IMAGING WORKBENCH

Software for Multichannel Dynamic Fluorescence Image Acquisition and Analysis

INDEC BioSystems
505 East Evelyn Ave.
Mountain View, CA 94041, USA
support@imagingworkbench.com

Imaging Workbench Application Note 11

ARF File Format

Imaging Workbench 5 uses the AXon Image (AXI) format as the native mode for storing multiple images, comments, protocol information and other information. An image in an AXI file can be exported as an Axon Raw Format (ARF) file for processing by user-generated software.

In earlier versions, Axon Imaging Workbench 2 exports images to Version 1 ARF files, with 1 image per file, and Axon Imaging Workbench 4 exports images to Version 2 ARF files, with multiple images per file.

An ARF file is made up of several sections as described below. At the end is a piece of C code that could be used to read a single (Version 1) ARF image into memory.

HEADER

Word 0 (2 bytes)	The byte ordering of the machine that wrote the file. If this evaluates to an integer 1 then your software is running on the same type of computer as was used to write the file. If not, the software should swap the high and low bytes in every word.
Word 1	The characters “AR”, indicating ‘Axon Raw’. If you don’t find this here then the file is not a true ARF file.
Word 2	The integer Version Number, either 1 or 2.
Word 3	The integer number of pixels in one row of the image.
Word 4	The integer number of pixels in one column of the image.
Word 5	The integer number of useable bits per pixel. The pixels are stored in the smallest integral type. This means that if a pixel occupies 8 or less bits per pixel, it is stored in a byte; if between 9 and 15, in 2 bytes; and if between 16 and 31, in 4 bytes (a double word).
Word 6	If the Version Number is 2, this contains the number of images in the file. Each image follows the preceding image with no unused bytes between. This parameter is not used in Version 1 ARF files.

COMMENTS

The next 512 bytes contains ‘Application Dependent’ data. Imaging Workbench 5 does not currently use this space, but it might in the future.

IMAGE

The image data is stored in this section. The origin is in the upper left of the image, and readout proceeds by row.

SAMPLE CODE FOR READING AN ARF FILE

```
// ReadARF - Reads ARF version 1
// 'Comments' and 'Image' are allocated here.
// 'Comments' is not allocated if the pointer that is passed in is valid.
// The image is *always* replaced, but no effort is made to clean up
// if the Image had previously been allocated.
// Make sure, if 'Comments' is preallocated, that it is big enough:
// maximum 513 bytes including any terminating null characters.

int FAR PASCAL ReadARF( LPSTR FileName, int *x, int *y, int *BitDepth, LPSTR
*Comments, BYTE huge **Image)
{
   UINT     fh;
   OFSTRUCT   of;
   BYTE Header[22];
   BYTE SwbHeader[22];
   int *iHeader;
   long BytesPerPixel;
   long BytesPerImage;

   fh = OpenFile (FileName, &of, OF_READ);

   if (fh == -1)
   {
      ErrorMessage(ERR_COULDNTREADFILE);
      return FALSE;
   }
   _lread (fh, (LPSTR)Header, 12);

      if ( ((int *)Header)[0] != 1 )    // Different-endian platform
   {
      swab(Header, SwbHeader, 12);
      iHeader = SwbHeader;
   }
   else
   {
      iHeader = (int *)Header;
   }

   if (Header[2] == 'A' && Header[3] == 'R' && iHeader[2] == 1)
                                    // Version 1
   {
      *x = iHeader[3];                  // Dimensions
      *y = iHeader[4];
      *BitDepth = iHeader[5];       // Useable bits per pixel
   }

   else
   {
      _lclose (fh);
      ErrorMessage(ERR_BADARFVERSION);
      return 0;
   }

   if (!*Comments)
      *Comments = GlobalAllocPtr(GPTR, 513);
                                    // Allow for a NULL terminating
                                    // character.
      _lread (fh, *Comments, 512);      // Application dependent information
      if ( *BitDepth <= 8)
      BytesPerPixel = 1;
   else if ( *BitDepth <= 16)
      BytesPerPixel = 2;
   else if ( *BitDepth <= 32)
      BytesPerPixel = 4;
```

```
    BytesPerImage = (LONG) *x * *y * BytesPerPixel;
    *Image = GlobalAllocPtr(GPTR, BytesPerImage);

    _hread (fh, *Image, BytesPerImage);  // Image

    if ( ((int *)Header)[0] != 1 )       // Different-endian platform
    {
       switch(BytesPerPixel)
       {
          case 1: break;                 // Nothing to do!
          case 2:
          {
             BYTE huge *swbImage;
             BYTE huge *tsrc;
             BYTE huge *tdst;
             int Total ;
             int Jump = 0x7FFE;          // Must be a positive even number

             tdst = swbImage = GlobalAllocPtr(GPTR, BytesPerImage);
                          tsrc = *Image;
                          // Number of Bytes to swap is an int, ie max ~32k, so
                          we need to do this in chunks

             for (Total = 0; (long)(Total * Jump) < BytesPerImage; Total ++)
             {
                swab( tsrc, tdst, Jump);
                tsrc += Jump;
                tdst += Jump;
             }
                                        // And the last chunk
             swab( tsrc, tdst, (int)((long)(Total * Jump) - BytesPerImage));
             GlobalFreePtr(*Image);
             *Image = swbImage;
          }
          break;
          case 4:
          {
                                        // Not implemented yet!!
                                        // Here we have to swap words
                                        // as well as bytes.
          }
          break;
       }
    }

    _lclose (fh);
    return TRUE;
}
```