CCR Exif v0.9.9

A small Delphi class library to edit, create and delete Exif metadata in JPEG files, including that in any associated XMP packet

Chris Rolliston

http://delphihaven.wordpress.com/

Contents

Release History

0.9.9 (2009-11-19)

- Two main themes for this release: adding sanity checks to the parsing code (thus, you shouldn't get access violations when attempting to stream in corrupted data any more) and better MakerNote support.
    - All TIFF offsets are now sanity checked.
    - Internally, there's a 'two strikes and out' policy when parsing an IFD — two invalid tag headers in a row, and parsing for that directory is abandoned. Check the new LoadErrors property on TExifSection to determine whether there were any issues.
    - The tag structures of Canon, Panasonic and Sony MakerNotes are now understood. You access the tags via either the MakerNote.Tags or Sections[esMakerNote] properties on TCustomExifData. The interpretation of tag **values** is still left to the user however, since the alternative was to add reams and reams of boilerplate tag property code.
    - Existing MakerNotes can be edited with TExifDataPatcher; TExifData will probably come later (I'm offsetted out at the moment…).
- Other, more minor changes:
    - Fixed typo in GPS direction tag setter which meant the value could never be changed.
    - Added memory leak fix to CCR.XMPUtils.pas suggested by David Hoyle.
    - Added delay loading semantics to the XMPPacket property of TCustomExifData, the idea being that attempts to read Exif tags should not ever lead to an EInvalidXMPPacket exception being raised. Equivalent behaviour has been built into the new MakerNote parser code too.
    - More helper methods of the TryGetXXXValue and ReadXXX kind.
    - Surfaced two interop IFD tags as properties on TCustomExifData.
    - MakerNote data are now moved back to their original position on save if the OffsetSchema tag had been set. (Actually, this should have been the case for the previous release but for a bug, typing Inc where I meant Dec.)
    - XMP 'about' attribute value now retrieved if previously written without a namespace.
    - Demos rejigged a bit — PanasonicMakerNoteView.exe removed (its functionality has been added to an improved ExifList.exe), and two new console ones added (CreateXMPSidecar.exe and PanaMakerPatch.exe).

0.9.8 (2009-10-18)

- Fixed how the ExifImageWidth and ExifImageHeight properties weren't flexible enough to understand word-sized values (the Exif spec says these tags can be either words or longwords).
- By default, when streaming out, the MakerNote tag (if it exists) now keeps the data offset it had when streamed in. This means you are free to add, remove and edit tags as you please with TExifData without fear of corrupting internal MakerNote offsets, though there's a small cost in terms of file size. If you want the old behaviour, you

can get it back by setting the new MakerNotePosition property of TExifData to mpCanMove.

- Much better XMP support has been implemented:
  - A new unit, CCR.Exif.XMPUtils, has been added; this implements an XMP packet parser directly on top of xmldom.pas (all that excess code for so little gain in XMLDoc.pas makes my eyes water…). A new demo, XMP Browser, shows its basic usage.
  - TCustomExifData has acquired an XMPPacket property that gets loaded in LoadFromJPEG, written out in SaveToJPEG, and updated as appropriate when certain tag properties are set. The default behaviour is to only change existing XMP property values, though this can be altered to mimic the behaviour of Vista's Windows Explorer, which adds them as well, by setting ExifData.XMPWritePolicy to xwAlwaysUpdate.
- EReadError (typically raised by TStream.ReadBuffer) is now trapped in one of the low-level loading routines so as to be 'friendlier' (to the end user!) when you attempt to load a corrupt Exif block. To be honest, I'm not massively keen on doing this, but anyhow, the change has been made.
- Other changes:
  - Some missing tag properties from v2.2 of the Exif standard added to TCustomExifData (Contrast, FlashPixVersion, Saturation, Sharpness).
  - ISOSpeedRatings is now a class rather than an array property.
  - Some of the GPS properties have become class properties.
  - Setting a string tag to an empty string now removes it; likewise, setting an invalid fraction value, where the destination tag only has one value currently, will remove the tag too.
  - Removed from CCR.Exif.JPEGUtils the symbols that were deprecated last time around.
  - Bug fix — TCustomExifData.LoadFromJPEG now clears the tags if no Exif segment was found in the source image.
  - Behaviour change — TExifData.SaveToJPEG now enforces a segment order of JFIF → Exif → XMP, and moreover, moves those segments to the top of the file if they weren't there already.
  - Renamed StreamHasXMPHeader to StreamHasXMPSegmentHeader to better reflect what it tests for.
  - TJPEGMarkerHeader renamed TJPEGSegmentHeader.
  - PosOfDataInStream property of IFoundJPEGSegment renamed OffsetOfData.
  - IGetJPEGHeaderParser collapsed into IJPEGHeaderParser.

0.9.7 (2009-06-22)

- The focus of this release was making the code more aware of what more recent versions of Windows Explorer (amongst other Microsoft applications) do when a person uses them to edit JPEG metadata.
- TExifData.SaveToStream now adds or updates the Microsoft-defined OffsetSchema tag if a MakerNote is defined (see http://support.microsoft.com/kb/927527).

- Various enumerated types have acquired a xxTagMissing value to more easily determine whether the underlying tag actually exists.
- The following symbols have been added (see the documentation for more info):
  - PreserveXMPData property to TCustomExifData; default is False.
  - IsPadding and SetAsPadding methods to TExifTag; RemovePaddingTag method to TExifSection; RemovePaddingTags method to TCustomExifData. All these concern the padding tags Microsoft applications tend to write out.
  - RemoveMetaDataFromJPEG, StreamHasExifHeader and StreamHasXMPHeader global functions to CCR.Exif.
  - DigitalZoomRatio, FocalLengthIn35mmFilm, GainControl, ImageUniqueID, MakerNoteDataOffset, Rendered, SceneCaptureType, SubjectDistanceRange, ThumbnailOrientation, ThumbnailResolution, and WhiteBalanceMode properties to TCustomExifData — all surface standard tags I missed out previously.
  - JPEGHeader global routine to CCR.Exif.JPEGUtils as a more flexible replacement for ParseJPEGHeader; unlike the latter, it doesn't take a callback method, being used instead with the for/in syntax:

    ```
    var
      Segment: IFoundJPEGSegment;
      JPEGImage: TJPEGImage;
    begin
      for Segment in JPEGHeader('filename.jpg') do
        ...
      for Segment in JPEGHeader(JPEGImage) do
        ...
    ```
- Deprecated:
  - The jmExif constant in CCR.Exif.JPEGUtils — use jmApp1 instead. The reason for this change is that Exif and XMP segments share the same marker number; the value of what was jmExif, then, isn't unique to Exif.
  - DefJFIFData global variable in CCR.Exif.JPEGUtils — this hasn't been used internally since v0.9.5.
  - ParseJPEGHeader function in CCR.Exif.JPEGUtils — use JPEGHeader with a for/in loop instead.
  - RemoveExifDataFromJPEG functions in CCR.Exif — use the new RemoveMetaDataFromJPEG functions instead. That said, a bug in the overload that takes a stream object has been fixed.
  - WriteJPEGMarkerToStream procedures in CCR.Exif.JPEGUtils — renamed WriteJPEGSegmentToStream.
- Custom exception classes slightly rearranged and documented.
- JPEG Dump and Exif List demos updated slightly.

0.9.6 (2009-06-14)

- Some documentation added, albeit only in the form of a PDF file.
- Fixed potential data corruption bug in TJpegImageEx.SaveToStream. As part of this fix, whereas before, a specific JFIF segment was always written out, now nothing is touched (including the JFIF segment — even if missing) other than any existing Exif segment.

- Added SaveToJPEG method to TExifData. Calling this will replace any Exif metadata in the specified JPEG file with what is currently defined by the TExifData object. In combination with the existing LoadFromJPEG method, this provides a more flexible way of editing existing Exif metadata than that provided by TExifDataPatcher. (Note that you could have got similar behaviour before if you had used TJpegImageEx instead of TExifData directly.) The old caveat about MakerNote data still remains though.
- Added RemoveExifDataFromJPEG global routine, which takes either a file name or a TJPEGImage instance as its parameter and returns True if there were Exif data to remove.
- Fixed TExifData.SaveToStream so that it now saves GPS and/or Interop sections (doh!). Nonetheless, they are only written out if they have any tags. So, say you load into an TExifData instance data that includes GPS info, call Clear on the esGPS section, and save out again; doing this will remove the GPS sub-IFD from actual file. Now, this behaviour will only save six bytes compared to writing out an empty IFD or sub-IFD. Nonetheless, it is aesthetically quite pleasing I think, and matches how the thumbnail IFD isn't (and never was) written out when Thumbnail.Empty returns True.
- Partly due to rewriting the saving code, partly due to a change of heart, the CanSafelySaveTag, GetTagsToSave, SaveSave and OnCanSafelySaveTag members of TExifData have been removed.
- In TCustomExifData, the ResolutionUnit, XResolution and YResolution properties have been replaced with a single Resolution property, with the FocalPlaneResolutionUnit, FocalPlaneResolutionX and FocalPlaneResolutionY properties replaced with FocalPlaneResolution. In the course of doing this, I fixed a bug of erroneously assuming the two sets of properties shared the same section.
- The ColorSpace property on TCustomExifData has changed its type from Word to a new word-sized enumeration (TExifColorSpace).
- Setting a tag or tag element to its current value is now much less likely to cause the container object to have its Modified property set to True.
- Added TagExists function to TExifSection.
- Added MissingOrInvalid function to TExifFlashInfo.
- More complete implementation of TExifFlashInfo.Assign.
- You can now remove tags from a TExifDataPatcher instance. Because of this, TCustomExifData now publicly exposes the Clear method, and TExifSection has acquired Remove and Clear from TExtendableExifSection, with ClearTag being deleted.
- Added testing mode to Exif List and JPEG Dump demos — pass the /test switch as a parameter to the EXE. Done mainly to test the revised saving code in TExifData, though it demonstrates the latter's new SaveToJPEG method I suppose…
- Thanks to Valerian Kadyshev for suggestions that were partly implemented in this release, along with David Hoyle again for a few more bug reports.

0.9.5b (2009-06-04)

- And another regression fixed… Should compile in D2009 again now.

0.9.5a (2009-06-03)

- Regression fixed — original release of 0.9.5 broke setting string tags in D2009. Thanks to commentator Andreas for spotting the bug.

0.9.5 (2009-06-01)

- A few typos fixed in the getter and setters of TCustomExifData.
- TExifTag.SetAsString fixed when DataType = tdUndefined.
- A new Boolean property, EnforceASCII, added to TCustomExifData; default value is True so that tags with a data type of tdASCII will now only allow ASCII data to be assigned to them. Set EnforceASCII to False to restore the old, lax behaviour.
- For Delphi 2009, ASCII tags are now surfaced as normal strings; because of this, you shouldn't cast to TiffString any more when setting a string tag. Furthermore, TiffString is itself now just type def'ed to AnsiString rather than an English AnsiString specifically.
- The Comments property of TCustomExifData now no longer checks for ttImageDescription.
- The ExifVersion and GPSVersion properties of TCustomExifData have changed their types, with the old ones removed.
- The DirectlyPhotographed property of TCustomExifData has been removed and replaced with SceneType.
- The SourceIsDigitalCamera property of TCustomExifData has been removed and replaced with FileSource.
- WritePreciseTimes property of TCustomExifData has become AlwaysWritePreciseTimes.
- drTrue → drTrueNorth, drMagnetic → drMagneticNorth.
- ShutterSpeedInMSecs function and GPSDateStamp, GPSDateTimeUTC and GPSDifferentialApplied properties added to TCustomExifData (GPSDateTimeUTC converts and combines the values stored in GPSDateStamp and GPSTimeStampXXX).
- The internal enumerator types have acquired a T prefix.
- Exif List demo added.
- Thanks to David Hoyle for pointing out needed bug fixes.

0.9.0 (2009-05-12)

- Original release.

Introduction

Exif — What Is It?

Exif metadata is the sort of metadata that most digital cameras write to the JPEG files they create, providing detailed information about the camera and photograph on the one hand and (optionally) a thumbnail image on the other. Since at least Windows XP, Windows Explorer understands it too, displaying key parts of it by default; numerous other programs then allow reading and (sometimes) writing it as well.

Internally, the Exif format is based upon the TIFF specification. To that effect, data can be stored in either big or little-endian format, and is structured into 'directories' (in full, 'image file directories' or IFDs) that are composed of 'tags', each tag then having a data type and data (the latter may or may not directly follow the tag header). Complicating matters is that individual tags may contain 'sub-directories', with the fact that any one tag is like this being something any parser needs to have special knowledge of — while the latest TIFF specification defines a 'sub-directory' tag data type, Exif does not use it.

Nevertheless, this problem only really raises its head with respect to the so-called 'MakerNote' tag. While this typically takes the form of a 'normal' sub-IFD, it may or may not have a header of varying length, may or may not respect the endianness of the containing Exif structure, and may or may not have other quirks as well.

That said, in CCR Exif, the distinction between 'directories' and 'sub-directories' is collapsed into the concept of 'sections'. The mapping goes as thus:

CCR Exif term	'Standard' Exif term	Notes
General section	Main IFD	Contains generic TIFF tags together with tags defined by Windows Explorer.
Details section	Exif sub-IFD	Contains Exif-specific tags.
Interop section	Interop sub-IFD	A bit pointless IMO, but hey, it's there.
GPS section	GPS sub-IFD	Contains positioning information tags.
Thumbnail section	Thumbnail IFD	Generally won't have anything interesting since the thumbnail should be a JPEG, and thus, have any important information in its own header.

As exposed in TCustomExifData (and thus, TExifData and TExifDataPatcher), standard tags have their data presented in a manner that does not require knowledge of what directory or sub-directory they belong to. Nonetheless, you are free to add, edit or delete tags at the level of individual sections if you so wish.

Basic Usage of CCR Exif

Say you want to retrieve the camera make and model; the following function will do the job:

```
uses
  CCR.Exif;

procedure ReadCameraMakeAndModel(const FileName: string;
  out Make, Model: string);
var
  ExifData: TExifData;
begin
  ExifData := TExifData.Create;
  try
    ExifData.LoadFromJPEG(FileName);
    Make := ExifData.CameraMake;
    Model := ExifData.CameraModel;
  finally
    ExifData.Free;
  end;
end;
```

Alternatively, if you want to load the actual image too, you can use TJpegImageEx, a simple descendent of TJPEGImage from the VCL that exposes an ExifData property. So, assuming a form with a TImage called imgJPEG and two TEdit controls called edtMake and edtModel:

```
uses
  CCR.Exif;

procedure TMyForm.LoadJPEG(const FileName: string);
var
  Jpeg: TJpegImageEx;
begin
  Jpeg := TJpegImageEx.Create;
  try
    Jpeg.LoadFromFile(FileName);
    imgJPEG.Picture.Assign(Jpeg);
    edtMake.Text := Jpeg.ExifData.CameraMake;
    edtModel.Text := Jpeg.ExifData.CameraModel;
  finally
    Jpeg.Free;
  end;
end;
```

Lastly, say you want to delete all tags with binary ('undefined') data in a certain file, considering them unnecessary because they are proprietary. To enumerate tags, you use the for-in syntax, calling Delete on the currently-enumerated tag as necessary. Thus, the following code would do the trick:

```
uses
  CCR.Exif;

procedure StripBinaryTags(const FileName: string);
var
  ExifData: TExifData;
  Section: TExifSection;
  Tag: TExifTag;
begin
  ExifData := TExifData.Create;
  try
    ExifData.LoadFromJPEG(FileName);
    for Section in Image.ExifData do
      for Tag in Section do
        if Tag.DataType = tdUndefined then
          Tag.Delete;
    ExifData.SaveToJPEG(FileName);
  finally
    ExifData.Free;
  end;
end;
```

An alternative to this would be to use TExifDataPatcher, which shares a common ancestor with TExifData. The difference is that while TExifData rewrites what it loads from scratch, TExifDataPatcher preserves the file structure exactly as it found it. This removes a lot of flexibility — in particular, no new tags may be added and no existing ones can have their data size increased. Nonetheless, tags can still be deleted, with the benefit being that any non-standard sub-directories have no chance of being corrupted:

```
uses
  CCR.Exif;

procedure StripBinaryTagsAlt(const FileName: string);
var
  ExifData: TExifDataPatcher;
  Section: TExifSection;
  Tag: TExifTag;
begin
  ExifData := TExifDataPatcher.Create;
  try
    ExifData.OpenFile(FileName);
    for Section in ExifData do
      for Tag in Section do
        if Tag.DataType = tdUndefined then
          Tag.Delete;
    ExifData.UpdateFile; //flush changes
  finally
    ExifData.Free;
  end;
end;
```

Originally, the main rationale of TExifDataPatcher was to allow limited editing that would not corrupt any MakerNote tag data. Since v0.9.8, however, TExifData's saving code will by default always save the latter to its original location.

Lastly, the following code creates an XMP 'sidecar' file for a given JPEG image:

```
uses
  CCR.Exif;

procedure CreateXMPSidecar(const JpegFile: string);
var
  ExifData: TExifData;
begin
  ExifData := TExifData.Create;
  try
    ExifData.LoadFromJpeg(JpegFile);
    ExifData.XMPWritePolicy := xwAlwaysUpdate;
    ExifData.Rewrite;
    ExifData.XMPPacket.SaveToFile(ChangeFileExt(JpegFile, '.xmp'));
  finally
    ExifData.Free;
  end;
end;
```

Demo Applications

For more advanced usage, please refer to the seven demos. In turn, they demonstrate the following:

- Listing known tag values (ExifList.exe).
- Creating a JPEG image with Exif metadata (author, subject, title, star rating, etc.) from scratch (Screenshooter.exe).
- Patching existing JPEG files with revised 'time taken' values (ExifTimeShift.exe).
- Parsing a JPEG file header (JpegDump.exe).
- Patching Panasonic MakerNote data (PanaMakerPatch.exe).
- Creating an XMP sidecar file for a given JPEG (CreateXMPSidecar.exe).
- Browsing a JPEG file's XMP packet, XMP being a 'meta-metadata' format that more recent versions of Windows Explorer write out alongside Exif (XMPBrowser.exe).
- Rewriting Exif and/or XMP metadata (ResavingTest.exe).

TExifTag Class

As its name suggests, an instance of TExifTag represents an Exif tag. Generally used as the item class of TExifSection, you can instantiate it independently if you want though.

Properties

AsBytes

```
  property AsBytes: TBytes read GetAsBytes write SetAsBytes;
```

Reads/writes tag data as a TBytes dynamic array.

AsString

```
  property AsString: string read GetAsString write SetAsString;
```

Reads/writes a string representation of the data, whatever the data type.

Data

```
  property Data: Pointer read FData;
```

Provides direct access to the tag's data in the form of an untyped pointer to an in-memory buffer. Generally you will want to use the higher level properties of TCustomExifData instead; nonetheless, it's there if you need it, be it for tags that haven't been surfaced at the TCustomExifData level or for some other reason. For example, to read the 'raw' value for the Exif version tag, you could use the following code:

```
uses
  CCR.Exif, CCR.Exif.TagIDs;

function GetRawExifVersion(ExifData: TCustomExifData): AnsiString;
var
  Tag: TExifTag;
begin
  if ExifData[esDetails].Find(ttExifVersion, Tag) then
    SetString(Result, PAnsiChar(Tag.Data), Tag.DataSize)
  else
    Result := '';
end;
```

Note that the ExifVersion property of TCustomExifData surfaces this particular tag in a more user-friendly fashion however.

DataSize

```
  property DataSize: Integer read GetDataSize;
```

Returns ElementCount multiplied by the size of one element of DataType (which will be 1 for tdAscii, tdByte and tdUndefined, 2 for tdWord and tdSmallInt, and so on).

DataType

```
  type
    TExifDataType = (tdByte = 1, tdAscii, tdWord, tdLongWord,
      tdLongWordFraction, tdShortInt, tdUndefined, tdSmallInt, tdLongInt,
      tdLongIntFraction, tdSingle, tdDouble, tdSubDirectory);
```

```
property DataType: TExifDataType read FDataType write SetDataType;
```

As its name suggests, this property read/writes the tag's data type. When changing it, the data is preserved when both old and new types are integral or fractions, otherwise the data becomes undefined.

ElementCount

```
property ElementCount: LongInt read FElementCount write
  SetElementCount;
```

If a tag should contains multiple values, check or change this property. Note that for string tags, each 'element' is just a single character, so can't have more than one string value in a single tag. Furthermore, string tag data includes a null terminator, so for string tags, ElementSize = Length(AsString) + 1.

ID

```
type
  TExifTagID = type Word;

property ID: TExifTagID read FID write SetID;
```

Reads/writes the identifying number of the tag, which should equal one of the ttXXX values in CCR.Exif.TagIDs. Note that no two tags within the same section can share the same ID — try to have it otherwise, and an exception will be raised.

Methods

Assign

```
procedure Assign(Source: TExifTag);
```

If Source is nil, sets ElementCount to 0, else copies the ID, ElementCount and Data from Source.

Changed

```
procedure Changed; overload;
```

Call this if you use the Data property to write new data directly.

Delete

```
procedure Delete;
```

Equivalent to calling Free — they do the same thing. Exists only for consistency with some standard VCL classes.

HasWindowsStringData

```
function HasWindowsStringData: Boolean;
```

In the jargon of CCR Exif, a 'Windows string' tag is one defined by Windows Explorer rather than the Exif specification proper; internally it is stored in a special way, though the details are hidden by TExifTag.

IsPadding

```
function IsPadding: Boolean;
```

Returns whether the tag is padding, and thus, does not contain any useful data. Internally, a padding tag has a data type of tdUndefined and ID of ttWindowsPadding, with the first two bytes of its data equalling the value of ttWindowsPadding too.

SetAsPadding

```
type
  TExifPaddingTagSize = 2..High(LongInt);

procedure SetAsPadding(Size: TExifPaddingTagSize);
```

Converts the tag to a padding tag of the specified size. To convert a padding tag to a normal tag, simply change its data type and/or ID.

UpdateData

```
procedure UpdateData(NewDataType: TExifDataType;
  NewElementCount: LongInt; const NewData); overload;

procedure UpdateData(const NewData); overload;
```

Low-level way to update a tag’s data. Generally you will want to use the higher level members of TExifSection and TCustomExifData instead; nonetheless, it’s there if you want it.

TExifSection Class

This class encapsulates the concept of an Exif directory (IFD) or sub-directory (sub-IFD).

Properties

Count

```
property Count: Integer read GetCount;
```

Returns the number of tags in the section.

Kind

```
type
  TExifSectionKindEx = (esUserDefined, esGeneral, esDetails, esInterop,
    esGPS, esThumbnail);

property Kind: TExifSectionKindEx read FKind;
```

Returns the section type — note that esGeneral denotes the ‘main IFD’ in standard Exif-speak, esDetails the ‘Exif sub-IFD’.

LoadErrors

```
type
  TExifSectionLoadErrors = set of (leBadOffset, leBadTagCount,
    leBadTagHeader);

property LoadErrors: TExifSectionLoadErrors read FLoadErrors;
```

Returns an empty set if the section loaded cleanly, a non-empty set otherwise.

Modified

```
property Modified: Boolean read FModified write FModified;
```

Read/writes whether the section has been changed since the last load, something particularly important for TExifDataPatcher.UpdateFile.

Owner

```
property Owner: TCustomExifData read FOwner;
```

Returns the container object.

Methods

GetEnumerator

```
function GetEnumerator: TEnumerator;
```

Support function required by Delphi itself to enable the for/in syntax on a TExifSection instance —

```
procedure EnumSection(Section: TExifSection);
var
  Tag: TExifTag;
begin
  for Tag in Section do
    ShowMessage(Tag.AsString);
end;
```

Clear

```
procedure Clear;
```

Removes all tags contained by the section. Remember to call SaveToJPEG (if using TExifData), UpdateFile (if using TExifDataPatcher) or SaveToFile/SaveToStream (if using TJPEGImageEx) to actually effect any changes.

EnforceASCII

```
function EnforceASCII: Boolean; virtual;
```

Returns True if Owner isn't nil and Owner.EnforceASCII (which is a read/write property) returns True, otherwise returns False.

Find

```
function Find(ID: TExifTagID; out Tag: TExifTag): Boolean;
```

If one exists, sets Tag to a contained object with the specified ID number and returns True; otherwise, sets Tag to nil and returns False.

GetXXXValue

```
function GetByteValue(TagID: TExifTagID; Index: Integer; Default: Byte;
  MinValue: Byte = 0; MaxValue: Byte = High(Byte)): Byte;

function GetDateTimeValue(TagID: TExifTagID;
  const Default: TDateTime = 0): TDateTime;

function GetFractionValue(TagID: TExifTagID;
  Index: Integer): TExifFraction; overload;

function GetFractionValue(TagID: TExifTagID; Index: Integer;
  const Default: TExifFraction): TExifFraction; overload;

function GetLongWordValue(TagID: TExifTagID; Index: Integer;
  Default: LongWord): LongWord;

function GetStringValue(TagID: TExifTagID;
  const Default: string = ''): string;

function GetWindowsStringValue(TagID: TExifTagID;
  const Default: UnicodeString = ''): UnicodeString; overload;

function GetWordValue(TagID: TExifTagID; Index: Integer; Default: Word;
  MinValue: Word = 0; MaxValue: Word = High(Word)): Word;
```

Returns the specified tag value if it exists and its data type is either that of method name or its unsigned or signed equivalent (if applicable); otherwise, returns the value of the Default parameter. The behaviour here is similar to that of the ReadXXX methods of

TCustomIniFile.

IsExtendable

```
function IsExtendable: Boolean;
```

Returns whether the instance is (or descends) from the TExtendableExifSection class.

Remove

```
function Remove(ID: TExifTagID): Boolean;
```

If a tag with the specified ID exists, deletes it and returns True; otherwise, returns False.

RemovePaddingTag

```
function RemovePaddingTag: Boolean;
```

If a padding tag exists in the section, it is deleted and the function returns True, else the function returns False.

SetXXXValue

```
function SetByteValue(TagID: TExifTagID; Index: Integer;
  Value: Byte): TExifTag;

procedure SetDateTimeValue(TagID: TExifTagID;
  const Value: TDateTime);

function SetFractionValue(TagID: TExifTagID; Index: Integer;
  const Value: TExifFraction);

function SetLongWordValue(TagID: TExifTagID; Index: Integer;
  Value: LongWord): TExifTag;

function SetSignedFractionValue(TagID: TExifTagID; Index: Integer;
  const Value: TExifSignedFraction);

procedure SetStringValue(TagID: TExifTagID;
  const Value: string);

function SetWindowsStringValue(TagID: TExifTagID;
  const Value: UnicodeString);

function SetWordValue(TagID: TExifTagID; Index: Integer;
  Value: Word): TExifTag;
```

Sets the specified tag or tag element, adding a tag object or changing the existing tag object’s data type if necessary.

TagExists

```
function TagExists(ID: TExifTagID; ValidDataTypes: TExifDataTypes =
  [Low(TExifDataType)..High(TExifDataType)];
  MinElementCount: LongInt = 1): Boolean;
```

Returns True if a tag with the specified ID, data type and minimum element count exists.

TExtendableExifSection Class

Descending from TExifSection, this is the section class for TExifData, adding methods to add tags.

Methods

Add

```
function Add(ID: TExifTagID; DataType: TExifDataType;
  ElementCount: LongInt = 1): TExifTag;
```

Adds a tag with the specified attributes. If a tag already exists with the specified ID, an exception is raised.

AddOrUpdate

```
function AddOrUpdate(ID: TExifTagID; DataType: TExifDataType;
  ElementCount: LongInt): TExifTag; overload;

function AddOrUpdate(ID: TExifTagID; DataType: TExifDataType;
  ElementCount: LongInt; const Data): TExifTag; overload;
```

If a tag with the specified ID already exists, its attributes are changed to those specified; otherwise, a new tag is added and initialised with the given parameters.

Assign

```
procedure Assign(Source: TExifSection);
```

Calls Clear before copying over the tags in Source.

CopyTags

```
procedure CopyTags(Section: TExifSection);
```

Similar to Assign, but doesn’t clear the existing tags first.

TCustomExifVersion Class

Descending from TPersistent, this is the base class for the TExifVersion, TFlashPixVersion, TGPSVersion and TInteropVersion classes. Use of a common base class abstracts from how the underlying tag data may be stored differently.

Properties

AsString

```
property AsString: string read GetAsString write SetAsString;
```

Reads/writes a string representation of the data, e.g. ‘2.21’. The getter function uses the value of DateSeparator from the SysUtils unit; the setter function then checks for ‘.’, ‘,’ and (if something else entirely) DateSeparator, though if the new value is an empty string, the underlying tag (ttExifVersion or ttGPSVersionID) is removed.

Owner

```
property Owner: TCustomExifData read FOwner;
```

Returns the container object.

Major

```
type
  TExifVersionElement = 0..9;

property Major: TExifVersionElement read GetMajor write SetMajor;
```

Reads/writes the major version number, typically 2.

Minor

```
type
  TExifVersionElement = 0..9;

property Minor: TExifVersionElement read GetMinor write SetMinor;
```

Reads/writes the minor version number, typically 2.

Release

```
type
  TExifVersionElement = 0..9;

property Release: TExifVersionElement read GetRelease write SetRelease;
```

Reads/writes the release version number.

Methods

Assign

```
procedure Assign(Source: TPersistent); override;
```

If Source is neither nil nor another TCustomExifVersion instance, then the inherited implementation is called, raising an exception. Otherwise, if Source is nil or has an underlying tag that doesn’t exist, our own underlying tag (ttExifVersion or ttGPSVersionID) is removed, else the values of Source are copied across.

MissingOrInvalid

```
  function MissingOrInvalid: Boolean;
```

Returns True if the underlying tag does not exist, False otherwise.

TCustomExifData Class

Descending directly from TInterfacedPersistent (though not itself implementing any interfaces), TCustomExifData is the base class for TExifDataPatcher and TExifData, and as such, is the key class of the library. In the main, the class presents tag data as originally found, though signed data is transparently presented as unsigned or vice versa as appropriate. For many properties of TCustomExifData, then, a good Exif reference will be handy to find out what the precise values mean.[1] If a tag value is exposed as a property on TCustomExifData and it is not documented below, please refer to that instead.

[1] E.g. http://www.awaresystems.be/imaging/tiff/tifftags/privateifd.html.

Class Methods

RegisterMakerNoteType

```
type
  TMakerNoteTypePriority = (mtTestForLast, mtTestForFirst);

class procedure RegisterMakerNoteType(AClass: TExifMakerNoteClass;
  Priority: TMakerNoteTypePriority);

class procedure RegisterMakerNoteTypes(const AClasses: array of
  TExifMakerNoteClass; Priority: TMakerNoteTypePriority);
```

Registers the specified TExifMakerNote descendant(s). When new data that includes a maker note is streamed into a TExifData or TExifDataPatcher instance, registered TExifMakerNote classes are enumerated until one returns True from its FormatIsOK method, after which it is instantiated and used to load the maker note’s tags.

UnregisterMakerNoteType

```
class procedure UnregisterMakerNoteType(AClass: TExifMakerNoteClass);
```

Unregisters the specified TExifMakerNote descendent.

Properties

AlwaysWritePreciseTimes

```
property AlwaysWritePreciseTimes: Boolean read FAlwaysWritePreciseTimes
  write FAlwaysWritePreciseTimes default False;
```

Controls whether sub-second tags are created and set when a date/time property is written to and no corresponding sub-second tag currently exists..

Author

```
property Author: UnicodeString read GetAuthor write SetAuthor;
```

The property getter first attempts to retrieve the ttWindowsAuthor tag, and if that doesn’t exist or is empty, the ttArtist tag (this behaviour mimics that of Windows Explorer). The property setter then always sets the ttWindowsAuthor tag **if** it already exists **or** the new value includes non-ASCII characters. The ttArtist tag is then set **if** the new value *only* includes ASCII characters. Note that UnicodeString is mapped to WideString in pre-Delphi 2009.

Comments

```
property Comments: UnicodeString read GetComments write SetComments;
```

The property getter looks firstly for ttWindowsComments, and secondly for ttUserComment (again, this mimics Windows Explorer). The property setter then **always** sets ttWindowsComments, and **if** it already exists, ttUserComment.. Note that UnicodeString is mapped to WideString in pre-Delphi 2009.

DateTime, DataTimeOriginal and DateTimeDigitized

```
property DateTime: TDateTime index ttDateTime read GetGeneralDateTime
  write SetGeneralDateTime;

property DateTimeOriginal: TDateTime index ttDateTimeOriginal
  read GetDetailsDateTime write SetDetailsDateTime;

property DateTimeDigitized: TDateTime index ttDateTimeDigitized
  read GetDetailsDateTime write SetDetailsDateTime;
```

As stored, Exif data/times are strings with a particular format; as presented in TCustomExifData, they are TDateTime values. Aside from converting the underlying string into a TDataTime value, however, the property getter also checks to see whether a corresponding SubSecTime tag exists; it if does, then its value is merged into the result. As for the property setter, this then sets the corresponding SubSecTime tag as well as the 'main' one **if** it already exists **or** AlwaysWritePreciseTimes is True. Because of this behaviour, the SubsecTime, SubsecTimeOriginal and SubsecTimeDigitized properties can generally be ignored.

Empty

```
property Empty: Boolean read GetEmpty;
```

Returns True if every section object contains no tags, False otherwise. Note that TExifData overrides the GetEmpty method to return False if a thumbnail image is loaded.

Endianness

```
type
  TEndianness = (SmallEndian, BigEndian);

property Endianness: TEndianness read FEndianness write SetEndianness;
```

After data has been streamed in, this property returns whether it was in big or little endian format (alias Motorola or Intel byte order) — Windows and so Delphi is little endian, so big endian data is converted as it is loaded. Toggle the property afterwards to change the format when the data is streamed back out. (When creating Exif data afresh, the default value is SmallEndian.)

EnforceASCII

```
property EnforceASCII: Boolean read FEnforceASCII write FEnforceASCII
```

```
  default True;
```

Standard Exif string tags (i.e., all string tags other than ttUserComment and the ones Windows Explorer defines) should contain only ASCII characters. By default, then, EnforceASCII is True with the effect of raising an exception whenever you attempt to set non-ASCII data to relevant tags. To allow setting ANSI data, set it to False.

ExifVersion

```
property ExifVersion: TCustomExifVersion read FExifVersion
  write SetExifVersion;
```

Exposes the ttExifVersion tag from the details section. Please see the documentation for TCustomExifVersion for more information.

Flash

```
type
  TExifFlashMode = (efUnknown, efCompulsoryFire, fCompulsorySuppression,
    efAuto);

  TExifStrobeLight = (esNoDetectionFunction, esUndetected, esDetected);

  TExifFlashInfo = class(TPersistent)
  //private and protected sections snipped
  public
    constructor Create(ASection: TExifSection);
    procedure Assign(Source: TPersistent); override;
    function MissingOrInvalid: Boolean; //does the underlying tag exist?
    property BitSet: TWordBitSet read GetBitSet write SetBitSet;
    property Section: TExifSection read FSection;
  published
    property Fired: Boolean index 0 read GetBit write SetBit;
    property Mode: TExifFlashMode read GetMode write SetMode;
    property Present: Boolean index 5 read GetInverseBit
      write SetInverseBit;
    property RedEyeReduction: Boolean index 6 read GetBit
      write SetBit;
    property StrobeEnergy: TExifFraction read GetStrobeEnergy
      write SetStrobeEnergy;
    property StrobeLight: TExifStrobeLight read GetStrobeLight
      write SetStrobeLight stored False;
  end;

property Flash: TExifFlashInfo read FFlash write SetFlash;
```

This class property groups the ttFlash and ttFlashEnergy tags, breaking down the former into its component parts, bits being read and written for whether a flash was present, what its mode was, whether it was fired, etc. Use the MissingOrInvalid method to see whether the underlying tag actually exists.

ExifVersion

```
property FlashPixVersion: TCustomExifVersion read FFlashPixVersion
  write SetFlashPixVersion;
```

Exposes the ttFlashPixVersion tag from the details section. Please see the documentation for TCustomExifVersion for more information.

FocalPlaneResolution

```
property FocalPlaneResolution: TExifResolution
  read FFocalPlaneResolution write SetFocalPlaneResolution;
```

Sharing a class type with the Resolution property, this groups the ttFocalPlaneXResolution, ttFocalPlaneYResolution and ttFocalPlaneResolutionUnit tags from the details section.

GPSVersion

```
property GPSVersion: TCustomExifVersion read FGPSVersion write
  SetGPSVersion;
```

Exposes the ttGPSVersionID tag from the GPS section. Please see the documentation for TCustomExifVersion for more information.

InteropVersion

```
property InteropVersion: TCustomExifVersion read FInteropVersion write
  SetInteropVersion;
```

Exposes the ttInteropVersion tag from the interop section. Please see the documentation for TCustomExifVersion for more information.

MakerNote

```
type
  TExifDataOffsetsType = (doFromExifStart, doFromMakerNoteStart,
    doFromIFDStart);

  TExifMakerNote = class abstract
    class function FormatIsOK(SourceTag: TExifTag): Boolean; overload;
    property DataOffsetsType: TExifDataOffsetsType read
      FDataOffsetsType;
    property Endianness: TEndianness read FEndianness;
    property Tags: TExifSection read FTags;
  end;

  TUnrecognizedMakerNote = class sealed(TExifMakerNote);

  TCanonMakerNote = class(TExifMakerNote)
  //...
  end;

  TPanasonicMakerNote = class(TExifMakerNote)
  //...
  end;

  TSonyMakerNote = class(TExifMakerNote)
  //...
  end;

property MakerNote: TExifMakerNote read GetMakerNote;
```

The actual type of the MakerNote property will be a descendent of TExifMakerNote, determined by enumerating all registered TExifMakerNote descendents and calling their IsFormatOK method. If no registered descendent returns True, the type of MakerNote will be TUnrecognisedMakerNote.

Note that while the maker note type is determined up front, maker note tags are only parsed when called for.

Modified

```
property Modified: Boolean read FModified write SetModified;
```

Returns True if one or more tags have been modified since data was last streamed in.

Resolution

```
property Resolution: TExifResolution read FResolution
  write SetResolution;
```

Sharing a class type with the FocalPlaneResolution property, this groups the ttXResolution, ttYResolution and ttResolutionUnit tags from the general section.

Sections

```
type
  TExifSectionKindEx = (esUserDefined, esGeneral, esDetails, esInterop,
    esGPS, esThumbnail);
  TExifSectionKind = esGeneral..esThumbnail;

property Sections[Section: TExifSectionKind]: TExifSection
  read GetSection; default;
```

Returns the specified section.

UserRating

```
type
  TWindowsStarRating = (urUndefined, urOneStar, urTwoStars,
    urThreeStars, urFourStars, urFiveStars);

property UserRating: TWindowsStarRating read GetUserRating
  write SetUserRating;
```

Reads/writes the star rating tag defined by more recent versions of Windows Explorer.

XMPPacket

```
property XMPPacket: TXMPPacket read FXMPPacket;
```

The associated XMP packet, streamed in by LoadFromJpeg, updated by the tag property setters, and streamed out by SaveToJpeg along with the Exif metadata proper. TXMPPacket is implemented in CCR.Exif.XMPUtils.pas.

Note that the XMP packet won't be parsed until either the XMPPacket property is explicitly referenced or a tag property is set. This is to prevent raising an EInvalidXMPPacket exception for corrupted packets when all you want to do is read a few standard Exif tags.

XMPWritePolicy

```
type
  TXMPWritePolicy = (xwAlwaysUpdate, xwUpdateIfExists, xwRemove);

property XMPWritePolicy: TXMPWritePolicy read GetXMPWritePolicy
  write SetXMPWritePolicy;
```

Determines how the XMPPacket property is updated whenever a tag property is set.

xwAlwaysUpdate	If the new value is an empty string, then the XMP property is deleted, else it is changed (or added).
xwUpdateIfExists	Any existing property is updated, but if it doesn't already exist, no property is added. This setting is the default; change to xwAlwaysUpdate to mimic Windows Vista's behaviour.
xwRemove	Always remove an equivalent XMP value.

Methods

GetEnumerator

```
function GetEnumerator: TEnumerator;
```

Support function required by Delphi itself to enable the for/in syntax on a TCustomExifData instance —

```
procedure EnumSections(Data: TCustomExifData);
const
  Names: array[TExifSectionKind] of string = (
    'General', 'Details', 'Interop', 'GPS', 'Thumbnail');
var
  Section: TExifSection;
begin
  for Section in Data do
    ShowMessageFmt('%s section tag count = %d',
      [Names[Section.Kind], Section.Count]);
end;
```

Note that when enumerating, empty sections are skipped.

BeginUpdate, EndUpdate, Updating

```
procedure BeginUpdate;
procedure EndUpdate;
property Updating: Boolean read GetUpdating;
```

Like the methods of the same names in TStrings, calling these prevents the OnChange event being called unnecessarily..

Clear

```
procedure Clear(XMPPacketToo: Boolean = True); virtual;
```

Removes all tags. Remember to call SaveToJPEG (if using TExifData), UpdateFile (if using TExifDataPatcher) or SaveToFile/SaveToStream on the container object (if using TJPEGImageEx) to actually persist any changes.

HasMakerNote

```
function HasMakerNote: Boolean;
```

Returns whether there is maker note data, defined as a tag with the ID of ttMakerNote in the details section. (The fact that Sections[esMakerNote].Count returns 0 does not by itself say there is no maker note data — it may just be that the format is unrecognised.)

RemovePaddingTags

```
procedure RemovePaddingTags;
```

Removes any padding tags. Recent versions of Windows Explorer (amongst other Microsoft applications) always write out two fairly large padding tags by default, one each for the general and details section.

Rewrite

```
procedure Rewrite;
```

Assigns all tag properties against themselves, which will have the effect of filling out the XMPPacket property if XMPWritePolicy has been set to xwAlwaysUpdate.

SetAllDateTimeValues

```
procedure SetAllDateTimeValues(const DateTime: TDateTime);
```

Sets the DateTime, DateTimeOriginal and DateTimeDigitized properties all in one go.

ShutterSpeedInMSecs

```
function ShutterSpeedInMSecs: Extended;
```

Returns the value of the ShutterSpeedValue property (which is in APEX units) converted to milliseconds.

Events

OnChange

```
property OnChange: TNotifyEvent read FOnChange write FOnChange;
```

Called when a contained tag is changed, added or deleted.

TExifDataPatcher Class

Descending from TCustomExifData, TExifDataPatcher enables reading and some editing of a JPEG file's Exif segment — specifically, it allows any edits that do not cause repositioning tag data in the file, which in practice means that new tags cannot be added and existing ones cannot have their total data size increased (the first restriction here is because the TIFF, and so Exif standard requires tags to be written out sorted by their IDs). The benefit gained by these restrictions is that any sub-IFD not recognised by CCR Exif will not be corrupted when the data is saved out.

Properties

FileDateTime

```
property FileDateTime: TDateTime read GetFileDateTime
  write SetFileDateTime;
```

Reads/writes the 'last modified' time of the open file. If no file is currently open, an exception is raised.

FileName

```
property FileName: string read GetFileName write OpenFile;
```

Getter returns the filename of the currently-opened file, an empty string if no file is open; the setter is the OpenFile method (passing an empty string will just have CloseFile called). Remember to call UpdateFile (or CloseFile with its parameter set to True) to flush any changes to disk before opening a new file.

PreserveFileDate

```
property PreserveFileDate: Boolean read FPreserveFileDate
  write FPreserveFileDate default False;
```

Reads/writes whether calling UpdateFile does not update the open file's 'last modified' value.

Methods

GetImage

```
procedure GetImage(Dest: TJPEGImage);
```

Streams the image data for the active file into Dest.

GetThumbnail, HasThumbnail

```
procedure GetThumbnail(Dest: TJPEGImage);
function HasThumbnail: Boolean;
```

GetThumbnail streams the thumbnail image data for the active file (if any) into Dest. Call HasThumbnail to see if the active file has a thumbnail.

OpenFile

```
procedure OpenFile(const FileName: string);
```

Opens the specified JPEG file, loading its Exif segment data into the object. Note that a handle to the file is kept open until CloseFile is explicitly called or the TExifDataPatcher instance is destroyed; also, remember to call UpdateFile (or CloseFile with its parameter set to True) to flush any changes to disk before opening a new file.

UpdateFile

```
procedure UpdateFile;
```

Flushes any changes made to the open file to disk.

CloseFile

```
procedure CloseFile(SaveChanges: Boolean = False);
```

Closes the open file; if SaveChanges is True, the UpdateFile method is called first to flush any changes to disk, else they are lost.

TExifData Class

Descending from TCustomExifData and implementing IStreamPersist, TExifData (unlike TExifDataPatcher) allows free-form editing of Exif tags.

Properties

RemovePaddingTagsOnSave

```
property RemovePaddingTagsOnSave: Boolean read FRemovePaddingTagsOnSave
  write FRemovePaddingTagsOnSave default True;
```

Windows Explorer can write out rather large padding tags that serve no useful purpose; by default, TExifData thus removes them on save.

Sections

```
type
  TExifSectionKindEx = (esUserDefined, esGeneral, esDetails, esInterop,
    esGPS, esThumbnail);
  TExifSectionKind = esGeneral..esThumbnail;

property Sections[Section: TExifSectionKind]: TExtendableExifSection
  read GetSection; default;
```

Returns the specified section (TExtendableExifSection extends TExifSection with methods to add tags).

Thumbnail

```
property Thumbnail: TJPEGImage read GetThumbnail write SetThumbnail;
```

Reads/writes the Exif thumbnail image.

Methods

Assign

```
procedure Assign(Source: TPersistent); override;
```

If Source is neither nil nor another TCustomExifData instance, then the inherited implementation is called, raising an exception. Otherwise, if Source is nil, Clear is called, else the tag and thumbnail data of Source is copied across.

Clear

```
procedure Clear; overrides;
```

Overrides the inherited implementation to clear the Thumbnail property too.

CreateThumbnail

```
const
  StandardExifThumbnailWidth = 160;
  StandardExifThumbnailHeight = 120;

procedure CreateThumbnail(Source: TGraphic; ThumbnailWidth: Integer =
  StandardExifThumbnailWidth; ThumbnailHeight: Integer =
  StandardExifThumbnailHeight);
```

Sets Thumbnail to be a thumbnail image of Source.

LoadFromJPEG

```
function LoadFromJPEG(JPEGStream: TStream): Boolean; overload;

function LoadFromJPEG(JPEGImage: TJPEGImage): Boolean; overload;

function LoadFromJPEG(const FileName: string): Boolean; overload;
```

Inspects the specified JPEG image for Exif metadata; if found, the data is loaded and the function returns True, else the TExifData object is cleared and the function returns False.

LoadFromStream

```
procedure LoadFromStream(Stream: TStream);
```

Loads data from the stream. Note that unlike LoadFromJPEG, LoadFromStream expects just an Exif segment, not a complete JPEG file.

SaveToJPEG

```
type
  TSaveWhenEmptyBehaviour = (sbRemoveSegmentIfEmpty,
                             sbAllowEmptySegment);

procedure SaveToJPEG(const JPEGFileName: string; SaveWhenEmptyBehaviour:
  TSaveWhenEmptyBehaviour = sbRemoveSegmentIfEmpty); overload;

procedure SaveToJPEG(JPEGImage: TJPEGImage; SaveWhenEmptyBehaviour:
  TSaveWhenEmptyBehaviour = sbRemoveSegmentIfEmpty); overload;
```

Replaces any existing Exif segment in the specified JPEG image with one containing the currently-defined tags (if an Exif segment doesn’t already exist, one is added).

- The second parameter refers to when Empty returns True — should an Exif segment still be written out, or should any existing segment in the destination image simply be removed?
- Don’t be surprised if loading and immediately saving to the same file causes the latter to shrink a bit, since many cameras tend to write out a fair bit of ‘padding’ that will be ignored by TExifData.
- In the first variant of the method, the file must already exist and allow read/write access, otherwise an exception is raised. Furthermore, if an exception is raised during saving, the original file will be lost.

SaveToStream

```
procedure SaveToStream(Stream: TStream);
```

Writes the currently-loaded tags and thumbnail image to the stream. Note that unlike SaveToJPEG, SaveToStream literally just writes an Exif segment rather than replacing (or adding) an Exif segment to an existing JPEG image.

StandardizeThumbnail

```
 procedure StandardizeThumbnail;
```

If the existing thumbnail is larger than the standard Exif thumbnail size of 160 by 120 pixels, this method resizes it.

TJPEGImageEx Class

A simple extension of the VCL's TJPEGImage class, TJPEGImageEx adds an ExifData property. An alternative is to use a TExifData instance directly and its LoadFromJPEG/SaveToJPEG methods, since these have overloads that take an TJPEGImage instance as a parameter.

Properties

ExifData

```
property ExifData: TExifData read FExifData;
```

When the object is first created, Exif data will be empty. If creating a JPEG file from scratch, you can then add tags, upon which an Exif segment will be added to streamed-out files (see the Screenshooter demo for an example of this). Alternatively, calling the LoadFromStream method (whether directly or indirectly via the LoadFromFile method) will cause this property to be filled with whatever Exif data was in the file, which you can then edit and stream back out again by calling SaveToFile or SaveToSteam.

Methods

CreateThumbnail

```
const
  StandardExifThumbnailWidth = 160;
  StandardExifThumbnailHeight = 120;

procedure CreateThumbnail(ThumbnailWidth: Integer =
  StandardExifThumbnailWidth; ThumbnailHeight: Integer =
  StandardExifThumbnailHeight);
```

Sets Exif.Thumbnail to be a thumbnail of the current image.

LoadFromStream

```
procedure LoadFromStream(Stream: TStream); override;
```

Overrides the inherited implementation to ensure any Exif segment is loaded into the Exif property (if there isn't any Exif segment, then the Exif property's Clear method is called).

SaveToStream

```
procedure SaveToStream(Stream: TStream); override;
```

Overrides the inherited implementation to ensure the data stored in the Exif property is written out along with the actual image.

Other Types

Exception Classes

ECCRExifException

```
ECCRExifException = class(Exception);
```

Base exception class for exceptions directly raised by the CCR.Exif and CCR.Exif.JPEGUtils units.

EInvalidJPEGHeader

```
EInvalidJPEGHeader = class(ECCRExifException);
```

Raised explicitly by the JPEGHeader function if either there is no JPEG file header or the segment structure is malformed, and the OpenFile method of TExifDataPatcher when any streaming error occurs. Because the JPEGHeader function is used internally by various other things (specifically, the RemoveMetaDataFromJPEG global function, the TCustomExifData.LoadFromJPEG. and the TExifData.SaveToJPEG), you may find the exception raised when calling any of those too.

EExifDataPatcherError, ENoExifFileOpenError, EIllegalEditOfExifData

```
EExifDataPatcherError = class(EInvalidOperation);
```

```
ENoExifFileOpenError = class(EExifDataPatcherError);
```

```
EIllegalEditOfExifData = class(EExifDataPatcherError);
```

As the name of EExifDataPatcherError implies, these exceptions are raised by methods of TExifDataPatcher. ENoExifFileOpenError is raised when a file should be open but none is, and EIllegalEditOfExifData is raised if and when you attempt to add a new tag or expand an existing one.

EInvalidExifData

```
EInvalidExifData = class(ECCRExifException);
```

EInvalidExifData is raised by the LoadFromStream method of TCustomExifData when no valid Exif header is found.

EInvalidTiffData

```
EInvalidTiffData = class(Exception);
```

EInvalidTiffData is both a parent class (of ENotOnlyASCIIError and ETagAlreadyExists) and an exception that is raised directly by the LoadTIFFInfo function as appropriate. Since LoadTIFFInfo is used internally by TCustomExifData.LoadFromStream, which itself is called by TCustomExifData.LoadFromJPEG, you may find it being raised by those methods too.

ENotOnlyASCIIError

```
ENotOnlyASCIIError = class(EInvalidExifData);
```

ENotOnlyASCIIError is raised when you attempt to assign strings with one or more non-ASCII characters to a standard string tag, assuming the parent TCustomExifData instance has its EnforceASCII property set to True (which is the default).

ETagAlreadyExists

```
ETagAlreadyExists = class(EInvalidTiffData);
```

ETagAlreadyExists is raised if and when you attempt to add or rename a tag to have the same ID as another in its section, an operation that would create invalid TIFF, and thus, invalid Exif data.

Global Routines

CCR.Exif unit

ContainsOnlyASCII

```
function ContainsOnlyASCII(const S: UnicodeString): Boolean; overload;

function ContainsOnlyASCII(const S: RawByteString): Boolean; overload;
```

Returns True if the specified string is empty or only contains ASCII characters, False otherwise. For pre-Delphi 2009, UnicodeString is made an alias to WideString and RawByteString an alias to AnsiString.

CreateExifThumbnail

```
const
  StandardExifThumbnailWidth = 160;
  StandardExifThumbnailHeight = 120;

procedure CreateExifThumbnail(Source: TGraphic; Dest: TJPEGImage;
  MaxWidth: Integer = StandardExifThumbnailWidth;
  MaxHeight: Integer = StandardExifThumbnailHeight);
```

Makes Dest a thumbnail of Source.

DateTimeToExifString, TryExifStringToDateTime

```
function DateTimeToExifString(const DateTime: TDateTime): string;

function TryExifStringToDateTime(const S: string;
  var DateTime: TDateTime): Boolean; overload;
```

Does TDateTime ↔ Exif date/time string conversions (an Exif date/time string has the format YYYY:MM:DD). Since TCustomExifData uses these functions internally, you generally won't need to call them yourself, or indeed even care that date/times are stored a certain way in an Exif segment.

ProportionallyResizeExtents

```
function ProportionallyResizeExtents(const Width, Height: Integer;
  const MaxWidth, MaxHeight: Integer): TSize;
```

Returns Width and Height proportionally resized to fit MaxWidth by MaxHeight.

RemoveMetaDataFromJPEG

```
type
  TJPEGMetaDataKind = (mkExif, mkXMP);
  TJPEGMetaDataKinds = set of TJPEGMetadataKind;

const
  AllJPEGMetaDataKinds = [Low(TJPEGMetaDataKind)..
                          High(TJPEGMetaDataKind)];

function RemoveMetaDataFromJPEG(const JPEGFileName: string;
  Kinds: TJPEGMetaDataKinds = AllJPEGMetaDataKinds): TJPEGMetaDataKinds;
```

```
function RemoveMetaDataFromJPEG(JPEGImage: TJpegImage;
  Kinds: TJPEGMetaDataKinds = AllJPEGMetaDataKinds): TJPEGMetaDataKinds;
```

Removes any Exif and/or XMP segment from the specified JPEG image, returning True if there actually were any of the specified segments to delete.

StreamHasExifHeader

```
type
  THeaderCheckOption = (hcAlwaysRewindStream, hcMovePositionOnSuccess);

function StreamHasExifHeader(Stream: TStream;
  StreamPosAtEnd: THeaderCheckOption = hcAlwaysRewindStream): Boolean;
```

Returns True if the stream contains Exif data at the current position, False otherwise. The second parameter determines what happens to the stream when True is returned: should it be rewound to its original position, or should the latter be left to be immediately after the header?

StreamHasXMPSegmentHeader

```
type
  THeaderCheckOption = (hcAlwaysRewindStream, hcMovePositionOnSuccess);

function StreamHasXMPHeader(Stream: TStream;
  StreamPosAtEnd: THeaderCheckOption = hcAlwaysRewindStream): Boolean;
```

Returns True if the stream (assumed to be from a JPEG file segment) contains XMP data at the current position, False otherwise. The second parameter determines what happens to the stream when True is returned: should it be rewound to its original position, or should the latter be left to be immediately after the header?

CCR.Exif.JpegUtils unit

GetJpegDataSize

```
function GetJpegDataSize(Data: TStream): Int64; overload;

function GetJpegDataSize(Jpeg: TJPEGImage): Int64; overload;
```

Finds the total size of the JPEG image by looking for its EOI (jmEndOfImage) marker. In the first version, the stream’s original position is restored after the marker has been found.

JPEGHeader

```
type
  TJPEGMarker = Byte;
  TJPEGMarkers = set of TJPEGMarker;

const
  AllJPEGMarkers = [Low(TJPEGMarker)..High(TJPEGMarker)];

  jmJFIF = TJPEGMarker($E0);
  jmApp1 = TJPEGMarker($E1); //= the marker number Exif uses
  //refer to the source for other jmXXX values

type
```

```
    IJPEGSegment = interface
      property Data: TCustomMemoryStream read GetData;
      property MarkerNum: TJPEGMarker read GetMarkerNum;
    end;

    IFoundJPEGSegment = interface(IJPEGSegment)
      property Offset: Int64 read GetOffset;
      property OffsetOfData: Int64 read GetOffsetOfData;
      property TotalSize: Word read GetTotalSize;
    end;

    IJPEGHeaderParser = interface
      function GetCurrent: IFoundJPEGSegment;
      function GetEnumerator: IJPEGHeaderParser;
      function MoveNext: Boolean;
      property Current: IFoundJPEGSegment read GetCurrent;
    end;

  function JPEGHeader(JPEGStream: TStream; const MarkersToLookFor:
    TJPEGMarkers; StreamOwnership: TStreamOwnership = soReference):
    IJPEGHeaderParser; overload;

  function JPEGHeader(const JPEGFile: string; const MarkersToLookFor:
    TJPEGMarkers = AnyJPEGMarker): IJPEGHeaderParser; overload; inline;

  function JPEGHeader(Image: TJPEGImage; const MarkersToLookFor:
    TJPEGMarkers = AnyJPEGMarker): IJPEGHeaderParser; overload;
```

Parses a JPEG file's header, returning segment information in the form of IFoundJPEGSegment instances — use the MarkersToLookFor parameter to restrict what types of segment to return. Basically, you use JPEGHeader in the context of a for/in loop:

```
  var
    Segment: IFoundJPEGSegment;
  begin
    for Segment in JPEGHeader('filename.jpg') do
      //use Segment
```

WriteJPEGSegmentToStream

```
  procedure WriteJPEGSegmentToStream(Stream: TStream;
    MarkerNum: TJPEGMarker; const Data; DataSize: Word); overload;

  procedure WriteJPEGSegmentToStream(Stream: TStream;
    MarkerNum: TJPEGMarker; Data: TStream; DataSize: Word = 0); overload;
```

Writes out a JPEG header segment with the segment header in the proper big endian byte order. In the second variant, if DataSize is 0, the Data stream is rewound to the beginning and the whole of its contents written out; if DataSize is not 0, the data to be saved is understood to be from the stream's current position only.