

# pnoJpegLib 2 Documentation

## Table of Contents

pnoJpegLib 2 Documentation.....	1
Decoding.....	2
pnoJpeg2Create.....	2
pnoJpeg2LoadFromPtr.....	2
pnoJpeg2LoadFromVFS.....	2
pnoJpeg2LoadFromHandle.....	3
pnoJpeg2LoadFromFileStream.....	3
pnoJpeg2GetInfo.....	3
pnoJpeg2SetScaleFactor.....	4
pnoJpeg2SetGrayscale.....	4
pnoJpeg2SetMaxDimensions.....	4
pnoJpeg2Read.....	5
pnoJpeg2Free.....	5
pnoJpeg2Version.....	5
Encoding.....	6
pnoJpeg2QuickEncode.....	6
pnoJpeg2EncodeCreate.....	6
pnoJpeg2EncodeSetQuality.....	7
pnoJpeg2EncodeProgressive.....	7
pnoJpeg2EncodeSetSmoothFactor.....	7
pnoJpeg2Write.....	7
pnoJpeg2EncodeFree.....	8
pnoJpeg2EncodeToMemHandle.....	8
pnoJpeg2EncodeToMemPtr.....	8
pnoJpeg2EncodeToFileStream.....	9
pnoJpeg2EncodeToVFS.....	9
Examples.....	10
Loading.....	10
QuickEncode.....	10
Normal Encoding .....	11

## Decoding

### **pnoJpeg2Create**

<b>Purpose</b>	Creates a pnoJpeg2-Data-Structure
<b>Prototype</b>	<b>Err pnoJpeg2Create (UInt16 refNum, pnoJpeg2Ptr *data)</b>
<b>Parameters</b>	<- data            Pointer to the new Data
<b>Result</b>	errNone memErrInvalidParam memErrNotEnoughSpace
<b>Comments</b>	You have to call pnoJpeg2Free after reading the image to free the data
<b>See also</b>	pnoJpeg2Free

### **pnoJpeg2LoadFromPtr**

<b>Purpose</b>	Load the JPEG-Data from a Pointer
<b>Prototype</b>	<b>Err pnoJpeg2LoadFromPtr (UInt16 refNum, pnoJpeg2Ptr data, MemPtr dataPtr, UInt32 dataSize)</b>
<b>Parameters</b>	-> data            Pointer to a pnoJpeg2-Structure -> dataPtr        Pointer to the JPEG-Data -> dataSize       size of the Data
<b>Result</b>	errNone memErrInvalidParam

### **pnoJpeg2LoadFromVFS**

<b>Purpose</b>	Load the JPEG-Data from a VFS-File
<b>Prototype</b>	<b>Err pnoJpeg2LoadFromVFS (UInt16 refNum, pnoJpeg2Ptr data, UInt16 volRefNum, char *filePath)</b>
<b>Parameters</b>	-> data            Pointer to a pnoJpeg2-Structure -> volRefNum       number of the volume -> filePath        path to the file
<b>Result</b>	errNone memErrInvalidParam

## **pnoJpeg2LoadFromHandle**

**Purpose** Load the JPEG-Data from a MemHandle

**Prototype** `Err pnoJpeg2LoadFromHandle(UInt16 refNum, pnoJpeg2Ptr data, MemHandle hImageData)`

**Parameters**

- > data Pointer to a pnoJpeg2-Structure
- > hImageData a Handle that contains the JPEG-Data

**Result**

- errNone
- memErrInvalidParam

## **pnoJpeg2LoadFromFileStream**

**Purpose** Load the JPEG-Data from a FileStream

**Prototype** `Err pnoJpeg2LoadFromFileStream(UInt16 refNum, pnoJpeg2Ptr data, FileHand fh)`

**Parameters**

- > data Pointer to a pnoJpeg2-Structure
- > fh a filehandle that contains the JPEG-Data

**Result**

- errNone
- memErrInvalidParam

**Comments** Thanks to Jean-Pierre Morfin for this function

## **pnoJpeg2GetInfo**

**Purpose** Read the Image Dimensions

**Prototype** `Err pnoJpeg2GetInfo(UInt16 refNum, pnoJpeg2Ptr data, Coord *width, Coord *height)`

**Parameters**

- > data Pointer to a pnoJpeg2-Structure
- <- width width of the image
- <- height height of the image

**Result**

- errNone
- memErrInvalidParam

**Comments** after getting the info about the Dimensions it's possible to change some loading settings, such as maxWidth, gayscale or scalefactor

## **pnoJpeg2SetScaleFactor**

<b>Purpose</b>	Sets the Scalefactor for the image loading
<b>Prototype</b>	<b>Err pnoJpeg2SetScaleFactor(UInt16 refNum, pnoJpeg2Ptr data, UInt16 factor)</b>
<b>Parameters</b>	-> data                    Pointer to a pnoJpeg2-Structure -> factor                   down-scale-factor
<b>Result</b>	errNone memErrInvalidParam
<b>Comments</b>	possible factors: 1, 2, 4, 8

## **pnoJpeg2SetGrayscale**

<b>Purpose</b>	Sets the Scalefactor for the image loading
<b>Prototype</b>	<b>Err pnoJpeg2SetGrayscale(UInt16 refNum, pnoJpeg2Ptr data, Boolean grayscale)</b>
<b>Parameters</b>	-> data                    Pointer to a pnoJpeg2-Structure -> grayscale               true: load grayscale image; false: load color Image
<b>Result</b>	errNone memErrInvalidParam

## **pnoJpeg2SetMaxDimensions**

<b>Purpose</b>	Set the maximum Dimensions for the image loading
<b>Prototype</b>	<b>Err pnoJpeg2SetMaxDimensions(UInt16 refNum, pnoJpeg2Ptr data, Coord maxWidth, Coord maxHeight)</b>
<b>Parameters</b>	-> data                    Pointer to a pnoJpeg2-Structure -> maxWidth ->maxHeight
<b>Result</b>	errNone memErrInvalidParam
<b>Comments</b>	Thanks to Jean-Pierre Morfin for this function

## **pnoJpeg2Read**

**Purpose** Reads the data and decode it to the Bitmap

**Prototype** `Err pnoJpeg2Read(UInt16 refNum, pnoJpeg2Ptr data, BitmapPtr *bmpPtr)`

**Parameters**

-> data	Pointer to a pnoJpeg2-Structure
<- bmpPtr	Contains the loaded Image as Bitmap

**Result**

errNone  
memErrInvalidParam

## **pnoJpeg2Free**

**Purpose** Frees the pnoJpeg2 data structure

**Prototype** `Err pnoJpeg2Free(UInt16 refNum, pnoJpeg2Ptr *data)`

**Parameters**

-> data	Pointer to the Pointer of a pnoJpeg2-Structure
---------	--

**Result**

errNone  
memErrInvalidParam

## **pnoJpeg2Version**

**Purpose** Returns the Version of the installed lib

**Prototype** `UInt16 pnoJpeg2Version(UInt16 refNum)`

**Parameters**

none

**Result** Returns the Version of the installed Lib

## Encoding

### **pnoJpeg2QuickEncode**

<b>Purpose</b>	Encode the given 16-Bit-Bitmap into the given MemHandle with the given quality
<b>Prototype</b>	<b>Err pnoJpeg2QuickEncode(UInt16 refNum, BitmapPtr sourceBmp, MemHandle *dest, UInt32 quality)</b>
<b>Parameters</b>	SourceBmp -> input Bitmap *dest -> Pointer to a Memhandle quality -> Encode Quality (1-100)
<b>Result</b>	errNone memErrInvalidParam memErrNotEnoughSpace pnoJpegErrInvalidBitDepth
<b>Comments</b>	The MemHandle will be allocated from the Library, the Application have to free it after using it.

### **pnoJpeg2EncodeCreate**

<b>Purpose</b>	Creates a pnoJpeg2Enc-Data-Structure
<b>Prototype</b>	<b>Err pnoJpeg2EncodeCreate(UInt16 refNum, pnoJpeg2EncPtr *data)</b>
<b>Parameters</b>	<- data          Pointer to the new Data
<b>Result</b>	errNone memErrInvalidParam memErrNotEnoughSpace
<b>Comments</b>	You have to call pnoJpeg2EncodeFree after writing the image to free the data.
<b>See also</b>	pnoJpeg2EncodeFree

## **pnoJpeg2EncodeSetQuality**

**Purpose** Sets the encoding Quality

**Prototype** `Err pnoJpeg2EncodeSetQuality(UInt16 refNum, pnoJpeg2EncPtr data, UInt32 quality)`

**Parameters**  
->data Pointer to a pnoJpeg2Enc-Data-Structure  
-> quality a Value between 0 - 100

**Result**  
errNone  
memErrInvalidParam

## **pnoJpeg2EncodeProgressive**

**Purpose** If set, the Result-Image will be a progressive JPEG

**Prototype** `Err pnoJpeg2EncodeProgressive(UInt16 refNum, pnoJpeg2EncPtr data, Boolean progressive)`

**Parameters**  
->data Pointer to a pnoJpeg2Enc-Data-Structure  
->progressive true for progressive Encoding

**Result**  
errNone  
memErrInvalidParam

## **pnoJpeg2EncodeSetSmoothFactor**

**Purpose** Sets the Smooth-Factor for image encoding

**Prototype** `Err pnoJpeg2EncodeSetSmoothFactor(UInt16 refNum, pnoJpeg2EncPtr data, UInt32 smoothfactor)`

**Parameters**  
->data Pointer to a pnoJpeg2Enc-Data-Structure  
->smoothfactor Value from 0-100 (0 is default)

**Result**  
errNone  
memErrInvalidParam

## **pnoJpeg2Write**

**Purpose** Writes the JPEG-Data with the given settings to the given destination

**Prototype** `Err pnoJpeg2Write(UInt16 refNum, pnoJpeg2EncPtr data, BitmapPtr source)`

## **pnoJpeg2Write**

**Parameters**      ->data          Pointer to a pnoJpeg2Enc-Data-Structure  
                     -> source        a 16-Bit-Bitmap

**Result**            errNone  
                     memErrInvalidParam  
                     memErrNotEnoughSpace  
                     pnoJpegErrInvalidBitDepth

## **pnoJpeg2EncodeFree**

**Purpose**            Frees the Data-Structure

**Prototype**        **Err pnoJpeg2EncodeFree (UInt16 refNum, pnoJpeg2EncPtr \*data)**

**Parameters**      ->data          Pointer to a pnoJpeg2Enc-Data-Structure Pointer

**Result**            errNone  
                     memErrInvalidParam

## **pnoJpeg2EncodeToMemHandle**

**Purpose**            Sets the Destination to a MemHandle

**Prototype**        **Err pnoJpeg2EncodeToMemHandle (UInt16 refNum, pnoJpeg2EncPtr data, MemHandle \*dest)**

**Parameters**      ->data          Pointer to a pnoJpeg2Enc-Data-Structure  
                     ->dest          Pointer to a MemHandle

**Result**            errNone  
                     memErrInvalidParam

**Comments**        The MemHandle will be allocated from the Library, the Application have to free it after using it.

## **pnoJpeg2EncodeToMemPtr**

**Purpose**            Sets the Destination to a MemPtr

**Prototype**        **Err pnoJpeg2EncodeToMemPtr (UInt16 refNum, pnoJpeg2EncPtr data, MemPtr \*dest)**

## **pnoJpeg2EncodeToMemPtr**

<b>Parameters</b>	->data            Pointer to a pnoJpeg2Enc-Data-Structure ->dest            Pointer to a MemPtr
<b>Result</b>	errNone memErrInvalidParam
<b>Comments</b>	The Memory will be allocated by the Library and have to be freed by the Application

## **pnoJpeg2EncodeToFileStream**

<b>Purpose</b>	Sets the Destination to a FileStream
<b>Prototype</b>	<b>Err pnoJpeg2EncodeToFileStream(UInt16 refNum, pnoJpeg2EncPtr data, FileHand fh)</b>
<b>Parameters</b>	->data            Pointer to a pnoJpeg2Enc-Data-Structure ->fh                Handle to an open FileStream
<b>Result</b>	errNone memErrInvalidParam

## **pnoJpeg2EncodeToVFS**

<b>Purpose</b>	Sets the Destination to a VFS-File
<b>Prototype</b>	<b>Err pnoJpeg2EncodeToVFS(UInt16 refNum, pnoJpeg2EncPtr data, UInt16 volRefNum, char *filePath)</b>
<b>Parameters</b>	->data            Pointer to a pnoJpeg2Enc-Data-Structure ->volRefNum        Number of the Volume ->filePath         Path and Name of the Destination
<b>Result</b>	errNone memErrInvalidParam
<b>Comments</b>	The Library will create a new File or overwrite an existing.

## Examples

### Loading

```
{
    pnoJpeg2Ptrjpeg = NULL;
    Err err = errNone
    BitmapPtrbmp = NULL;

    err = pnoJpeg2Create(pnoJpegRefNum, &jpegData);
    if(err != errNone) return;

    err = pnoJpeg2LoadFromVFS(pnoJpegRefNum, jpegData, volRefNum, "/jpegFile.jpg");
    if(err != errNone)
    {
        pnoJpeg2Free(pnoJpegRefNum, &jpegData);
    }

    err = pnoJpeg2Read(pnoJpegRefNum, jpegData, &bmp);
    if(err == memErrNotEnoughSpace)
    {
        // Image seems to be too big, load it smaller
        pnoJpeg2SetScaleFactor(pnoJpegRefNum, jpegData, 2);
        err = pnoJpeg2Read(pnoJpegRefNum, jpegData, &bmp);
    }

    pnoJpeg2Free(pnoJpegRefNum, &jpegData);
    if(bmp)
    {
        // Display Bitmap and Delete it
    }
}
```

### QuickEncode

With pnoJpeg2QuickEncode is the easiest way to encode a BitmapPtr to JPEG

```
Err pnoJpeg2QuickEncode(UInt16 refNum, BitmapPtr sourceBmp, MemHandle *dest, UInt32 quality)
{
    MemHandle dest;
    BitmapPtr bmp; // We assume the bmp is a Pointer to a 16-Bit-Bitmap
    Err err = errNone;

    err = pnoJpeg2QuickEncode(pnoJpeg2RefNum, bmp, &dest, 50);
    // Now do what you want with the JPEG-Data
    MemHandleFree(dest);
}
```

## Normal Encoding

```
{
pnoJpeg2EncPtr  jpegEncode;
BitmapPtr      bmp; // We assume the bmp is a Pointer to a 16-Bit-Bitmap
Memhandle      hJpeg;
Err            err;

err = pnoJpeg2EncodeCreate(pnoJpegRefNum, &jpegEncode);
if(err != errNone) return;
pnoJpeg2EncodeToMemHandle(pnoJpegRefNum, jpegEncode, &hJpeg);
pnoJpeg2EncodeSetQuality(pnoJpegRefNum, jpegEncode, 50);
pnoJpeg2EncodeProgressive(pnoJpegRefNum, jpegEncode, true);
err = pnoJpeg2Write(pnoJpegRefNum, jpegEncode, bmp);
pnoJpeg2EncodeFree(pnoJpegRefNum, &jpegEncode);

if(err == errNone)
{
// Now do what you want with the JPEG-Data
MemHandleFree(hJpeg);
}
}
```