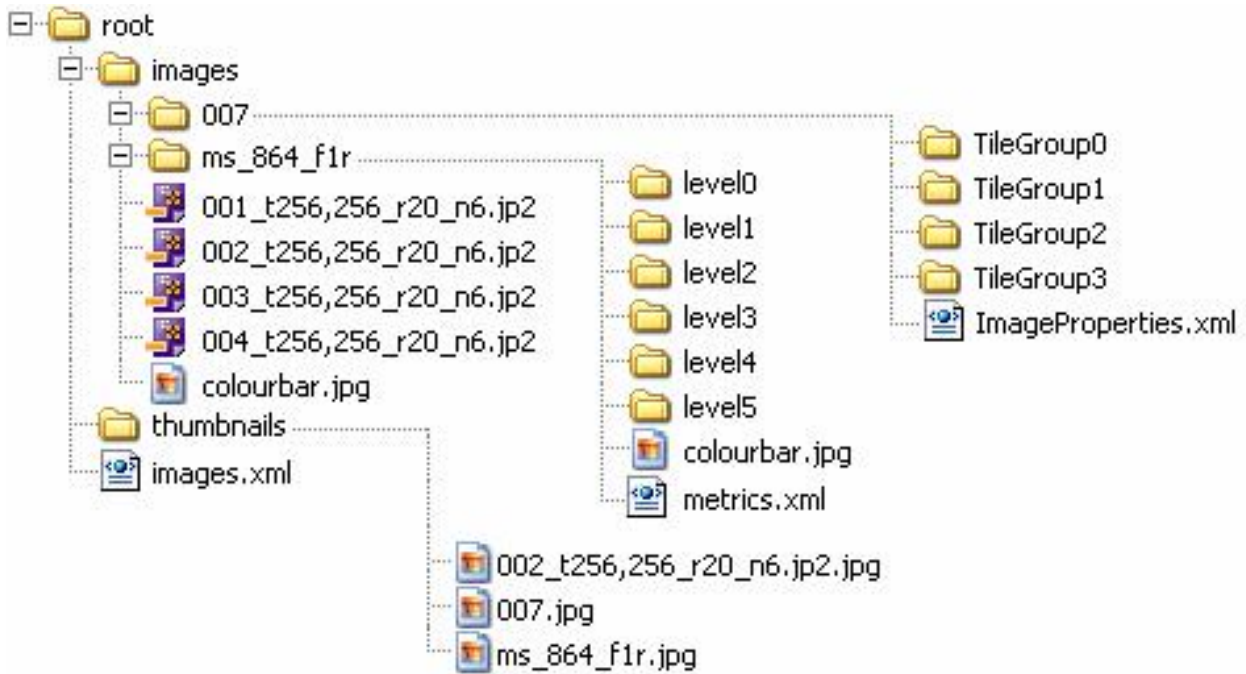
## 4 File Structure

Virtual Vellum expects a rigid file structuring for the images and their location. When the software is run, an XML file is specified, including its path. The folder that contains this file is the root folder for all images and metadata. The general structure is illustrated in Figure 4, where the root XML file is called `images.xml`.

---

<sup>2</sup> The Virtual Vellum source code provides the framework for adding extra functionality to eliminate certain assumptions in future developments.

<sup>3</sup> There is scope to encode tiles in different ways but this is not fully compliant with the JPEG 2000 standard at present. The source code should be consulted for further details



**Figure 4:** Directory structure for the Virtual Vellum images and metadata

The key directories that must exist are the `images` and `thumbnails` folders that are direct child objects of the root folder. The `images` folder contains all the images relating to a full-sized image (including metadata, resized images and JPEG 2000 images). The `thumbnails` folder contains a collection of thumbnail-sized images of the full-sized image for quick display within the thumbnail preview region of Virtual Vellum.

Figure 4 illustrates how the different image formats and tiling methodologies can be mixed together in the `images` folder; `007` is a `TileGroup` JPEG tiled image, `ms_864_fr1` is a `level` JPEG tiled image and all the `.jp2` files are JPEG 2000 images.

For a thumbnail image to be displayed in the thumbnail preview region of Virtual Vellum of a tiled JPEG image, a picture with the filename corresponding to the image collection folder must exist within the `thumbnails` folder. In Figure 4 there are JPEG thumbnails called `007.jpg` and `ms_864_fr1.jpg` in the `thumbnails` folder that correspond to the folders in the `images` directory.

It is not strictly necessary to have a thumbnail of the JPEG 2000 images. However to improve the display time of the JPEG 2000 thumbnail in the thumbnail preview region of Virtual Vellum it may be desirable. This can be achieved by creating a thumbnail in the `thumbnails` folder using the same name as the JPEG 2000 image with `.jpg` on the end of the filename. For example, in Figure 4, `002_t256,256_r20_n6.jp2.jpg` is a JPEG thumbnail for the full-sized `002_t256,256_r20_n6.jp2` JPEG 2000 image.

The `colourbar.jpg` files represent the colour bar images that Virtual Vellum uses. For JPEG tiled images, the colour bar image should be placed in the folder that is the root folder of a single image that contains the collection of images and metadata (see Figure 4). The colour bar image for JPEG 2000 images should be stored in the same folder as the JP2 files (see Figure 4).

## 5 Embedding Virtual Vellum within a Webpage

In addition to running Virtual Vellum as a standalone application from the desktop, it can also be run from within a web browser environment as an applet. The HTML required to embed Virtual Vellum into a web page is illustrated below:

```
<APPLET CODE="WebLaunch.class" ARCHIVE="vv.jar" WIDTH=100 HEIGHT=50>
  <PARAM NAME="xmlfile" VALUE="images.xml" />
</APPLET>
```

The description encapsulated by the `APPLET` tag (closed by `</APPLET>`) provides the web browser with all the details required to locate and run Virtual Vellum. The `ARCHIVE` field identifies the Java program code, which is compressed into a single JAR file – `vv.jar`. The `CODE` field specifies which class the web browser should run as an applet from code within the JAR file. When running Virtual Vellum as an applet, this class should be `WebLaunch` (as opposed to `VirtualVellum` when running from the desktop). The `WIDTH` and `HEIGHT` parameters tell the web browser how much space the applet consumes on the page, thus allowing the remainder of the page to be displayed around the application. Since the main Virtual Vellum application runs in a separate popup window, the width and height values are only required to display a “Re-Launch Viewer” button. This button is used to redisplay Virtual Vellum if the user hides the application. *(Note: The web applet is always live when the web page is open. Therefore, if the user “exits” the application when it is running as an applet then the window is only hidden. The applet is fully exited when the user navigates away from the web page that embeds the applet.)*

The `PARAM` tag provides initialisation information to Virtual Vellum. The `NAME` field identifies a parameter, called `xmlfile`, which is set to the string given by the `VALUE` field. Therefore in the above example we have `xmlfile=images.xml`. When Virtual Vellum runs, it looks for the `xmlfile` parameter and retrieves the image list from the file identified by the value of this parameter (i.e. `images.xml`). This is similar to command line parameters used when running Virtual Vellum as a standalone desktop application.

In the above example, the path of both the JAR and XML files is specified as a relative address. However these paths could equally be a fully qualified URL. *It is important to note that due to applet security permissions, when running Virtual Vellum as an applet, you are prohibited from accessing data files from your local hard drive. Therefore if you wish to access a local dataset, you must run the application in its desktop mode (please see the user guide for further information).*

## 6 Behind the Scenes of Virtual Vellum

The following list details some of the technical achievements of Virtual Vellum that sit within the code, out of general sight of the user. This list is only intended to provide an overview, where the source code should be the main reference point for further details:

- Multiple processor support for decoding image tiles simultaneously
- Intelligent cache policy:
  - When a user is viewing a JPEG 2000 image and the decoding threads are sitting idle, the rest of the image data is loaded in the background to reduce subsequent tile decodes that would normally require data from the network.
  - The JPEG 2000 image data is cached locally when it is loaded until a new image is loaded.
  - When skipping over regions of a JPEG 2000 image, the system will dynamically determine whether it is quicker to perform a “skip and disregard” or a close and

reopen connection operation (where the reopen operation immediately positions the next read to the desired file location). (When connecting to images over the web using HTTP, a skip operation is actually a read and disregard operation therefore a significant amount of time can be required to skip over large regions of unneeded file data.)

- A “skip and disregard” operation has been recoded to perform a “skip and cache”, thus increase efficiency (note that if a large skip is required and it would be quicker to reopen the file then the reopen approach is still taken).
- JPEG 2000 tile decoding is ordered such that earlier tiles are decoded before those that appear later in the file. This reduces the need to step back through a data file (if the data is not already cached) and hence improves overall performance; a HTTP skip can only go forwards and therefore a close and reopen operation is always needed under these circumstances.
- Automatic JVM detection which provides extra software functionality on later versions, such as the mouse scroll wheel