



## **Atalasoft DotImage JPEG2000 4.0 Developer's Guide**

Atalasoft, Inc.  
116 Pleasant St, Suite 321  
Easthampton, MA 01027  
413-572-4443  
866-568-0129

**[www.atalasoft.com](http://www.atalasoft.com/) (<http://www.atalasoft.com/>)**  
**[info@atalasoft.com](mailto:info@atalasoft.com)**

## Table Of Contents

---

1.	Introduction	3
2.	Overview	4
3.	Getting Started	5
4.	Deploying DotImage Jpeg2000	6
5.	Programming with DotImage JPEG2000	7
5.1.	Encoding Jpeg2000 Images	7-8
5.2.	Tile and Component Encoding Options	8-9
5.3.	Decoding Jpeg2000 Images	9-10
5.4.	Metadata	10-11
5.5.	Metadata Examples	11-12
6.	Object Reference	13

## 1 Introduction

---

Atalasoft DotImage JPEG2000 image compression offers the full range of features and functionality provided by the JPEG2000 wavelet compression: excellent image quality at high compression rates, fast compression and decompression, exact control of compression rate, progressive decompression, without JPEG-like blocking artifacts and selection of the reconstruction rate.

This documentation covers DotImage JPEG2000 Standard and Professional Editions. Please see the **Atalasoft.dotImage.Jpeg2000 (Atalasoft.dotImage.Jpeg2000.html)** assembly in the Object Reference located in the HTML Help for navigating the object structure and object documentation.

## 2 Overview

---

The DotImage JPEG2000 Codec is based on the Lurawave.jp2 codec. It requires the managed assembly, *Atalasoft.dotImage.Jpeg2000.dll* installed along side your .NET application.

**DotImage JPEG2000 Codec Standard Edition** includes the following features:

- Decodes all JPEG2000 images. Resulting colorspace are 8-bit grayscale, 16-bit grayscale, or 24-bit RGB.
- Encodes to the JP2 format as 8-bit grayscale or 24-bit RGB images.
- Supports adjusting compression level.
- Supports Lossless JPEG2000 compression.
- Industry leading performance.
- An easy to use managed .NET object model, fully integrating with DotImage.
- Thread-safe with easy to use multithreaded decoding and encoding when using the DotImage Workspace.
- Supports low level codec properties such as byte order, precision, quality style, speed mode, code block size, packet markers, progression order, quality layers, quantization style, wavelet filter methods, and wavelet levels. Many of these properties can be specified independently for the entire image, component, or tile.

**DotImage JPEG2000 Codec Professional Edition** includes the following features:

- Supports decoding JPEG2000 images into 8-bit grayscale, 16-bit grayscale, 24-bit RGB, 32-bit RGB/RGBA, 32-bit CMYK. 12-bit gray is converted to 16-bit grayscale.
- Supports encoding 8-bit gray, 12-bit gray, 16-bit gray, 24-bit RGB, 32-bit RGB/RGBA, 32-bit CMYK, 48-bit RGB, and 64-bit RGB/RGBA to JP2, JPX, or JPEG2000 codestream.
- Supports progressive decoding.
- Supports metadata such as UUID, UUID Info Box, XML, Intellectual Property Rights Data, and ICC Color Profiles.
- Supports Enhanced Region encoding, also known as Region of Interest Encoding, allowing up to 16 rectangular areas in an image to be encoded at higher quality.

## 3 Getting Started

---

### 3.1 Registering the JPEG2000 codec in DotImage

To setup DotImage to decode JP2 images, add an instance of the **Jp2Decoder** to the **Atalasoft.Imaging.codec.RegisteredDecoders.Decoders** collection.

```
[C#]
using Atalasoft.Imaging.codec.Jpeg2000;
...
Jp2Decoder jp2 = new Jp2Decoder();
RegisteredDecoders.Decoders.Add(jp2);
[Visual Basic]
Imports Atalasoft.Imaging.codec.Jpeg2000
...
Dim jp2 As Jp2Decoder = New Jp2Decoder()
RegisteredDecoders.Decoders.Add(jp2)
```

### 3.2 Linking to the License File

To compile DotImage JPEG2000 in a Windows Forms application such that the royalty free license is installed into the application resource, a file called licenses.licx must be added to the project, with the following line in that file:

```
Atalasoft.Imaging.Codec.Jpeg2000.Jp2Decoder, Atalasoft.dotImage.Jpeg2000
```

This tells the Visual Studio .NET compiler to link the license file to the resources in the EXE.

### 3.3 Decoding JPEG2000 Images

Once the decoder is registered, all methods that decode an image, such as **Workspace.Open** or new **AtalaImage(filename)** will recognize JP2 images as valid supported images. The **Read** method of **Jp2Decoder** may also be called directly to bypass the image format check determining the codec to use.

See **Decoding JPEG2000 Images (Section 5.3)** for more information.

### 3.4 Getting Information from JPEG2000 Images

It is possible to retrieve information from a JPEG2000 image without decoding it by using the **GetImageInfo** method of the **Jp2Decoder** class, or the **GetImageInfo** method of the **RegisteredDecoders** class. Information, including width, height, bitdepth, is available. To access Intellectual Property Rights data and the type of code stream, cast the returned **ImageInfo** class to a **Jp2ImageInfo**. Follow the example below:

```
[C#]
using Atalasoft.Imaging.Codec.Jpeg2000;
...
Jp2ImageInfo info = (Jp2ImageInfo)RegisteredDecoders.GetImageInfo("myimage.jp2");
Console.WriteLine(info.FileFormat);
[Visual Basic]
Imports Atalasoft.Imaging.Codec.Jpeg2000
...
Dim info As Jp2ImageInfo = CType(RegisteredDecoders.GetImageInfo("myimage.jp2"), _
    Jp2ImageInfo)
Console.WriteLine(info.FileFormat)
```

### 3.5 Encoding JPEG2000 Images

To encode JPEG2000 images, create an instance of the **Jp2Encoder** class, and pass it into the **Workspace.Save** method, the **AtalaImage.Save** method, or the **Save** method in the **Jp2Encoder** class. The **Compression** property in the **Jp2Encoder** can be set to compress the resulting image to the desired amount. For example, setting the compression to 5 will result in an image that is approximately 5% the size of the original uncompressed image.

See **Encoding JPEG2000 Images (Section 5.1)** for more information.

## 4 Deploying DotImage Jpeg2000

---

When deploying DotImage JPEG2000 to a client machine or server, the following must be copied to the same folder as the EXE which references the Atalasoft assemblies:

- *Atalasoft.dotImage.dll*
- *Atalasoft.dotImage.Jpeg2000.dll*
- *Atalasoft.dotImage.Lib.dll*

When distributing client desktop applications, the license file is embedded into the resource, and there is no need to distribute or activate any additional licenses. See **Getting Started (Section 3)** for more information on linking the license file to the resource.

When installing on a production server, a server license must be acquired and activated for the server.

## 5 Programming with DotImage JPEG2000

---

### 5.1 Encoding Jpeg2000 Images

---

As indicated in **Getting Started (Section 3)**, encoding JP2 images involves creating an instance of **Jp2Encoder**, which derives from **ImageEncoder**, and then invoking the **Save** method. The Standard edition, allows adjustment of just one property, the **Compression** property. The Professional edition grants low level access to the Codec. This topic covers all encoder settings.

#### 5.1.1 Encoding Tiled Images

Among the important features of JPEG2000 is capability of encoding a large image - up to  $(2^{32} - 1) \times (2^{32} - 1)$  pixels - without breaking into tiles. JPEG2000 also has a facility of compressing an image into tiles in which data may be compressed independently in each tile or in each color component. Encoding with (or without) tiles using DotImage Jpeg2000 are explained below.

##### To encode without tiles:

Set the **TileSize** property to of **Jp2Encoder** to (0,0) or *Size.Empty* which implies no tiles.

##### To encode with tiles:

Set the **TileSize** property of **Jp2Encoder** to any non-zero size.

#### 5.1.2 Encoding Region of Interest

The JPEG2000 specification includes the ability to encode user specific areas of the image at higher quality. This gives more detail to certain areas of an image, without compromising file size. The **Jp2Encoder** has an **EnhancedRegions** property, which points to a collection of **Jp2EnhancedRegion** objects. Setting the **EnhancementFactor** in the **Jp2EnhancedRegions** collection boosts the quality of all regions by a factor over the *Compression* property.

#### 5.1.3 Baseline Encoder Properties

These properties can be set for the entire image only (cannot be independently set to tiles or components).

- **Compression** - Gets or sets the compressed size of the image as a percentage of an uncompressed image.
- **EnhancedRegions** - Gets a collection of Enhanced Regions that can be set at a higher quality level than the rest of the image during compression.
- **FileFormat** - Gets or sets the file format to generate (JP2, JPEG2000 codestream, or JPX).
- **IPData** - Gets or sets intellectual property rights data to be encoded with the image.
- **IptcTags** - Gets or sets IPTC data that will be stored in the encoded image.
- **Precision** - Gets or sets a value indicating the precision of the wavelet coefficients.
- **QualityStyle** - Gets or sets a value indicating the quality mode during lossy compression.
- **SpeedMode** - Gets or sets the speed mode (Fast or Accurate) to use during lossy compression.
- **TileSize** - Gets or sets the size in pixels of each individual tile.
- **UuidBoxes** - Gets or sets UuidBox metadata to be stored in the encoded image.
- **UuidInfoBoxes** - Gets or sets UuidInfoBox metadata to be stored in the encoded image.
- **WriteTileLengthMarker** - Gets or sets a value indicating if a tile length marker is written to the encoded image.
- **XmlBoxes** - Gets or sets XML metadata to be stored with the image.

#### 5.1.4 Per Tile Encoder Properties

These properties can be set for the entire image or independently for each tile. These properties exist in **EncoderOptions**, and by default will be applied to the entire image unless overriding **GetEncoderOptions**. See **Tile and Component Encoding Options (Section 5.2)** for more information on this technique.

- **QualityLayers** - Gets or sets the number of quality layers in the code stream for use with progressive

decoding.

- **ProgressionOrder** - Gets or sets the organization of the coded data.
- **PacketMarkers** - Gets or sets a value that creates special markers at the beginning and/or at the end of each block of a coded area.

### 5.1.5 Per Tile and Per Component Properties

These properties can be set for the entire image, independently for each tile, and independently for each component. These properties exist in **EncoderOptions**, and by default will be applied to the entire image unless overriding **GetEncoderOptions**. See **Tile and Component Encoding Options (Section 5.2)** for more information on this technique.

- **WaveletFilterMethod** - Gets or sets a value selecting reversible (*WaveletFiveThree*) or irreversible (*WaveletNineFive*) wavelet filters.
- **WaveletLevels** - Gets or sets the number of wavelet transformation levels.
- **QuantizationStyle** - Gets or sets the quantization steps.
- **CodeBlockSize** - Gets or sets the size of the blocks of coded data.
- **CoderOptions** - Gets or sets the coder options for faster compression / decompression.

## 5.2 Tile and Component Encoding Options

---

JPEG2000 streams can have encoding options that are independent for each component in each tile. DotImage Jpeg2000 provides the ability to encode image with this advanced feature.

In order to accomplish this advanced encoding behavior, derive from **Jp2Encoder** and override the protected **GetEncoderOptions** method. The *GetEncoderOptions* method returns an **Jp2EncoderOptions** class that is applied to the component, or tile of interest. The default implementation of *GetEncoderOptions* returns the an **EncoderOptions** property which is options applied to the entire image.

Two examples are provided below: one that provides separate encoding options depending on the channel (or component) and one that provides separate encoding options depending on the location of the tile.

To provide a set of options for only a few tiles in the entire set, override *GetEncoderOptions* and return the options for the specific tiles you are concerned with, otherwise return *base.GetEncoderOptions* (see below).

### 5.2.1 Per-Channel Encoding Options Example

This example demonstrates changing the Encoding options for each of the three red, green, and blue color components.

```
[C#]
public class SeparateChannelOptionsJp2Encoder : Jp2Encoder
{
    private Jp2EncoderOptions _redOpts;
    private Jp2EncoderOptions _greenOpts;
    private Jp2EncoderOptions _blueOpts;
    public SeparateChannelOptionsJp2Encoder(int nChannels) : base(nChannels)
    {
    }
    public Jp2EncoderOptions RedOptions
    {
        set { _redOpts = value; }
        get { return _redOpts; }
    }
    public Jp2EncoderOptions GreenOptions
    {
        set { _greenOpts = value; }
        get { return _greenOpts; }
    }
    public Jp2EncoderOptions BlueOptions
    {
        set { _blueOpts = value; }
        get { return _blueOpts; }
    }
}
```

```

protected override Jp2EncoderOptions GetEncoderOptions(Point tileLocation,
    ChannelFlags channel)
{
    switch (channel)
    {
        case ChannelFlags.Channel1:
            return RedOptions;
        case ChannelFlags.Channel2:
            return BlueOptions;
        case ChannelFlags.Channel3:
            return GreenOptions;
        default:
            return base.GetTileEncoderOptions(tileLocation, channel);
    }
}
}

```

### 5.2.2 Per-Tile Encoding Options Example

This example changes the encoding options based on a checkerboard pattern through the tiles. This is not a real world example per se, but demonstrates how to set individual options based on the position of the tiles. "black" tiles use a special set of encoding options and "white" tiles use defaults.

```

[C#]
public class CheckerOptionsJp2Encoder : Jp2Encoder
{
    private Jp2EncoderOptions _blackOptions;
    public Jp2EncoderOptions BlackOptions
    {
        set { _blackOptions = value; }
        get { return _blackOptions; }
    }
    public CheckerOptionsJp2Encoder (int nChannels) : base(nChannels)
    {
    }
    protected override Jp2EncoderOptions GetEncoderOptions(Point tileLocation,
        ColorChannel channel)
    {
        if (((tileLocation.X + tileLocation.Y) & 1) == 0)
        {
            return BlackOptions;
        }
        else
        {
            return base.GetEncoderOptions(tileLocation, channel);
        }
    }
}

```

## 5.3 Decoding Jpeg2000 Images

---

As indicated in **Getting Started (Section 3)**, once the decoder is registered, all methods that decode an image, such as **Workspace.Open** or new **AtalaImage(filename)** will recognize JP2 images as valid supported images. The Professional edition supports progressive image decoding and adjusting low level properties as outlined below.

### 5.3.1 Progressive Decoding

DotImage JPEG2000 Professional supports progressively decoding JP2 images. This is especially useful when reading JP2 images over the internet and other low bandwidth situations. The codec will fire the **ProgressiveImage** event, which can be used to incrementally display the entire image as it's being loaded and decoded. To enable progressive decoding, set the **EnableProgressiveDecompression** property of the **Jp2Decoder** to true, and set the **ProgressiveDecodeSteps** to the number of steps you wish to render.

### 5.3.2 Decoder Settings

The following properties are available in the **Jp2Decoder** to adjust how the decoder functions.

- **ByteOrder** - Gets or sets the byte order of the decoded data from one of little endian, big endian, or machine automatic.
- **CacheOption** - Gets or sets a value specifying whether the compress data should be stored in an internal cache.
- **EnableProgressiveDecompression** - Gets or sets a value enabling the ability to progressively decode the image.
- **Precision** - Gets or sets a value indicating the precision of the wavelet coefficients.
- **ProgressiveDecodeSteps** - Gets or sets the number of progressive decompression steps.
- **ScaleDown** - Gets or sets a factor that is used to scale downwards the image as it's decoded.

## 5.4 Metadata

---

The JPEG2000 file format supports various types of metadata, in particular intellectual property rights information, XML data, IPTC data, UUID Box data, and UUID info data. DotImage JPEG2000 Professional supports reading and writing each of these with a simple interface.

Note that metadata can only be stored into the Jp2 and Jpx file formats. Metadata may not be stored in a plain code stream.

Metadata in JPEG2000 images is covered in detail by the document ISO/IEC CD15444-1, *Information technology - JPEG2000 image coding system*

### 5.4.1 IP Data

This metadata can be obtained by calling **GetImageInfo** of the **Jp2Decoder** class. It is a simple string which may or may not be set to anything. It represents Intellectual Property Rights information.

To store this data, set the **IPData** property of the **Jp2Encoder** to the string value.

### 5.4.2 XML Box

A JPEG2000 image can contain arbitrary XML information. Multiple XML "Boxes" can be stored, where each Box contains a separate XML document. In the DotImage JPEG2000 interface, each XML box is defined by a string. See the **XmlBoxParser** for obtaining the XML boxes from a JPEG2000 image. To store XML in a JPEG image, create an array of XML Box strings, and set it to the **XmlBoxes** property in the **Jp2Encoder** class.

### 5.4.3 IPTC

IPTC information can be stored within a JPEG2000 image inside a UUID Box. IPTC information can be obtained using the **Jp2IptcParser**, which derives from the **IptcParser** within the *Atalasoft.Imaging.Metadata* namespace. See the **IPTC (on-line documentation)** topic for more information about IPTC. IPTC can be stored in a JPEG2000 image by setting the **IptcTags** property in the **Jp2Encoder** class.

### 5.4.4 UUID Box

A UUID box contains vendor specific data other than that data defined within the ISO specification. There may be zero or more UUID boxes with in the file. No UUID box shall contain information necessary for decoding the image to the extent that is defined within international standard, nor will the interpretation of the data in any UUID change the visual appearance of the image. Each UUID consists of a Uuid and Data property. The Uuid is a 128-bit integer defined by an **AtalaInt128** structure and the Data is defined by a byte array.

To obtain all UUID Boxes in a JP2 image, use the **UuidBoxParser** class, which returns a collection of **UuidBox** objects.

To store UUID boxes in a JP2 image, set the **UuidBoxes** property in the **Jp2Encoder** class.

### 5.4.5 UUID Info Box

While it is useful to allow vendors to extend JP2 files by adding binary data using UUID boxes, it is also useful to provide information in a standard form which can be used by non-extended applications to get more information about the extensions in the file. A JP2 file may contain zero or more UUID Info boxes. Each UUID Info Box contains a list of UUID's as defined in the **ListBox** property, and a Url associated with the UUID Info Box as defined in the **UrlBox** property.

To obtain all UUID Info Boxes in a JP2 image, use the **UuidInfoBoxParser** class, which returns a collection of **UuidInfoBox** objects.

To store UUID Info Boxes in a JP2 image, set the **UuidInfoBoxes** property in the **Jp2Encoder** class.

## 5.5 Metadata Examples

---

These examples demonstrate how to encode and retrieve metadata to and from JPEG2000 images.

### 5.5.1 Storing Metadata

```
[C#]
Jp2Encoder je = new Jp2Encoder();
// IPR data
string iprData = "Copyright(c) 2005, Atalasoft Inc.";
// set IPR data to encoder
je.IPRData = iprData;
// XML data
string[] xmlData = new string[1];
xmlData[0] = "<?xml version='1.0' encoding='utf-8'?><data></data>";
// set XML data collection to encoder
je.XmlBoxes = xmlData;
// IPTC data
IptcCollection iptcCollection = new IptcCollection();
string iptcData = "sample";
IptcTag iptcTag = new IptcTag(2, 5, 0, iptcData);
// set IPTC data collection to encoder
je.IptcTags = iptcCollection;
// UUID box data
UuidBoxCollection uuidBoxCollection = new UuidBoxCollection();
UuidBox uuidBox = new UuidBox();
// set UUID box data collection to encoder
je.UuidBoxes = uuidBoxCollection;
// UUID info data
UuidInfoBoxCollection uuidInfoBoxCollection = new UuidInfoBoxCollection();
UuidInfoBox uuidInfoBox = new UuidInfoBox();
// set UUID info box data collection to encoder
je.UuidInfoBoxes = uuidInfoBoxCollection;
//save the image with metadata
je.Save(stream, null);
```

### 5.5.2 Retrieving Metadata

```
[C#]
// get an instance of Jp2ImageInfo from Jp2Decoder and Stream
Jp2Decoder jp2 = new Jp2Decoder();
Jp2ImageInfo jp2ImageInfo = (Jp2ImageInfo)jp2.GetImageInfo(fs);
// get IpData
string ipData = jp2ImageInfo.IPRData;
// create an instance of Jp2IptcParser
Jp2IptcParser iptcParser = new Jp2IptcParser();
// get an instance of IPTC Collection
IptcCollection iptcCollection = iptcParser.ParseFromImage(fs);
// create an instance of XmlParser
Jp2XmlParser xmlParser = new XmlParser();
// get an instance of XmlBoxCollection
string[] xmlBoxes = xmlParser.ParseFromImage(fs);
// create an instance of UuidBoxParser
UuidBoxParser uuidBoxParser = new UuidBoxParser();
```

```
// get an instance of UuidBoxCollection
UuidBoxCollection uuidBoxCollection = uuidBoxParser.ParseFromImage(fs);
// create an instance of UuidInfoBoxParser
UuidInfoBoxParser uuidInfoBoxParser = new UuidInfoBoxParser();
// get an instance of UuidBoxCollection
UuidInfoBoxCollection uuidInfoBoxCollection = uuidInfoBoxParser.ParseFromImage(fs);
```

## 6 Object Reference

---

For a complete object reference of each property, method, class, event, and interface including help and examples, see the integrated HTML 2.0 help. This help can be accessed from the start menu, or from within Visual Studio .NET.