DIGITAL OPTICS
PRECISION DIGITAL IMAGING

Storing V++ Attachments in a Standard TIFF File

Last update: 16 August 2007

This document describes TIFF extensions used by Digital Optics V++ to store image attachments and other proprietary data. The method used is fully compatible with other TIFF readers, even if they don't understand the extensions.

We assume that the reader already has a good understanding of the TIFF format. For more information, refer to the TIFF 6.0 Specification, now available on Adobe's web site, and to the TIFF Background paper available on the Digital Optics web site.

Attachment Tag

Proprietary data can be stored safely in a standard TIFF file, without affecting other TIFF readers, by the use of private tags. Digital Optics owns a registered block of private tags with numbers from 33821 to 33830 inclusive.

The tag used to incorporate attachment data is the AttachmentTag which is defined as follows:

AttachmentTag
Tag = 33825 (8421.H)
Type = UBYTE
Count = Total number of bytes stored
Offset = Offset to start of attachment data in file

To include V++ attachment data in your own TIFF files, you must allocate space for the data in the file (this can be at any location you choose, including at the end of the file) and add an AttachmentTag to the IFD to enable a compatible reader to find the data. The data itself must conform to the list structure described below.

IMPORTANT: It is not mandatory to include an attachment list in your TIFF files, nor is any particular item required if you do include a list.

Multi-frame TIFF Files

Image sequences are stored as multi-frame TIFF files and each frame has its own IFD. In this type of file, there should be only one attachment list but an AttachmentTag should be included in every frame's IFD. Since there is only one attachment list stored in the file, every instance of the AttachmentTag will have the same file offset.

Attachment List

All image attachments (flags, ROIs, settings and so on) are stored in a single list with a fixed header. The structure of each list item is different and depends on the requirements of the attachment concerned.

The attachments list starts with a fixed 12 byte header, made up of three 32-bit signed integers, as follows (in Pascal format):

```
Sign   : integer ; // signature number, must always be set to 33825
NBytes : integer ; // total size (includes all headers, equal to AttachmentTag.Count)
NItems : integer ; // number of attachments stored
```

Each attachment item in the list has its own header with the following structure:

```
GUID   : TGUID ;   // unique identifier for the attachment
Count  : integer ; // data size, in bytes, not including this header
```

The actual data block is stored immediately after the item header and its size is the byte count indicated in the Count field of the item header.

The GUID is a Windows identifier, 16 bytes long, that is unique for every attachment. This enables any version of V++ to scan through the list and locate attachments that it understands. If new attachments are introduced in future versions, they will simply be ignored by older versions that don't recognise them.
GUIDs are used throughout the Windows operating system and are usually written in a specific format, like this:

{5A4F8640-F5B2-11D1-B611-00AA00C0D2AA}

The GUID basically consists of a string of 16 bytes which are written in memory order.

You may define your own list items and identify them with a GUID you generate yourself. GUIDs are guaranteed to be globally unique so there would be no need to inform Digital Optics about this. However, V++ will only read attachments that it understands so we encourage you to get in touch with us about any proposals you may have for new attachments.

An attachment list containing 2 items would look like this:

Signature	*Always set to 33825*
NBytes	*Total size of entire list structure (in bytes)*
NItems	*Number of items in list (in this case, 2)*
Item 1 GUID	*Identifier for first list item*
Item 1 Count	*Size of data block for item 1 (in bytes)*
Item 1 Data	*Data block for item 1*
Item 2 GUID	*Identifier for second list item*
Item 2 Count	*Size of data block for item 2 (in bytes)*
Item 2 Data	*Data block for item 2*

Note that there is no limit to the size of an attachment list, or the number of items, provided it is structured correctly. To add more items, simply alter the NItems field and add the items to the end of the list. You will also need to adjust NBytes to correspond to the new overall list size.

To calculate the NBytes value in the main header, you must add up the Count values for all list items, add the item header size (20) multiplied by the number of items, and add the size of the fixed header (12).

Flags
Flags are location indicators and the attachment data consists of a simple list of coordinates.

The GUID identifier for flags is: {5A4F8640-F5B2-11D1-B611-00AA00C0D2AA}

A single flag coordinate is made up of four 32-bit integers, as follows:

```
x : integer ;  // x position
y : integer ;  // y position
z : integer ;  // z position (set to –1 if flag appears in every frame)
t : integer ;  // reserved (set to –1)
```

The data block for flags consists of a series of these (x,y,z,t) values. To work out how many flags there are, V++ divides the Count by the number of bytes required to store a single flag (16).

Note that the z and t values should be set to –1 by default. If you set any other z value then it associates the flag with a specific frame number in a sequence. The t value is ignored at the moment but should be set to –1 to ensure future compatibility.

There should be only one item header for flags in the attachments list and its data block must contain all of the flags defined on the image. If there are multiple flag items in the list then only the last one will be loaded.

Regions of Interest (ROIs)
ROIs are rectangular areas and the attachment data consists of a simple list of rectangle definitions.

The GUID identifier for ROIs is: {7FF2F720-F895-11D1-B611-00AA00C0D2AA}

Each ROI is stored as a Windows TRect structure, as follows:

```
Left   : integer ; // left border
Top    : integer ; // top border
Right  : integer ; // right border + 1
Bottom : integer ; // bottom border + 1
```

The data block for ROIs consists of a series of TRect structures. To work out how many ROIs there are, V++ simply divides the Count by the bytes required to store a single TRect (16).

IMPORTANT: V++ follows the Windows "+1" convention for storing rectangles, which means that the right and bottom boundaries are 1 unit beyond the actual inclusive boundary. Therefore the width of the rectangle can be calculated as Right – Left and its Height as Bottom – Top. This allows V++ ROI definitions to be directly compatible with Windows GDI and rectangle functions.

There should be only one item header for ROIs in the attachments list and its data block must contain all of the ROIs defined on the image. If there are multiple ROI items in the list then only the last one will be loaded.

Polygon ROI
An image can have a polygon definition attached which is used to outline a non-rectangular region of interest. There is only one polygon per image.

The GUID identifier for the Polygon ROI is: {B3B2DA90-62AE-4C28-9B9A-F1B2D60E4ACF}

The polygon is stored as a list of Windows TPoint structures, as follows:

```
X : integer ;
Y : integer ;
```

The data block for the polygon consists of a series of these (x,y) values. To work out how many vertices there are in the list, V++ divides the Count by the number of bytes required to store a single vertex (8).

There should be only one polygon item header in the attachments list and its data block contains all of the vertices. If there are multiple polygon items in the list then only the last one will be loaded.

Image Settings

The contrast, brightness and gamma settings you make in V++ don't change any pixel values but are preserved by saving them in the TIFF file as an image attachment.

The GUID identifier for Image Settings is: {5BBF00C0-5788-11D4-BD97-C05751C10000}

Image settings are stored as follows, in Pascal format:

```
Bright        : integer ;        // overall brightness
Contrast      : integer ;        // overall contrast
Gamma         : single ;         // overall gamma
RedBright     : integer ;        // red channel brightness adjust
GreenBright   : integer ;        // green channel brightness adjust
BlueBright    : integer ;        // blue channel brightness adjust
RedContrast   : integer ;        // red channel contrast adjust
GreenContrast : integer ;        // green channel contrast adjust
BlueContrast  : integer ;        // blue channel contrast adjust
RedGamma      : single ;         // red channel gamma adjust
GreenGamma    : single ;         // green channel gamma adjust
BlueGamma     : single ;         // blue channel gamma adjust
Mode          : TDisplayMode ;   // display mode structure (see comments below)
Interval      : integer ;        // movie frame interval (ms)
```

The Mode field is an undocumented TDisplayMode structure, 68 bytes long, which is used internally by V++ to keep track of display mode settings.

There should be only one item header for image settings in the attachments list and its data block must contain all of the fields shown above. If there are multiple image settings items in the attachments list then only the last one will be loaded.

If you wish to exploit the Image Settings attachment then please contact Digital Optics for more information.

Sample Attachment List

To include 3 flags at (10,10), (20,20) and (30,30) in a TIFF file, you would need to create an attachment list that looks like this:

	Stored Value	*Size (bytes)*
Signature:	33825	4
NBytes:	80	4
NItems:	1	4
GUID:	{5A4F8640-F5B2-11D1-B611-00AA00C0D2AA}	16
Count:	48	4
Data:	10,10,-1,-1,20,20,-1,-1,30,30,-1,-1	48

The actual size of each field is shown at the right. Note that the total of these sizes is 80 which is the value we store in NBytes in the main header.

To store this attachment list in the TIFF file, we need to create an AttachmentTag like this:

```
Tag    =  33825
Type   =  UBYTE
Count  =  80
Offset =  Actual file offset
```

The file offset will depend on how large the rest of the TIFF file is and where you decide to store the attachment list. Like all TIFF structures, the attachment list can go anywhere in the file after the initial 8 byte TIFF header. V++ usually puts it at the end of the file.

The AttachmentTag must be included in the TIFF file's IFD and it is therefore usually necessary to compute a variety of file offsets prior to actually writing the file. Remember that you should include an identical AttachmentTag in every IFD if you create a multi-frame TIFF file.

Contact

For more information, contact Digital Optics at the following address:

Digital Optics	*E-mail:*	info@digitaloptics.co.nz
PO Box 35-715	*Web:*	http://www.digitaloptics.co.nz
Browns Bay		
Auckland	*Phone:*	+64 (9) 478 5779
NEW ZEALAND	*Fax:*	+64 (9) 479 4750