1 The API

1.1 Compression

```
public class wv4DBWCompressor
  public wv4DBWCompressor( float p, float threshold,
                           java.io.DataOutputStream outStream,
                           int actualNumFirstTimepoint,
                           String baseFileName,
                           int numTimepointsCompressed,
                           int numBitfilesCreated,
                           int numFocalplanesInStack,
                           int numBlocks,
                           int numTimepointsInEachBlock,
                           short grayLevel) throws java.io.IOException
  public void setBlockSize(int width, int height,
                           int numFocalPlanesInaBlock)
  public void setImage(Image img, int spaceIndex, int timeIndex)
  public void setImage(int[][] val, int spaceIndex, int timeIndex)
  public void doWaveletTransform()
  public void writeEncoded() throws java.io.IOException
```

1.1.1 Preconditions for the parameters to the constructor

Some parameters are crucial for the wavelet compression to proceed correctly, whereas other parameters are only meant to be used to store bookkeeping information in the .bit file. Hence the distinction 'core/info'.

`Parameter`	allowed values	Core/ Info	Use/Intended use
`p`	any	core	The l_p space in which the quantization should take place.
`threshold`	> 0	core	The quantization level.
`outStream`	$\neq$ null	core	The output stream to which the bitstream will be appended.
`actualNum-FirstTime-point`	any	info	The actual number of the first timepoint. E.g. if this class is compressing stack6.tif, ..., stack10.tif, this could be set to the value of 6.
`baseFilename`	$\neq$ null	info	The base of the filename, e.g. when compressing stack1.tif, ... stack10.tif, this could be 'stack'.

`numTimepointsCompressed`	any	info	The total number of timepoints compressed. E.g. when compressing stack1.tif, ... stack10.tif in this session this could be 10, even if this class is only compressing stack6.tif, ..., stack10.tif. Compare with `numTimepointsInEachBlock`.
`numBitfilesCreated`	any	info	The number of .bit files created when compressing the current data set.
`numFocalplanesInStack`	any	info	The total number of focal planes in each of the stacks, stack1.tif, ..., stack10.tif
`numBlocks`	> 0	core	The number of blocks this class will be used to compress.
`numTimepointsInEachBlock`	> 0	core	The number of timepoints in each block that will be fed into this class for compression.
`grayLevel`	enum	core	The gray level of this image. It's value is restricted to `wvGraylevels.GRAY8BIT`, `wvGraylevels.GRAY16BIT`, `wvGraylevels.GRAY24BIT`

1.1.2 Pre- and postconditions

-
  ```
  public void setBlockSize(int width, int height,
                           int numFocalPlanesInaBlock)
  ```

  1. `width` > 0
  2. `height` > 0
  3. `numFocalPlanesInaBlock` > 0

  This makes the class ready to accept image data.

-
  ```
  public void setImage(Image img, int spaceIndex, int timeIndex)
  public void setImage(int[][] val, int spaceIndex, int timeIndex)
  ```

  1. The method `setBlockSize` must have been called.
  2. $0 \leq$ `spaceIndex` $<$ `numFocalPlanesInaBlock`
  3. $0 \leq$ `timeIndex` $<$ `numTimepointsInEachBlock`
  4. The size of the image has to be consistent with `setBlockSize`:
     (a) `img.getHeight() = height`, `img.getWidth() = width`.

   (b) `val.length = height`, `val[i].length = width`.

- `public void doWaveletTransform()`

   1. The method `setBlockSize` must have been called.

  This computes the wavelet transform of the data that has been input through `setImage` after the last call to `setBlockSize`.

- `public void writeEncoded()`

   1. The method `public void doWaveletTransform()` must have been called.
   2. The method `setBlockSize` must not have been called after the last call to `doWaveletTransform`.

  The class writes to `outStream` the wavelet coefficients computed during the last call to `doWaveletTransform`.

1.2 Decompression

```
public class wv4DBWDecompressor
  public wv4DBWDecompressor(java.io.DataInputStream inStream)
                       throws java.io.IOException,  Exception
  public void readEncoded() throws java.io.IOException
  public void doInverseWaveletTransform()
  public Image getImage(int spaceIndex, int timeIndex)
  public void getImage(int[] val, int spaceIndex, int timeIndex)
  public void getImage(int[][] val, int spaceIndex, int timeIndex)
  public int getNumberOfBlocks()
  public int getWidth()
  public int getHeight()
  public int getNumberOfPlanesInCurrentBlock()
  public int getNumberOfTimepointsInEachBlock()
  public int getNumberOfPlanesInStack()
  public int getActualNumberOfFirstTimepoint()
  public int getNumberOfTimepointsCompressed()
  public int getNumberOfBitfilesCreated()
  public String getBasefilename()
  public short getGrayLevel()
```

1.2.1 Pre- and postconditions

-
  ```
  public wv4DBWDecompressor(java.io.DataInputStream inStream)
                         throws java.io.IOException,  Exception
  ```

   1. `inStream` $\neq$ `null`
   2. `inStream` is positioned at the start of a segment created by `wv4DBW-Compressor`

The class reads basic information from the stream, awaits further commands before proceeding with the reading and decompression.

- `public void readEncoded() throws java.io.IOException`

  Reads the next block from `inStream`.

- `public void doInverseWaveletTransform()`

  1. `readEncoded` has been successfully called.

  Inverts the wavelet coefficients read in the last call to `readEncoded`.

-
  ```
  public Image getImage(int spaceIndex, int timeIndex)
  public void getImage(int[] val, int spaceIndex, int timeIndex)
  public void getImage(int[][] val, int spaceIndex, int timeIndex)
  ```

  1. `doInverseWaveletTransform` has been called.
  2. $0 \leq$ `spaceIndex` $<$ `getNumberOfPlanesInCurrentBlock()`
  3. $0 \leq$ `timeIndex` $<$ `getNumberOfTimepointsInEachBlock()`
  4. If `val` is passed as an input parameter, it must be of the appropriate size, so it has to satisfy one of:
     (a) `val.length = getHeight()*getWidth()`
     (b) `val.length = getHeight()`, `val[i].length = getWidth()`

     depending on whether `val` is of type `int[]` or `int[][]`.

  Returns an image from the block most recently read in `readEncoded`.

1.2.2 Methods accessible right after construction

```
public int getNumberOfBlocks()
public int getNumberOfPlanesInStack()
public int getActualNumberOfFirstTimepoint()
public int getNumberOfTimepointsCompressed()
public int getNumberOfBitfilesCreated()
public String getBasefilename()
public short getGrayLevel()
```

All these methods are accessible right after the class has been constructed and they return values read from `inStream`. Their return values are the same as the input values to `wv4DBWCompressor`.

1.2.3 Method accessible after the first call to `readEncoded`

In addition to the methods listed above, the following are accessible when the method `readEncoded` has been called.

```
  public int getWidth()
  public int getHeight()
  public int getNumberOfPlanesInCurrentBlock()
  public int getNumberOfTimepointsInEachBlock()
```

Their return values are the same as the input values to `wv4DBWCompressor`.
Note: The return values are undefined if `readEncoded` has not been invoked.

1.3 Estimation of threshold

The class `wvFindThreshold` can be used to estimate the threshold needed to achieve a given compression ratio.

The user chooses a 4D block of images on which the calculations will be performed, feeds that block into an instance of wvFindThreshold and calls the method `findBounds`. The results are returned in the form of `wvThresholdAndValue`.

Care has been taken to make this class thread-safe. The methods are classified as belonging to either of 2 categories:

T	All methods in this group may be invoked in different threads while any other method in this group is still running.
U	All other methods, both **T** and **U** , must have completed before an **U** method is invoked.

```
public class wvFindThreshold

  public wvFindThreshold(String filename,
                       int numFocalplanesInBlock,
                       int numTimepointsInBlock,
                       int width, int height, double p,
                       short grayLevel)                            U
  public wvThresholdAndValue findBounds(double startThreshold,
                                    double targetValue)
                                    throws Exception               U
  public synchronized void findBoundsThreaded(double startThreshold,
                                            double targetValue)    T
  public void setImage(int[][] val, int spaceIndex, int timeIndex)   U
  public void setImage(Image img, int spaceIndex, int timeIndex)     U
  public synchronized boolean maxItersExceeded()                     T
  public synchronized boolean errorOccurred()                        T
  public synchronized boolean doneCalculating()                      T
  public synchronized wvThresholdAndValue getBestPoint()             T
  public synchronized String getMessage()                            T
  public final static double START_THRESHOLD

public class wvThresholdAndValue
  public double m_threshold = 0;
  public double m_value = 0;
```

1.3.1 Preconditions for the parameters to the constructor

`Parameter`	allowed values	Use/Intended use
`filename`	$\neq$ null	The name of a file in which we can save temporary .bit files
`numFocal-planesInBlock`	> 0	The number of focalplanes in the block that will be fed into this class for compression.
`numTime-pointsInBlock`	> 0	The number of timepoints in the block that will be fed into this class for compression.
`width`	> 0	The width of the images that will be fed into this class.
`height`	> 0	The height of the images that will be fed into this class.
`p`	any	The l_p space in which the quantization should take place.
`grayLevel`	enum	The gray level of this image. It's value is restricted to `wvGraylevels.GRAY8BIT`, `wvGraylevels.GRAY16BIT`, `wvGraylevels.GRAY24BIT`

1.3.2 Preconditions to other methods

- `public void setImage(Image img, int spaceIndex, int timeIndex)` **U**

  `public void setImage(int[][] val, int spaceIndex, int timeIndex)` **U**

  1. $0 \leq$ `spaceIndex` $<$ `numFocalPlanesInBlock`
  2. $0 \leq$ `timeIndex` $<$ `numTimepointsInBlock`
  3. The applicable one of:
     (a) `img` $\neq$ `null`, `img.getHeight() = height`, `img.getWidth() = width`.
     (b) `val.length = height`, `val[i].length = width`.
  4. `findBounds` has not been called.

- `public synchronized boolean maxItersExceeded()` **T**

  1. No preconditions.

  Returns `true` if either either `findBounds` or `findBoundsThreaded` have been called and the maximum number of iterations was exceeded in the last call to these methods. Returns `false` otherwise.

-
  ```
  public wvThresholdAndValue findBounds(double startThreshold,
                                        double targetValue)
                                        throws Exception            U
  ```

  1. The method `setImage` must have been called to feed all the images into this class.
  2. `startThreshold` > 0 is an initial guess of the threshold. Recommended value is `wvFindThreshold.START_THRESHOLD`.
  3. `targetValue` > 1 is the desired compression ratio. Would be 100 for a compression rate of 1:100.
  4. Previous calls to `findBounds` or `findBoundsThreaded` must have completed.

  The method returns the estimated threshold and the actual compression ratio calculated. If it fails to calculate an estimated threshold, an exception is thrown.

-
  ```
  public synchronized void findBoundsThreaded(double startThreshold,
                                              double targetValue)  T
  ```

  1. This method has the same preconditions as `findBounds`.

  The method returns immediately and spawns another thread that calls `findBounds`. Note: this method does not throw any exceptions. See `boolean errorOccurred()` and `boolean doneCalculating()`.

-
  ```
  public synchronized boolean errorOccurred()                        T
  ```

  1. No preconditions.

  Returns `true` if `findBoundsThreaded` has been called and the last call to it resulted in an error. Returns `false` otherwise. The errors that can occur include, but are not limited to, that the maximum number of iterations was exceeded, see `boolean maxItersExceeded()`.

-
  ```
  public synchronized boolean doneCalculating()                      T
  ```

  1. No preconditions.

  Returns `true` if `findBoundsThreaded` has been called and the method has finished its calculations. Returns `false` otherwise.

-
  ```
  public synchronized wvThresholdAndValue getBestPoint()             T
  ```

  1. No preconditions.

If neither `findBounds` nor `findBoundsThreaded` have been called, the method will return an `wvThresholdAndValue` object with a negative threshold and a negative compression ratio. If either one has been called, it's the last one of these calls that determines the behavious of this method.

1. If `findBoundsThreaded` was the last of the two methods to be invoked:
   (a) If it has been started, but has not completed and it has calculated an estimate of the compression ratio, it will return the best estimate calculated so far.
   (b) If the method has been started, but has not completed and has not calculated an estimate of the compression ratio, the object returned will have a negative threshold and a negative compression ratio.
   (c) If the method has run to completion [see `doneCalculating`] and `errorOccurred` returns `false`, the method returns the best estimate calculated.
   (d) If the method has run to completion [see `doneCalculating`] and `errorOccurred` returns `true`, the object returned will have undefined values.
2. If `findBounds` was the last of the two methods to be invoked and it has run to completion:
   (a) If no exception was thrown, this method will return the best estimate calculated in the call to that method.
   (b) If an exception was thrown, the object returned will have undefined values.

`public synchronized String getMessage()` **T**

1. No preconditions

This class maintains a message queue to deliver messages to the user. All messages are put into this queue as opposed to calling System.out.println(). This method returns `null` if no messages are in the queue. If one or more messages are in the queue, it returns the first message and removes it from the queue.