TER user manual

(version 1.1)

GICI group

Department of Information and Communications Engineering
Universitat Autònoma Barcelona
http://www.gici.uab.es - http://www.gici.uab.es/TER

January 2007

1 Overview

TER is an implementation of the CCSDS Recommendation for Image Data Compression (Recommended Standard CCSDS 122.0-B-1 Blue Book). TER implements not only the CCSDS Recommendation, but also new features add to the Recommended Standard. TER is designed and programmed with the aim to provide a good basis to test and develop the CCSDS Recommendation for Image Data Compression. The application provides different functionalitites that can be controlled via its parameters. These parameters can be passed as program arguments, selecting some of the algorithms that the application incorporates. More than 30 parameters allow the combination of different processes to produce a compressed image codestream. Each one of these parameters and its valid arguments are explained in the application help and in this manual. When added features are used, the encoded file is not compilant with the Recommendation. In these cases, in the decoding process the user must employ a parameter -rc no / –recommendation no which indicate that the file is not compilant with the recomendation. Parameters employed in the added features are also required as program arguments to correctly perform the decoding process.

A development manual is also available and the API is well documented to facilitate an easy understanding, extension and modification of the application. All the design and implementation details are widely explained in these manuals and the source code is commented. All these information is publicly available in the web page http://www.gici.uab.es/TER, where you can download the sources and the manuals. To guarantee a free distribution, TER has the General Public License (GNU GPL) of the Free Software Foundation (http://www.fsf.org/licensing/licenses/gpl.html).

TER incorparates algorithms to make it flexible enough to manage different kind of data, to control and monitor the compression stages, to extract statistical information, etc. As we explained above TER is a platform to test new ideas, not to be a commercial product or to be integrated to other applications that need support for encoding/decoding files compilant with the CCSDS Recommendation for Image Data Coding. The double functionality of the application makes that the application performance cannot be as good as some other implementations, so for commercial purposes the GICI group does not recommend the use of this application.

We have tried to make a good design and efforts have been made to develop and implement the application as useful as desirable. However, we could not foresee all the needs of TER users, so we will thank you for all the suggestions and comments that you can report to us (gici-dev@abra.uab.es).

We hope you enjoy it,

GICI group

2 Usage

TER is divided in 2 applications: the encoder and the decoder. Both programs are encapsulated in jar files in the *dist/* directory and can be executed separately using the JVM. Each application accepts its own parameters which can be passed as simple command arguments, i.e. *java -jar dist/TERcode.jar -i inputImage.pgm*. The distribution incorporates a shell script both for the encoder and the decoder called *TERcode* and *TERdecode* to facilitate executions in a GNU/Linux environment. TER may use a great amount of memory. Hence, it is recommended to set the maximum amount of memory that the application can allocate (usually the same as you computer RAM) via the *-Xmx* parameter of the JVM, i.e. *java -Xmx512m -jar dist/TERcode.jar -i inputImage*.

The usage of TER is very simple. To compress an image you can use the encoder with some specific parameters. All the functionalities of TER are explained with the application parameters, so it is recommended to read them to know what you can do. Do not worry if you select incompatible parameters or functionalities; the application detects these problems and displays warning messages to the user.

When you have a compressed codestream, you can use the decoder to recover the original image and save it with other common formats. TER decoder can manage codestreams compliant with the CCSDS Recommendation for Image Data Coding generated by other applications. However not all the options defined in the Recommended Standard are implemented. TER decoder can also manage specific TER files not compilant with the Recommendation. In such cases, it is required that the user indicates that the file is not compilant and that passes to the decoder the parameters using by the encoder as program parameters.

A typical use of the application is as follows:

```
    # TERcode -i lena.pgm -o lenaCompressed.ter
    # TERdecode -i lenaCompressed.ter -o lenaRecovered.pgm
```

or if you do not use the shell scripts:

```
    # java -Xmx512m -jar dist/TERcode.jar -i lena.pgm -o lenaCompressed
    # java -Xmx512m -jar dist/TERdecode.jar -i lenaCompressed.ter -o lenaRecovered.pgm
```

3 Parameters

In this section all supported parameters both for the encoder and the decoder are explained. This documentation is extracted from the application help and it explains all the algorithms incorporated to TER. By reading them you will know all the functionalities of the application.

Parameters have two formats: the long and the short specification. Long specification has −− at the beginning while short specification has − (it does not matter which one you choose). Each parameter has its own arguments, which usually are integers, floats, booleans (0 to indicate false and 1 to indicate true) or strings. If the user specifies some invalid arguments, the application will display warning messages. Most of these parameters are not mandatory. When they are not specified default values are used. The following table shows how each parameter will be displayed in this manual:

−−**longParameter**	{parameter arguments}
−**shortParameter**	*Mandatory:* Yes/No
Explanation:	Parameter explanation
Default:	Parameter default values.

3.1 TERcoder parameters

−−**help**	
−**h**	*Mandatory:* No
Explanation:	Displays this help and exits program.
Default:	

−−**inputImage**	{string}
−**i**	*Mandatory:* Yes
Explanation:	Input image. Valid formats are: pgm, ppm, pbm, jpg, tiff, png, bmp, gif, fpx. If image is raw data file extension must be ”.raw” or ”.img” and ”-g” parameter is mandatory.
Default:	

−−**imageGeometry**	{int int int int boolean}
−**g**	*Mandatory:* No
Explanation:	Geometry of raw image data. Parameters are: 1- zSize (number of image components) 2- ySize (image height) 3- xSize (image width) 4- data type. Possible values are: 0- boolean (1 byte) 1- unsigned int (1 byte) 2- unsigned int (2 bytes) 3- signed int (2 bytes) 4- signed int (4 bytes) 5- signed int (8 bytes) 6- float (4 bytes) 7- double (8 bytes) 5- Byte order (0 if BIG ENDIAN, 1 if LITTLE ENDIAN) 6- 1 if 3 first components are RGB, 0 otherwise.
Default:	

−−**outFile**	{string}
−**o**	*Mandatory:* No
Explanation:	Output image file name.
Default:	workDir/default.ter

−−**part2Flag**	{int[int[int[...]]]}
−**h2**	*Mandatory:* No
Explanation:	Indicates for each segment the presence of Part 2 header. 0 - Part 2 header absent 1 - Part 2 header present
Default:	1

−−**part3Flag**	{int[int[int[...]]]}
−**h3**	*Mandatory:* No
Explanation:	Indicates for each segment the presence of Part 3 header. 0 - Part 3 header absent 1 - Part 3 header present
Default:	1

−−**part4Flag**	{int[int[int[...]]]}
−**h4**	*Mandatory:* No
Explanation:	Indicates for each segment the presence of Part 4 header. 0 - Part 4 header absent 1 - Part 4 header present
Default:	1

−−**segByteLimit**	{int[int[int[...]]]}
−**bl**	*Mandatory:* No
Explanation:	Maximum number of bytes that can be used in a segment. The value of SegByteLimits includes bytes used for the header, and is applied to the subsequent segments until a new value is given. Value should be in the interval (0 , 134217728] and it is expressed mod(134217728)
Default:	134217728

−−**DCStop**	{int[int[int[...]]]}
−**dc**	*Mandatory:* No
Explanation:	Indicates whether compressed output stops after coding of quantized DC coefficients. If only one value is specified, type will be used for all image segments.
Default:	0

−−**bitPlaneStop**	{int[int[int[...]]]}
−**bp**	*Mandatory:* No
Explanation:	Only used when DCStop equals 0. Indicates the bit plane index where the codification should stop. If only one value is specified, type will be used for all image segments.
Default:	0

−−**stageStop**	{int[int[int[...]]]}
−**ss**	*Mandatory:* No
Explanation:	Indicates the stage where the codification ends. If only one value is specified, type will be used for all image segments. If three levels of DWT are applied, then: 1 - stage 1 2 - stage 2 3 - stage 3 4 - stage 4
Default:	The same of wavelet levels plus one

−−**useFill**	{int[int[int[...]]]}
−**uf**	*Mandatory:* No
Explanation:	Specifies whether fill bits will be used to produce SegByteLimit bytes in each segment. If only one value is specified, type will be used for all image segments.
Default:	0

−−**blocksPerSegment**	{int[int[int[...]]]}
−**bs**	*Mandatory:* No
Explanation:	Indicates the number of blocks contained in a segment
Default:	1048576

−−**optDCSelect**	{int[int[int[...]]]}
−**ds**	*Mandatory:* No
Explanation:	Indicates the method used to select k parameter while coding DC components
Default:	1

−−**optACSelect**	{int[int[int[...]]]}
−**as**	*Mandatory:* No
Explanation:	Indicates the method used to select k parameter while coding BitDepthAC_Block components
Default:	1

−−**WTType**	{int[int[int[...]]]}
−**wt**	*Mandatory:* No
Explanation:	Discrete wavelet transform type for each image component. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelete transform type will be the same for all components. Valid values are: 0 - No wavelet transform 1 - Integer (Reversible) 5/3 DWT 2 - Real Isorange (irreversible) 9/7 DWT (JPEG2000 standard) 3 - Real Isonorm (irreversible) 9/7 DWT (CCSDS-Recommended) 4 - Integer (Reversible) 9/7M DWT (CCSDS-Recommended) 5 - Integer 5/3 DWT (classic construction) 6 - Integer 9/7 DWT (classic construction)
Default:	4

−−**signedPixels**	{int[int[int[...]]]}
−**sp**	*Mandatory:* No
Explanation:	For each segment (entire image image in the Recommendation) specifies if the input pixels are signed or unsigned quantities. Valid values are: 0 - unsigned 1 - signed
Default:	0

−−**transposeImg**	{int[int[int[...]]]}
−**ti**	*Mandatory:* No
Explanation:	Indicates whether the channel (entire image in the Recommendation) should be transposed after reconstruction. Valid values are: 0 - do not transpose image 1 - transpose image
Default:	0

−−**codeWordLength**	{int[int[int[...]]]}
−**cl**	*Mandatory:* No
Explanation:	Indicated the coded word length for each segment. Valid values are: 0 - 8-bit word 1 - 16-bit word 2 - 24-bit word 3 - 32-bit word
Default:	0

−−**customWtFlag**	{int[int[int[...]]]}
−**wg**	*Mandatory:* No
Explanation:	Weighting type for each component. First value is for the first component, second for the second component and so on. If only one value is specified, type will be used for all image components. 0 - CCSDS recommended 1 - Defined by the user 2 - No weighting
Default:	0

−−**customWeight**	{float[float[float[...]]]}
−**cw**	*Mandatory:* No
Explanation:	Custom weights defined by the user. For each subband of each component a weighting factor should be given for each subband. If only the number of parameters introduced is not correct the programm stops. The order of the weights is as follows: For each component : LL_n,HL_n,LH_n,HH_n,HL_n-1,LH_n-1,HH_n-1,...,HL_0,LH_0,HH_0.
Default:	

−−**WTLevels**	{int[int[int[...]]]}
−**wl**	*Mandatory:* No
Explanation:	Discrete wavelet transform levels for each image component. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelet transform levels will be the same for all components.
Default:	3

−−**imageExtensionType**	{int[int[int[...]]]}
−**ie**	*Mandatory:* No
Explanation:	Indicates the kind of extension that has been applied to the each component of the image in order to make it able to be compressed using TER. If only one value is specified, image extension type will be the same for all components. Valid values are: 0 - Repeating last value 1 - Symmetric expansion 2 - No extension
Default:	0

−−**WTOrder**	{int[int[int[...]]]}
−**wo**	*Mandatory:* No
Explanation:	Order in which the Discrete wavelet transform is performed over the spatial dimentions. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelet transform order will be the same for all components. 0 - Horizontal - Vertical 1 - Vertical - Horizontal 2 - Only horizontal
Default:	0

−−**gaggleDCSize**	{int[int[int[...]]]}
−**gd**	*Mandatory:* No
Explanation:	Size of a gaggle for DC components in each segment of the image. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the size of DC gaggles will be the same for all segments. 0 value means that all the blocks of the segment are in the same gaggle
Default:	16

−−**gaggleACSize**	{int[int[int[...]]]}
−**ga**	*Mandatory:* No
Explanation:	Size of a gaggle for AC components in each segment of the image. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the size of AC gaggles will be the same for all segments. 0 value means that all the blocks of the segment are in the same gaggle
Default:	16

−−**idDC**	{int[int[int[...]]]}
−**id**	*Mandatory:* No
Explanation:	Number which indicates how often the ID DC appears. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the frequency of ID DC will be the same for all segments. 0 value means that only an ID DC is used for the segment
Default:	0

−−**idAC**	{int[int[int[...]]]}
−**ia**	*Mandatory:* No
Explanation:	Number which indicates how often the ID AC appears. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the frequency of ID AC will be the same for all segments. 0 value means that only an ID DC is used for the segment
Default:	0

−−**entropyAC**	{int[int[int[...]]]}
−**ea**	*Mandatory:* No
Explanation:	Integer array which indicates the entropy coder selected for coding AC components for each segment. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the AC entropy coder will be the same for all segments. If only one value is specified, entropy encoder for AC components will be the same for all segments. Valid values are: 0 - No entropy code for AC components 1 - CCSDS Recommended entropy coder
Default:	1

−−**resolutionLevels**	{int[int[int[...]]]}
−**rl**	*Mandatory:* No
Explanation:	Highest resolution level to be coded for each image component. 0 means that only LL subband will be compressed, 1 is LL + (HL1 + LH1 + HH1), 2 is LL + (HL1 + LH1 + HH1) + (HL2 + LH2 + HH2) and so on. Values must be between 0 to WT levels. If a value is greater than the number of WT levels it will be understood that all resolution levels of the component are to be compressed. If only one value is specified, it will be used for all image components.
Default:	same of WT levels specified

−−**pixelBitDepth**	{int[int[int[...]]]}
−**pd**	*Mandatory:* No
Explanation:	Integer array which indicates the pixel bit depth for each channel. First value is for the first channel, second value for the second channel and so on. If only one value is specified, the pixel bit depth will be the same for all channel.
Default:	

−−**verboseComputation**	{boolean}
−**vc**	*Mandatory:* No
Explanation:	Show some information about time and used memory for each compression stage. Value is a boolean: 0 indicates NO show and 1 indicates show.
Default:	0

−−**verboseParameters**	{boolean}
−**vp**	*Mandatory:* No
Explanation:	Show TER encoding parameters. Value is a boolean: 0 indicates NO show and 1 indicates show.
Default:	0

−−**bitsPerPixel**	{float[float[float[...]]]}
−**bpp**	*Mandatory:* No
Explanation:	Float array which indicates the bits per pixel of each encoded segment. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the bits per pixel will be the same for all segments.
Default:	

−−**compressionFactor**	{float[float[float[...]]]}
−**cf**	*Mandatory:* No
Explanation:	Float array which indicates the compression factor for each segment. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the compression factor will be the same for all segments.
Default:	1.0

−−**truncationPoints**	{int[int[int[...]]]}
−**tp**	*Mandatory:* No
Explanation:	For each segment specifies the available truncation points. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the truncation points will be the same for all segments. Valid values are: 0 - any 1 - end of bitplane (bitPlaneStop) 2 - end of resolution level (stageStop) 3 - end of gaggle
Default:	0

−−**adjustHeaderParameters**	{int[int[int[...]]]}
−**ap**	*Mandatory:* No
Explanation:	For each segment specifies the parameters that should be adjusted. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the asjusted parameters will be the same for all segments. Valid values are: 0 - No adjustement 1 - Adjust segByteLimit and useFill after interleaving
Default:	0

−−**compressionOrder**	{int[int[int[...]]]}
−**co**	*Mandatory:* No
Explanation:	This parameter specifies the way of creating each encoded segment. First value is for the first segment, second value for the second segment and so on. If only one value is specified, it will be the same for all segments. Valid values are: 0 - From the highest to the lowest bitplane 1 - From the fisrt to the last block
Default:	0

−−**shiftType**	{int}
−**st**	*Mandatory:* No
Explanation:	Level shift is a preprocessing operation for unsigned channels to convert them in signed channels. It can be performed in some diferent ways: 0- No level shift 1- JPEG2000 standard level shifting (only non-negative channels) 2- Range center substract 3- Average substract 4- Specific values substract (see ”-sv” parameter)
Default:	0

−−**shiftChannels**	{boolean[boolean[boolean[...]]]}
−**sc**	*Mandatory:* No
Explanation:	Specification in which channels level shift will be applied. Each value specifies a channel (0 is the first image channel).
Default:	true

−−**shiftValues**	{int[int[int[...]]]}
−**sv**	*Mandatory:* No
Explanation:	Substracted values in each image channel. If only one is specified, the value will be used for all image channel otherwise first value is for the first component, second value is for the second component, and so on.
Default:	null

−−**coefficientsApproximation**	{int[int[int[...]]]}
−**ca**	*Mandatory:* No
Explanation:	This parameter specifies the approximation to be applied to coefficients of each channel. First value is for the first channel, second value for the second channel and so on. If only one value is specified, it will be the same for all channels. Valid values are: 0 - Cast to integer 1 - Round 2 - Floor 3 - Ceil
Default:	0

3.2 TERdecoder parameters

−−**help**	
−**h**	*Mandatory:* No
Explanation:	Displays this help and exits program.
Default:	

−−**outputImage**	{string}
−**o**	*Mandatory:* Yes
Explanation:	Decoded image. Valid formats are: pgm, ppm, pbm, jpg, tiff, png, bmp, gif, fpx. If image is raw data file extension must be ".raw" and "-g" parameter is mandatory.
Default:	

−−**imageGeometry**	{int int int int boolean}
−**g**	*Mandatory:* No
Explanation:	Geometry of raw image data. Parameters are: 1- zSize (number of image components) 2- ySize (image height) 3- xSize (image width) 4- data type. Possible values are: 0- boolean (1 byte) 1- unsigned int (1 byte) 2- unsigned int (2 bytes) 3- signed int (2 bytes) 4- signed int (4 bytes) 5- signed int (8 bytes) 6- float (4 bytes) 7- double (8 bytes) 5- Byte order (0 if BIG ENDIAN, 1 if LITTLE ENDIAN) 6- 1 if 3 first components are RGB, 0 otherwise.
Default:	

−−**inputFile**	{string}
−**i**	*Mandatory:* Yes
Explanation:	Input encoded image file name.
Default:	

−−**segByteLimit**	{int[int[int[...]]]}
−**bl**	*Mandatory:* No
Explanation:	Maximum number of bytes that can be used in a segment. The value of SegByteLimits includes bytes used for the header, and is applied to the subsequent segments until a new value is given. Value should be in the interval (0 , 134217728]
Default:	134217728

−−**DCStop**	{int[int[int[...]]]}
−**dc**	*Mandatory:* No
Explanation:	Indicates whether compressed output stops after coding of quantized DC coefficients. If only one value is specified, type will be used for all image segments.
Default:	0

−−**bitPlaneStop**	{int[int[int[...]]]}
−**bp**	*Mandatory:* No
Explanation:	Only used when DCStop equals 0. Indicates the bit plane index where the codification should stop. If only one value is specified, type will be used for all image segments.
Default:	0

−−**stageStop** −**ss**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Indicates the stage where the codification ends. If only one value is specified, type will be used for all image segments. If three levels of DWT are applied, then: 1 - stage 1 2 - stage 2 3 - stage 3 4 - stage 4
Default:	The same of wavelet levels plus one

−−**useFill** −**uf**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Specifies whether fill bits will be used to produce SegByteLimit bytes in each segment. If only one value is specified, type will be used for all image segments.
Default:	0

−−**blocksPerSegment** −**bs**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Indicates the number of blocks contained in a segment
Default:	1048576

−−**WTType** −**wt**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Discrete wavelet transform type for each image component. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelete transform type will be the same for all components. Valid values are: 0 - No wavelet transform 1 - Integer (Reversible) 5/3 DWT 2 - Real Isorange (irreversible) 9/7 DWT (JPEG2000 standard) 3 - Real Isonorm (irreversible) 9/7 DWT (CCSDS-Recommended) 4 - Integer (Reversible) 9/7M DWT (CCSDS-Recommended) 5 - Integer 5/3 DWT (classic construction) 6 - Integer 9/7 DWT (classic construction)
Default:	4

−−**signedPixels** −**sp**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	For each segment (entire image image in the Recommendation) specifies if the input pixels are signed or unsigned quantities.
Default:	0

−−**transposeImg** −**ti**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Indicates whether the channel (entire image in the Recommendation) should be transposed after reconstruction
Default:	0

−−**codeWordLength** −**cl**	{int[int[int[...]]]} *Mandatory:* No
Explanation:	Indicated the coded word length for each segment. Valid values are: 0 - 8-bit word 1 - 16-bit word 2 - 24-bit word 3 - 32-bit word
Default:	0

−−**customWtFlag**	{int[int[int[...]]]}
−**wg**	*Mandatory:* No
Explanation:	Weighting type for each component. First value is for the first component, second for the second component and so on. If only one value is specified, type will be used for all image components. 0 - CCSDS recommended 1 - Defined by the user 2 - No weighting
Default:	0

−−**customWeight**	{float[float[float[...]]]}
−**cw**	*Mandatory:* No
Explanation:	Custom weights defined by the user. For each subband of each component a weighting factor should be given for each subband. If only the number of parameters introduced is not correct the programm stops. The order of the weights is as follows: For each component : LL_n,HL_n,LH_n,HH_n,HL_n-1,LH_n-1,HH_n-1,...,HL_0,LH_0,HH_0.
Default:	

−−**WTLevels**	{int[int[int[...]]]}
−**wl**	*Mandatory:* No
Explanation:	Discrete wavelet transform levels for each image component. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelet transform levels will be the same for all components.
Default:	3

−−**imageExtensionType**	{int[int[int[...]]]}
−**ie**	*Mandatory:* No
Explanation:	Indicates the kind of extension that has been applied to the each component of the image in order to make it able to be compressed using TER. If only one value is specified, image extension type will be the same for all components. Valid values are: 0 - Repeating last value 1 - Symmetric expansion 2 - No extension
Default:	0

−−**WTOrder**	{int[int[int[...]]]}
−**wo**	*Mandatory:* No
Explanation:	Order in which the Discrete wavelet transform is performed over the spatial dimentions. First value is for the first component, second value for the second component and so on. If only one value is specified, wavelet transform order will be the same for all components. 0 - Horizontal - Vertical 1 - Vertical - Horizontal 2 - Only horizontal
Default:	0

−−**gaggleDCSize**	{int[int[int[...]]]}
−**gd**	*Mandatory:* No
Explanation:	Size of a gaggle for DC components in each segment of the image. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the size of DC gaggles will be the same for all segments. 0 value means that all the blocks of the segment are in the same gaggle
Default:	16

−−**gaggleACSize**	{int[int[int[...]]]}
−**ga**	*Mandatory:* No
Explanation:	Size of a gaggle for AC components in each segment of the image. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the size of AC gaggles will be the same for all segments. 0 value means that all the blocks of the segment are in the same gaggle
Default:	16

−−**idDC**	{int[int[int[...]]]}
−**id**	*Mandatory:* No
Explanation:	Number which indicates how often the ID DC appears. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the frequency of ID DC will be the same for all segments. 0 value means that only an ID DC is used for the segment
Default:	0

−−**idAC**	{int[int[int[...]]]}
−**ia**	*Mandatory:* No
Explanation:	Number which indicates how often the ID AC appears. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the frequency of ID AC will be the same for all segments. 0 value means that only an ID DC is used for the segment
Default:	0

−−**entropyAC**	{int[int[int[...]]]}
−**ea**	*Mandatory:* No
Explanation:	Integer array which indicates the entropy coder selected for coding AC components for each segment. First value is for the first segment, second value for the second segment and so on. If only one value is specified, the AC entropy coder will be the same for all segments. Valid values are: 0 - No entropy code for AC components 1 - CCSDS Recommended entropy coder
Default:	1

−−**resolutionLevels**	{int[int[int[...]]]}
−**rl**	*Mandatory:* No
Explanation:	Highest resolution level to be coded for each image component. 0 means that only LL subband will be compressed, 1 is LL + (HL1 + LH1 + HH1), 2 is LL + (HL1 + LH1 + HH1) + (HL2 + LH2 + HH2) and so on. Values must be between 0 to WT levels. If a value is greater than the number of WT levels it will be understood that all resolution levels of the component are to be compressed. If only one value is specified, it will be used for all image components.
Default:	same of WT levels specified

−−**verboseComputation**	{boolean}
−**vc**	*Mandatory:* No
Explanation:	Show some information about time and used memory for each compression stage. Value is a boolean: 0 indicates NO show and 1 indicates show.
Default:	0

−−**verboseParameters**	{boolean}
−**vp**	*Mandatory:* No
Explanation:	Show TER encoding parameters. Value is a boolean: 0 indicates NO show and 1 indicates show.
Default:	0

−−**verboseMessages**	{boolean}
−**vm**	*Mandatory:* No
Explanation:	Show TER decoder messages, for instance if the file ends unexpectedly. Value is a boolean: 0 indicates NO show and 1 indicates show.
Default:	0

−−**padRows**	{int[int[int[...]]]}
−**pr**	*Mandatory:* No
Explanation:	Integer array that indicates for each channel the number of padding rows to be deleted after the inverse wavelet transform. If only one value is specified, padRows will be the same for all channels.
Default:	0

−−**gammaValue**	{float[float[float[...]]]}
−**gv**	*Mandatory:* No
Explanation:	Float array that indicates for each channel the way of recovering the values. First value is for the first channel, second value for the second channel and so on. If only one value is specified, it will be the same for all channels. This float indicates the point of the interval where values are recovered in the decoding process. When decoding a value inside an interval is possible to approximate this value to the middle value of the interval, to the low value, to 0,25 of the interval, to ... Example: in bit plane 3, a recovered value can be approximated to 8 (low value) or to 12 (middle value). Valid values are real numbers in the interval [0,1). For example: 0.0 - Lower value (gamma = 0) 0.5 - Middle value (gamma = 1/2) 0.375 - 3/8 of the threshold value (gamma = 3/8)
Default:	0.375

−−**completionMode**	{int[int[int[...]]]}
−**cm**	*Mandatory:* No
Explanation:	Integer array that indicates for each channel the way of completing DC values when the bitstream ends unexpectedly. First value is for the first channel, second value for the second channel and so on. If only one value is specified, it will be the same for all channels. Valid values are: 0 - Add zeros 1 - Repeat the middle value of the magnitude interval where the components are being decoded, i.e., (max(-xMin,xMax)+1)/2. 2 - Repeat the mean of the decoded values n - For n¿2, repeat the mean value of the n-2 last recovered values. If less than n-2 values have been recovered, only the recovered values will be used. Note that for n=3 we repeat the last value.
Default:	2

−−**shiftType**	{int}
−**st**	*Mandatory:* No
Explanation:	Level shift is a preprocessing operation for unsigned channels to convert them in signed channels. It can be performed in some diferent ways: 0- No level shift 1- JPEG2000 standard level shifting (only non-negative channels) 2- Range center substract 3- Average substract 4- Specific values substract (see ”-sv” parameter)
Default:	0

−−**shiftChannels**	{boolean[boolean[boolean[...]]]}
−**sc**	*Mandatory:* No
Explanation:	Specification in which channels level shift will be applied. Each value specifies a channel (0 is the first image channel).
Default:	true

−−**shiftValues**	{int[int[int[...]]]}
−**sv**	*Mandatory:* No
Explanation:	Substracted values in each image channel. If only one is specified, the value will be used for all image channel otherwise first value is for the first component, second value is for the second component, and so on.
Default:	null

4 Examples

LOSSLESS COMPRESSION USING A SINGLE SEGMENT

```
#TERcode -i workDir/mars512x32.raw -g 1 32 512 1 0 0 -o workDir/mars512x32.ter
```

This execution will generate a file (compilant with the Recommendation) called *mars512x32.ter*. The default mode of TER is lossless compression using a single segment and employing part 2,3 and 4 flag for the segment. To decompress the file and recover the original image run:

```
#TERdecode -i workDir/mars512x32.ter -o workDir/mars512x32.ter.raw -g 1 32 512 1 0 0
```

If the user wants to store the image in a known format, for example ’pgm’ format, then there is not needed the geometry of the output image. The user could run:

```
# TERdecode -i workDir/mars512x32.ter -o workDir/mars512x32.ter.pgm
```

LOSSLESS COMPRESSION USING MORE THAN ONE SEGMENT

```
#TERcode -i workDir/sar16bit.raw -g 1 512 512 2 0 0 -wt 4 –part2Flag 1 0 –part3Flag 1 0 –part4Flag 1 0
–blocksPerSegment 64 -o workDir/sar16bit.raw.ter
```

This execution will generate a file (compilant with the Recommendation) called *sar16bit.raw.ter*. The wavelet employed is the integer 9/7, each segment contains 64 blocks and only the first segment contains the part 2, 3 and 4 header. All other segments have only Part1 header. To decompress the file and recover the original image run:

```
#./TERdecode -i workDir/sar16bit.raw.ter -g 1 512 512 2 0 0 -o workDir/sar16bit.raw.ter.raw
```

LOSSY COMPRESSION

```
#TERcode -i workDir/foc.raw -g 1 512 1024 2 0 0 –pixelBitDepth 12 - –blocksPerSegment 128 wt 3 -o
workDir/foc.raw.ter
```

This exection uses the float 9/7 wavelet transform, then some loss is produce when coefficients are transformed. To decompress the file and recover the original image run:

```
#TERdecode -i workDir/foc.raw.ter -o workDir/foc.raw.ter.raw -g 1 512 1024 2 0 0
```

COMPRESSION USING A FIXED SEGBYTELIMIT

```
#./TERcode -i workDir/sar16bit.raw -g 1 512 512 2 0 0 -wt 4 –part2Flag 1 0 –part3Flag 1 0 –part4Flag 1 0
–blocksPerSegment 64 -o workDir/sar16bit.raw.ter –segByteLimit 3072 –useFill 1
```

This execution creates a file where each segment size is limited to 3072 bytes, in case the size of the segment in lower that 3072, it is completed with zeros up to the required size. To decompress the file and recover the original image run:

```
#./TERdecode -i workDir/sar16bit.raw.ter -g 1 512 512 2 0 0 -o workDir/sar16bit.raw.ter.raw
```

COMPRESSION USING CUSTOM WEIGHTS

```
#TERcode -i workDir/marstest.raw -g 1 512 512 1 0 0 -wt 4 –part2Flag 1 0 –part3Flag 1 0 –part4Flag 1 0
–blocksPerSegment 64 –pixelBitDepth 8 -o workDir/marstest.raw.ter -wg 1 -cw 4 4 4 2 2 2 1 1 1 1
```

This execution produces a encoded file with the scaling factor for each subband of the transformed image given by the user. To decompress the file and recover the original image run:

```
#TERdecode -i workDir/marstest.raw.ter -o workDir/marstest.raw.ter.raw -g 1 512 512 1 0 0
```

ONLY DC COMPONENTS COMPRESSION *#TERcode -i workDir/marstest.raw -g 1 512 512 1 0 0 -o*

```
workDir/temp.coded -wt 4 –blocksPerSegment 64 –part2Flag 1 0 –part3Flag 1 0 –part4Flag 1 0 -dc 1
```

This execution produces a encoded file where only DC components are encoded. To decompress the file and recover the original image run:

```
#TERdecode -i workDir/marstest.raw.ter -o workDir/marstest.raw.ter.raw -g 1 512 512 1 0 0
```

COMPRESSION UP TO A FIXED BITPLANE AND STAGE

```
#TERcode -i workDir/marstest.raw -g 1 512 512 1 0 0 -o workDir/marstest.raw.ter -wt 4 –blocksPerSegment 64
–part2Flag 1 0 –part3Flag 1 0 –part4Flag 1 0 -dc 0 -bp 1 -ss 2
```

This execution produces a encoded file that each segment is encoded up to the bit plane 1 and the stage 2. To decompress the file and recover the original image run:

```
#TERdecode -i workDir/marstest.raw.ter -g 1 512 512 1 0 0 -o workDir/marstest.raw.ter.raw
```

COMPRESSION USING A DIFFERENT ANY NUMBER OF WAVELET LEVELS

```
#TERcode -i workDir/lena.pgm -wl 5 -wt 3 -o workDir/lena.pgm.ter
```

This execution produces a file that is not compilant with the Recommendation. To decompress the file and recover the original image run:

```
#TERcode -i workDir/lena.pgm.ter -wl 5 -wt 3 -o workDir/lena.pgm.ter.pgm –recommendation no
```

SHOWING COMPRESSION STATISTICS

In some cases it is interesting to show the time that each stage lasts and the used memory. It can be done by:

```
#TERcode -i inputImage.pgm -o outputFile -vc 1
```