INTRODUCTION

The success of the TGAɔ File Format for storing color images can be attributed to its ease of use, the small amount of program memory needed to parse the file, and the fact that it was the first true-color file format widely available. Truevision∀ defined the TGA file format in 1984 for use with its first videographics products. Since then, it has been estimated that today over 80 percent of the color images stored on hard drives employ some variation of the TGA file format. Many government offices, corporations, service bureaus, production shops and nearly all Truevision developers have standardized on the TGA format as a means of allowing cross-product and cross-application compatibility. Truevision recommends that this format be used by all software developed for Truevision products since it allows customers flexibility in combining many applications together to provide a total solution to meet their needs.

The original Truevision TGA File Format has been widely accepted by the graphics industry. However, newer technology and techniques have created the need for additional image information to be recorded in the file. In 1989, Truevision introduced extensions to the TGA File Format to satisfy requests made by the graphics industry and to ensure that the standard will meet future needs of the color imaging marketplace. The extensions are optional and will have no impact on existing packages (assuming the packages followed the original TGA File Format guidelines). In particular, the new TGA File Format addresses the following needs:

* The inclusion of a scaled-down "postage stamp" copy of the image
* Date and Time of image file creation
* Author Name
* Author Comments
* Job Name
* Job Accumulated Time
* Gamma Value
* Correct Color LUT
* Pixel Aspect Ratio
* Scan Line Offset Table
* Key Color
* Software Package Name and Version Number
* Developer Definable Areas
* Attribute (Alpha) channel Type
* The ability for simple expansion

DEFINITIONS

Throughout this document, we will be using the terms **Pseudo-Color**, **True-Color** and **Direct-Color**. These terms are defined as follows:

Pseudo-Color - Each pixel value is used as a single index into a programmable color map which contains the actual red, green and blue intensities to be displayed. The Truevision products that use this type of image are: VDA, VDA/D, TARGA∀ M8, ATVista∀, NuVista∀ and HRЭ videographics boards.

True-Color - Each pixel value is sub-divided into red, green and blue fields that directly determine the intensities of each primary color. The Truevision products that use this type of image are: ICB, TARGA 16, TARGA 24, TARGA 32, ATVista and NuVista videographics boards.

Direct-Color - Each pixel value is sub-divided into red, green and blue fields which are used as separate indices to access independent, programmable look-up tables. The outputs of the individual color maps directly determine the intensities of each primary color. A Direct-Color system is similar to Pseudo-Color except that the values in the color maps can be altered individually for the red, green and blue channels; whereas, the red, green and blue values in a Pseudo-Color system are loaded into one map which is accessed by a single index. Truevision products that use this type of image are: ATVista and NuVista videographics boards.

The TrueVista∀ (ATVista and NuVista) videographics cards can be programmed to accept images which are Pseudo-Color, True-Color and Direct-Color. When they are functioning in any of the Linked Modes, they are said to be acting as Pseudo-Color devices. When they are configured for any of the Independent Modes, they are said to be acting as Direct-Color devices. When bypassing the look-up tables altogether, they are said to be acting as True-Color devices.

The VDA, VDA/D, TARGA M8 and HR can only be used as Pseudo-Color devices.
The ICB, TARGA 16, 24 and 32 can only be used as True-Color devices.

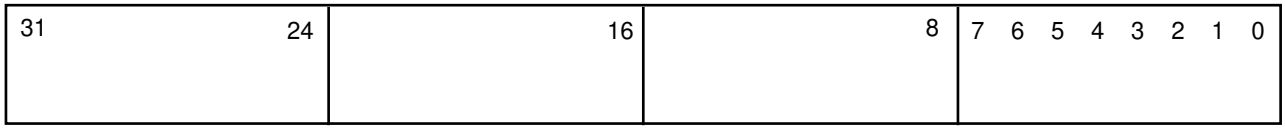
Long = 32 bit value

Short = 16 bit value

Byte = 8 bit value

ASCII = sequence of bytes conforming to the ASCII definition (Truevision recommends that the ASCII fields contain only printable ASCII characters, with exception of the null terminator, and that all formatting be performed by the application)

Bit Numbering (for diagrams in this document)



Byte Ordering

TGA files are stored using the Intel byte ordering convention (least significant byte first, most significant byte last). For this reason, applications running on Motorola-based systems will need to invert the ordering of bytes for short and long values after a file has been read.

Suffix and File Type Definitions

Since there is a great need to be able to locate files easily on mass storage devices, we have defined a filename suffix for DOS and UNIX operating systems, and a file type for the Macintosh environment. You should use this filename suffix or file type for all TGA image files.

1. DOS, UNIX and XENIX environments - Append to the end of the filename a three character suffix "TGA" after the "." indicator. Example: "Image.tga."

2. Macintosh environment - Use a file type of "TPIC".

However, the demonstration software and the programs in the Truevision software series currently use the suffixes, ".VDA", ".ICB", ".TGA", and ".VST", for VDA/D, ICB, TARGA and ATVista images. It is therefore suggested that applications should allow the user the capability of reading images with any of these extensions, but should only use the extension ".TGA" when writing images. As future versions of Truevision software products are made available, they will convert to using the ".TGA" and "TPIC" conventions exclusively.

TGA FILE FORMAT SPECIFICATION

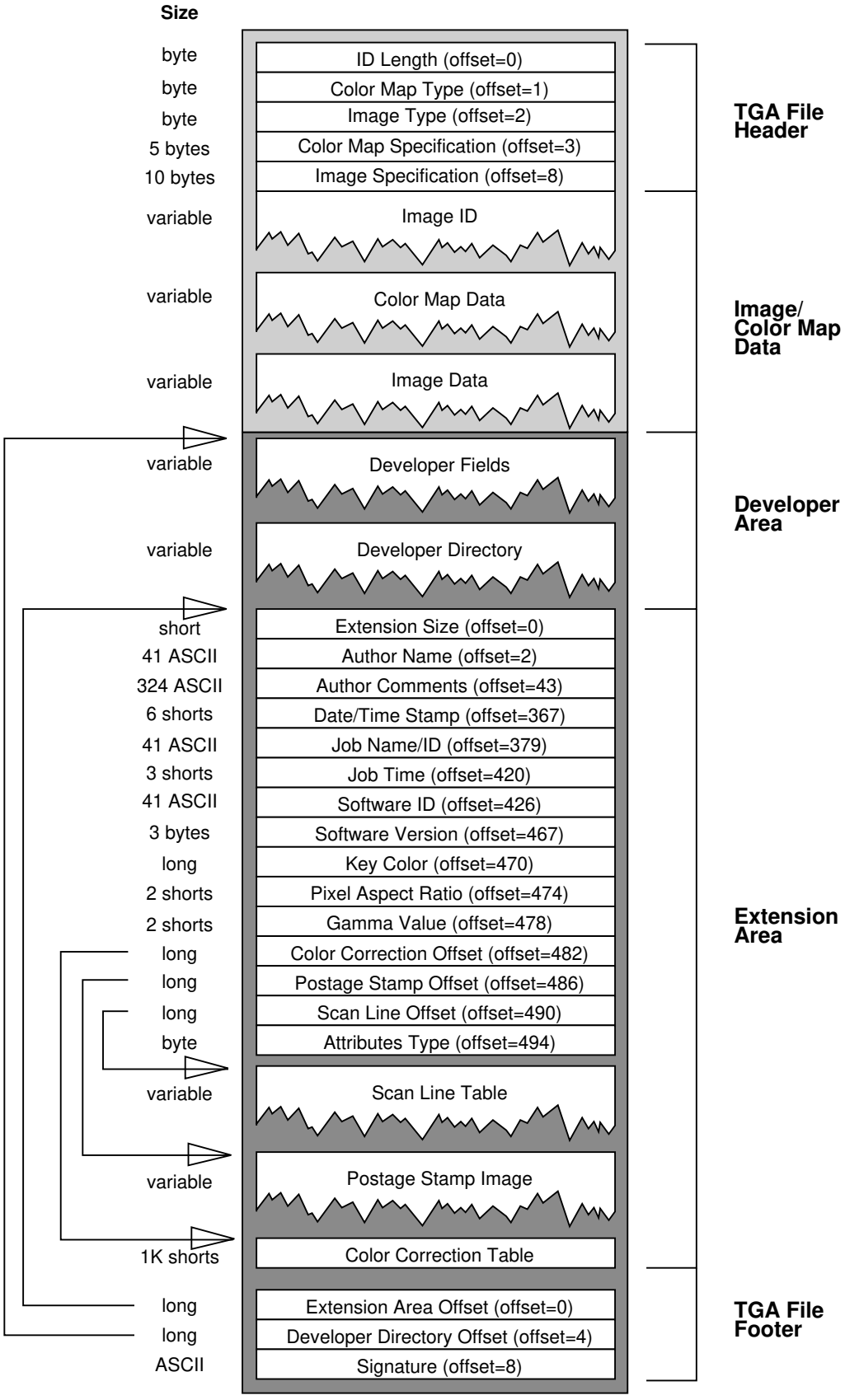


Figure 1 - TGA File Format

While Figure 1 demonstrates one possible arrangement for the various data blocks within a TGA File, other arrangements are possible since many of the data blocks are referenced by an offset position from the beginning of the file. In particular, developers may find it easier to create the file if the Scan Line Table, the Postage Stamp Image and the Color Correction Table are located before the Extension Size field (Field 10). The Truevision TGA File Format comprises 5 areas, each of which contains one or more fields of fixed or variable length. The 5 file areas are: (1) TGA File Header, (2) Image/ColorMap Data, (3) Developer Area, (4) Extension Area and (5) TGA File Footer.

The last 3 areas, the Developer Area, the Extension Area and the TGA File Footer are new to the file specification as of September, 1989. For this reason, images created with software written before September, 1989 will probably not contain these three fields. In this document, the TGA File format prior to September, 1989 will be referred to as the Original TGA Format while the current TGA File format will be referred to as the New TGA Format.

1. A TGA Reader should begin by determining whether the desired file is in the Original TGA Format or the New TGA Format. This is accomplished by examining the last 26 bytes of the file (most operating systems support some type of SEEK function). Reading the last 26 bytes from the file will either retrieve the last 26 bytes of image data (if the file is in the Original TGA Format), or it will retrieve the TGA File Footer (if the file is in the New TGA Format).

2. To determine whether the acquired data constitutes a legal TGA File Footer, scan bytes 8-23 of the footer as ASCII characters and determine whether they match the signature string:

   TRUEVISION-XFILE

   This string is exactly 16 bytes long and is formatted exactly as shown above (capital letters), with a hyphen between "TRUEVISION" and "XFILE." If the signature is detected, the file is assumed to be in the New TGA format and MAY, therefore, contain the Developer Area and/or the Extension Area fields. If the signature is not found, then the file is assumed to be in the Original TGA format and should only contain areas 1 and 2; therefore, the byte format for the TGA File Footer is defined as follows:

Bytes 0-3:	The Extension Area Offset
Bytes 4-7:	The Developer Directory Offset
Bytes 8-23:	The Signature
Byte 24:	ASCII Character "."
Byte 25:	Binary zero string terminator (0x00)

Note: DEVELOPERS ARE NOT REQUIRED TO READ, WRITE OR USE THE EXTENSION OR DEVELOPER AREAS; THEY ARE OPTIONAL. EVEN IF THESE AREAS ARE NOT USED, IT IS RECOMMENDED THAT A TGA FILE FOOTER STILL BE INCLUDED WITH THE FILE. Please see page 19 for more information about the TGA File Footer.

TGA FILE HEADER

ID Length - Field 1 (1 byte):

This field identifies the number of bytes contained in Field 6, the Image ID Field. The maximum number of characters is 255. A value of zero indicates that no Image ID field is included with the image.

Color Map Type - Field 2 (1 byte):

This field indicates the type of color map (if any) included with the image. There are currently 2 defined values for this field:

0 - indicates that no color-map data is included with this image.

1 - indicates that a color-map is included with this image.

The first 128 Color Map Type codes (Field 2) are reserved for use by Truevision, while the second set of 128 Color Map Type codes (128 to 255) may be used for developer applications.

True-Color images do not normally make use of the color map field, but some current applications store palette information or developer-defined information in this field. It is best to check Field 3, Image Type, to make sure you have a file which can use the data stored in the Color Map Field. Otherwise ignore the information. When saving or creating files for True-Color images do not use this field and set it to Zero to ensure compatibility. Please refer to the Developer Area specification for methods of storing developer defined information.

Image Type - Field 3 (1 byte):

The TGA File Format can be used to store Pseudo-Color, True-Color and Direct-Color images of various pixel depths. Truevision has currently defined seven image types:

Image Type	Description
0	No Image Data Included
1	Uncompressed, Color-mapped Image
2	Uncompressed, True-color Image
3	Uncompressed, Black-and-white Image
9	Run-length encoded, Color-mapped Image
10	Run-length encoded, True-color Image
11	Run-length encoded, Black-and-white Image

Table 1 - Image Types

Image Data Type codes 0 to 127 are reserved for use by Truevision for general applications. Image Data Type codes 128 to 255 may be used for developer applications.

For a complete description of these image-data types, see the IMAGE TYPES section later in this manual.

Color Map Specification - Field 4 (5 bytes):

This field and its sub-fields describe the color map (if any) used for the image. If the Color Map Type field is set to zero, indicating that no color map exists, then these 5 bytes should be set to zero. These bytes always must be written to the file.

Field 4.1 (2 bytes) - First Entry Index:

Index of the first color map entry. Index refers to the starting entry in loading the color map.

Example: If you would have 1024 entries in the entire color map but you only need to store 72 of those entries, this field allows you to start in the middle of the color-map (e.g., position 342).

Field 4.2 (2 bytes) - Color map Length:

Total number of color map entries included.

Field 4.3 (1 byte) - Color map Entry Size:

Establishes the number of bits per entry. Typically 15, 16, 24 or 32-bit values are used.

When working with VDA or VDA/D cards it is preferred that you select 16 bits (5 bits per primary with 1 bit to select interrupt control) and set the 16th bit to 0 so that the interrupt bit is disabled. Even if this field is set to 15 bits (5 bits per primary) you must still parse the color map data 16 bits at a time and ignore the 16th bit.

When working with a TARGA M8 card you would select 24 bits (8 bits per primary) since the color map is defined as 256 entries of 24 bit color values.

When working with a TrueVista card (ATVista or NuVista) you would select 24-bit (8 bits per primary) or 32-bit (8 bits per primary including Alpha channel) depending on your application's use of look-up tables. It is suggested that when working with 16-bit and 32-bit color images, you store them as True-Color images and do not use the color map field to store look-up tables. Please refer to the TGA Extensions for fields better suited to storing look-up table information.

Image Specification - Field 5 (10 bytes):

This field and its sub-fields describe the image screen location, size and pixel depth. These bytes are always written to the file.

Field 5.1 (2 bytes) - X-origin of Image:

These bytes specify the absolute horizontal coordinate for the lower left corner of the image as it is positioned on a display device having an origin at the lower left of the screen (e.g., the TARGA series).

Field 5.2 (2 bytes) - Y-origin of Image:

These bytes specify the absolute vertical coordinate for the lower left corner of the image as it is positioned on a display device having an origin at the lower left of the screen (e.g., the TARGA series).

Field 5.3 (2 bytes) - Image Width:

This field specifies the width of the image in pixels.

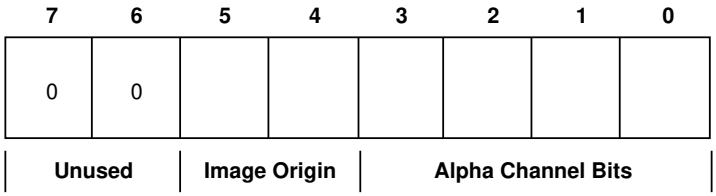
Field 5.4 (2 bytes) - Image Height:

This field specifies the height of the image in pixels.

Field 5.5 (1 byte) - Pixel Depth:

This field indicates the number of bits per pixel. This number includes the Attribute or Alpha channel bits. Common values are 8, 16, 24 and 32 but other pixel depths could be used.

Field 5.6 (1 byte) - Image Descriptor:



Bits 3-0: These bits specify the number of attribute bits per pixel. In the case of the TrueVista, these bits indicate the number of bits per pixel which are designated as Alpha Channel bits. For the ICB and TARGA products, these bits indicate the number of overlay bits available per pixel. See field 24 (Attributes Type) for more information.

Bits 5 & 4: These bits are used to indicate the order in which pixel data is transferred from the file to the screen. Bit 4 is for left-to-right ordering and bit 5 is for top-to-bottom ordering as shown below.

Screen destination of first pixel	Image Origin bit 5	bit 4
bottom left	0	0
bottom right	0	1
top left	1	0
top right	1	1

Table 2 - Image Origin

Bits 7 & 6: Must be zero to insure future compatibility.

IMAGE/COLOR MAP DATA

Image ID - Field 6 (variable):

This optional field contains identifying information about the image. The maximum length for this field is 255 bytes. Refer to Field 1 for the length of this field. If field 1 is set to Zero indicating that no Image ID exists then these bytes are not written to the file.

Color Map Data - Field 7 (variable):

If the Color Map Type (field 2) field is set to zero indicating that no Color-Map exists then this field will not be present (i.e., no bytes written to the file).

This variable-length field contains the actual color map information (LUT data). Field 4.3 specifies the width in bits of each color map entry while Field 4.2 specifies the number of color map entries in this field. These two fields together are used to determine the number of bytes contained in field 7.

Each color map entry is stored using an integral number of bytes. The RGB specification for each color map entry is stored in successive bit-fields in the multi-byte entries. Each color bit-field is assumed to be MIN(Field4.3/3, 8) bits in length. If Field 4.3 contains 24, then each color specification is 8 bits in length; if Field 4.3 contains 32, then each color specification is also 8 bits (32/3 gives 10, but 8 is smaller). Unused bit(s) in the multi-byte entries are assumed to specify attribute bits. The attribute bit field is often called the Alpha Channel, Overlay Bit(s) or Interrupt Bit(s).

For the TARGA M-8, ATVista and NuVista, the number of bits in a color map specification is 24 (or 32). The red, green, and blue components are each represented by one byte.

Image Data - Field 8 (variable):

This field contains (Width)x(Height) pixels. Each pixel specifies image data in one of the following formats: a single color-map index for Pseudo-Color; Attribute, Red, Green and Blue ordered data for True-Color; and independent color-map indices for Direct-Color. The values for Width and Height are specified in Fields 5.3 and 5.4 respectively.

The number of attribute and color-definition bits for each pixel are defined in Fields 5.6 and 5.5, respectively. Each pixel is stored as an integral number of bytes.

DEVELOPER AREA

Developer Data - Field 9 (variable):

Truevision created the Developer Area to satisfy the needs of developers who have already created similar areas on their own. We have made an attempt to remain upwardly compatible with these developers' methods.

The Developer Area is an area which may be used to store any type of information, although it is recommended that it be used only for application specific information, that is, information which would not be applicable to other products, and which is not already covered by the TGA File Format.

The size and format of the Developer Fields is totally up to the developer. Readers of files containing these fields should ignore the fields unless they have an understanding of their organization and content.

Since a file may contain more than one Developer Field, a Developer Directory was created which contains a "map" to the fields included in the Developer Area. The Developer Directory is located by using the offset pointer contained in bytes 4-7 of the TGA File Footer. The contents of this field are a byte offset from the beginning of the file to the start of the Developer Directory. If the Offset is ZERO (binary zero) no directory and no Developer Area fields exist.

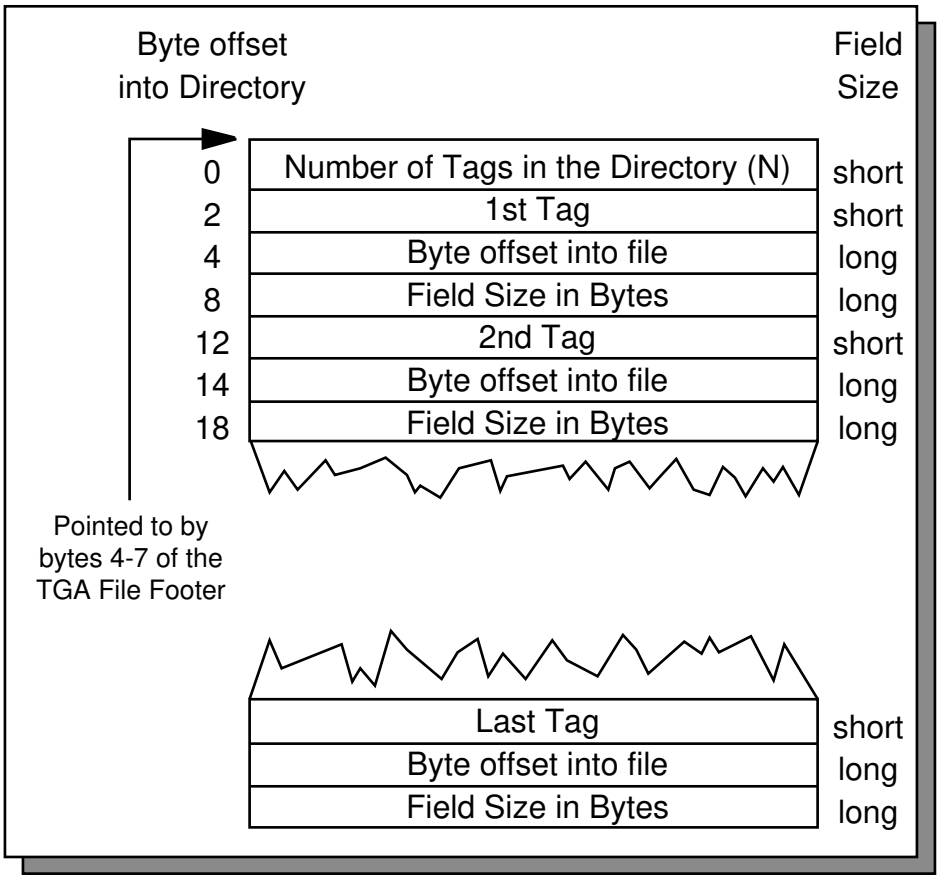


Figure 2 - Developer Directory

The first SHORT in the directory specifies the number of tags which are currently in the directory. The rest of the directory contains a series of TAG, OFFSET and SIZE combinations. Each TAG is a SHORT value in the range of 0 to 65535. Values from 0 - 32767 are available for developer use, while values from 32768 - 65535 are reserved for Truevision. Truevision will maintain a list of tags assigned to companies. To get a tag assignment for a product, simply contact Truevision Developer Services to request the next assignable tag. If you wish to supply Truevision with the data format of your field, we will make this information available to other developers who may wish to read your data (this is at your request only).

Following each TAG is an OFFSET. This OFFSET is a LONG value which specifies the number of bytes from the beginning of the file to the start of the field referenced by the tag.

Following the OFFSET is a FIELD SIZE. The FIELD SIZE is a LONG value which specifies the number of bytes in the field. This is useful for the allocation of buffer space for reading the field and for determining when the end of the field has been reached. There is no termination character to indicate the end of a field's data.

Each TAG, OFFSET, SIZE set references a particular field of data in the Developer Area. The TAGS may appear in any order in the directory (i.e., they do not need to be sorted). Since each set is 10 bytes in size (1 short, 2 longs), the total size of the directory will be:

(NUMBER_OF_TAGS_IN_THE_DIRECTORY * 10) + 2

bytes in size. The "+ 2" includes the 2 bytes for the SHORT value specifying the number of tags in the directory.

Although the size and format of the actual Developer Area fields are totally up to the developer, please define your formats to address future considerations you might have concerning your fields. This means that if you anticipate changing a field, build flexibility into the format to make these changes easy on other developers. Major changes to an existing TAG's definition should never happen. If major changes are required, we would prefer that you request another TAG number so as not to confuse programs which may honor your existing tag. Additionally, the developer should take care that the field is COMPLETE. That is, if there are any variable sized sub-fields, they should be INSIDE the field and the SIZE indicated in TAG, OFFSET and SIZE should contain the complete size. In this way, programs which may not understand a particular TAG and associated field can still preserve the data (i.e., pass it on in the file) without worry that data pointed to by a potential offset was not copied.

EXTENSION AREA

The Extension Area was created to satisfy requests from developers for additional file information. Every attempt has been made to keep the format of the Extension Area simple yet flexible enough to satisfy most developers' needs.

The Extension Area is pointed to by the Extension Area Offset in the TGA File Footer. If the Offset is ZERO (binary zero) then no Extension Area exists. Otherwise, this value is the offset from the beginning of the file to the beginning of the Extension Area.

The combination of the Developer Area and the Extension Area should serve Truevision and its developer community well into the future. However, should technology dictate additional changes to the specification, they will be quite easy to implement. The first field of the Extension Area is the "Extension Size" field. This field currently contains the value 495, indicating that there are 495 bytes in the fixed-length portion of the Extension Area. If future development warrants additional fields, then this value may change to indicate that new fields have been added. Any change in the number of bytes in the Extension Area will be made by Truevision and will be relayed to the Developer community via a revised TGA File Format Specification. TGA readers should only parse as much as they understand (i.e., 495 bytes for this first release). If they encounter additional bytes (as specified by the "Extension Size") they should not preserve them if rebuilding the file, since they do not know the nature of the extra bytes.

Extension Size - Field 10 (2 Bytes):

Bytes 0-1 - This field is a SHORT field which specifies the number of BYTES in the fixed-length portion of the Extension Area. For Version 2.0 of the TGA File Format, this number should be set to 495. If the number found in this field is not 495, then the file will be assumed to be of a version other than 2.0. If it ever becomes necessary to alter this number, the change will be controlled by Truevision, and will be accompanied by a revision to the TGA File Format with an accompanying change in the version number.

Author Name - Field 11 (41 Bytes):

Bytes 2-42 - This field is an ASCII field of 41 bytes where the last byte must be a null (binary zero). This gives a total of 40 ASCII characters for the name. If the field is used, it should contain the name of the person who created the image(author). If the field is not used, you may fill it with nulls or a series of blanks (spaces) terminated by a null. The 41st byte must always be a null.

Author Comments - Field 12 (324 Bytes):

Bytes 43-366 - This is an ASCII field consisting of 324 bytes which are organized as four lines of 80 characters, each followed by a null terminator. This field is provided, in addition to the original IMAGE ID field (in the original TGA format), because it was determined that a few developers had used the IMAGE ID field for their own purposes. This field gives the developer four lines of 80 characters each, to use as

an Author Comment area. Each line is fixed to 81 bytes which makes access to the four lines easy. Each line must be terminated by a null. If you do not use all 80 available characters in the line, place the null after the last character and blank or null fill the rest of the line. The 81st byte of each of the four lines must be null.

Date/Time Stamp - Field 13 (12 Bytes):

Bytes 367-378 - This field contains a series of 6 SHORT values which define the integer value for the date and time that the image was saved. This data is formatted as follows:

SHORT 0: Month (1 - 12)
SHORT 1: Day (1 - 31)
SHORT 2: Year (4 digit, ie. 1989)
SHORT 3: Hour (0 - 23)
SHORT 4: Minute (0 - 59)
SHORT 5: Second (0 - 59)

Even though operating systems typically time- and date-stamp files, this feature is provided because the operating system may change the time and date stamp if the file is copied. By using this area, you are guaranteed an unmodified region for date and time recording. If the fields are not used, you should fill them with binary zeros (0).

Job Name/ID - Field 14 (41 Bytes):

Bytes 379-419 - This field is an ASCII field of 41 bytes where the last byte must be a binary zero. This gives a total of 40 ASCII characters for the job name or the ID. If the field is used, it should contain a name or id tag which refers to the job with which the image was associated. This allows production companies (and others) to tie images with jobs by using this field as a job name (i.e., CITY BANK) or job id number (i.e., CITY023). If the field is not used, you may fill it with a null terminated series of blanks (spaces) or nulls. In any case, the 41st byte must be a null.

Job Time - Field 15 (6 Bytes):

Bytes 420-425 - This field contains a series of 3 SHORT values which define the integer value for the job elapsed time when the image was saved. This data is formatted as follows:

SHORT 0: Hours (0 - 65535)
SHORT 1: Minutes (0 - 59)
SHORT 2: Seconds (0 - 59)

The purpose of this field is to allow production houses (and others) to keep a running total of the amount of time invested in a particular image. This may be useful for billing, costing, and time estimating. If the fields are not used, you should fill them with binary zeros (0).

Software ID - Field 16 (41 Bytes):

Bytes 426-466 - This field is an ASCII field of 41 bytes where the last byte must be a binary zero (null). This gives a total of 40 ASCII characters for the Software ID. The purpose of this field is to allow software to determine and record with what program a particular image was created. If the field is not used, you may fill it with a null terminated series of blanks (spaces) or nulls. The 41st byte must always be a null.

Software Version - Field 17 (3 Bytes):

Bytes 467-469 - This field consists of two sub-fields, a SHORT and an ASCII BYTE. The purpose of this field is to define the version of software defined by the "Software ID" field above. The SHORT contains the version number as a binary integer times 100. Therefore, software version 4.17 would be the integer value 417. This allows for two decimal positions of sub-version. The ASCII BYTE supports developers who also tag a release letter to the end. For example, if the version number is 1.17b, then the SHORT would contain 117. and the ASCII BYTE would contain "b". The organization is as follows:

SHORT	(Bytes 0 - 1):	Version Number * 100
BYTE	(Byte 2):	Version Letter

If you do not use this field, set the SHORT to binary zero, and the BYTE to a space (" ").

Key Color - Field 18 (4 Bytes):

Bytes 470-473 - This field contains a long value which is the key color in effect at the time the image is saved. The format is in A:R:G:B where 'A' (most significant byte) is the alpha channel key color (if you don't have an alpha channel in your application, keep this byte zero [0]).

The Key Color can be thought of as the 'background color' or 'transparent color'. This is the color of the 'non image' area of the screen, and the same color that the screen would be cleared to if erased in the application. If you don't use this field, set it to all zeros (0). Setting the field to all zeros is the same as selecting a key color of black.

A good example of a key color is the 'transparent color' used in TIPS ɜ for WINDOW loading/saving.

Pixel Aspect Ratio - Field 19 (4 Bytes):

Bytes 474-477 - This field contains two SHORT sub-fields, which when taken together specify a pixel size ratio. The format is as follows:

SHORT 0: Pixel Ratio Numerator (pixel width)
SHORT 1: Pixel Ratio Denominator (pixel height)

These sub-fields may be used to determine the aspect ratio of a pixel. This is useful when it is important to preserve the proper aspect ratio of the saved image. If the two values are set to the same non-zero value, then the image is composed of square pixels. A zero in the second sub-field (denominator) indicates that no pixel aspect ratio is specified.

Gamma Value - Field 20 (4 Bytes):

Bytes 478-481 - This field contains two SHORT sub-fields, which when taken together in a ratio, provide a fractional gamma value. The format is as follows:

SHORT 0: Gamma Numerator
SHORT 1: Gamma Denominator

The resulting value should be in the range of 0.0 to 10.0, with only one decimal place of precision necessary. An uncorrected image (an image with no gamma) should have the value 1.0 as the result. This may be accomplished by placing the same, non-zero values in both positions (i.e., 1/1). If you decide to totally ignore this field, please set the denominator (the second SHORT) to the value zero. This will indicate that the Gamma Value field is not being used.

Color Correction Offset - Field 21 (4 Bytes):

Bytes 482-485 - This field is a 4-byte field containing a single offset value. This is an offset from the beginning of the file to the start of the Color Correction table. This table may be written anywhere between the end of the Image Data field (field 8) and the start of the TGA File Footer. If the image has no Color Correction Table or if the Gamma Value setting is sufficient, set this value to zero and do not write a Correction Table anywhere.

Postage Stamp Offset - Field 22 (4 Bytes):

Bytes 486-489 - This field is a 4-byte field containing a single offset value. This is an offset from the beginning of the file to the start of the Postage Stamp Image. The Postage Stamp Image must be written after Field 25 (Scan Line Table) but before the start of the TGA File Footer. If no postage stamp is stored, set this field to the value zero (0).

Scan Line Offset - Field 23 (4 Bytes):

Bytes 490-493 - This field is a 4-byte field containing a single offset value. This is an offset from the beginning of the file to the start of the Scan Line Table.

Attributes Type - Field 24 (1 Byte):

Byte 494 - This single byte field contains a value which specifies the type of Alpha channel data contained in the file.

Value	Meaning
0:	no Alpha data included (bits 3-0 of field 5.6 should also be set to zero)
1:	undefined data in the Alpha field, can be ignored
2:	undefined data in the Alpha field, but should be retained
3:	useful Alpha channel data is present
4:	pre-multiplied Alpha (see description below)
5 -127:	RESERVED
128-255:	Un-assigned

Pre-multiplied Alpha Example: Suppose the Alpha channel data is being used to specify the opacity of each pixel (for use when the image is overlayed on another image), where 0 indicates that the pixel is completely transparent and a value of 1 indicates that the pixel is completely opaque (assume all component values have been normalized). A quadruple (a, r, g, b) of (0.5, 1, 0, 0) would indicate that the pixel is pure red with a transparency of one-half. For numerous reasons (including image compositing) is is better to pre-multiply the individual color components with the value in the Alpha channel. A pre-multiplication of the above would produce a quadruple (0.5, 0.5, 0, 0).

A value of 3 in the Attributes Type Field (field 23) would indicate that the color components of the pixel have already been scaled by the value in the Alpha channel. For more information concerning pre-multiplied values, refer to the 1984 SIGGRAPH Conference Proceedings.

Porter, T., T. Duff, "Compositing Digital Images," SIGGRAPH '84 Conference Proceedings, vol. 18, no. 3, p. 254 , ACM, July 1984.

Scan Line Table - Field 25 (Variable):

This information is provided, at the developers' request, for two purposes:

1) To make random access of compressed images easy.

2) To allow "giant picture" access in smaller "chunks".

This table should contain a series of 4-byte offsets. Each offset you write should point to the start of the next scan line, in the order that the image was saved (i.e., top down or bottom up). The offset should be from the start of the file. Therefore, you will have a four byte value for each scan line in your image. This means that if your image is 768 pixels tall, you will have 768, 4-byte offset pointers (for a total of 3072 bytes). This size is not extreme, and thus this table can be built and maintained in memory, and then written out at the proper time.

Postage Stamp Image - Field 26 (Variable):

The Postage Stamp area is a smaller representation of the original image. This is useful for "browsing" a collection of image files. If your application can deal with a postage stamp image, it is recommended that you create one using sub-sampling techniques to create the best representation possible. The postage stamp image must be stored in the same format as the normal image specified in the file, but without any compression. The first byte of the postage stamp image specifies the X size of the stamp in pixels, the second byte of the stamp image specifies the Y size of the stamp in pixels. Truevision does not recommend stamps larger than 64 x 64 pixels, and suggests that any stamps stored larger be clipped. Obviously, the storage of the postage stamp relies heavily on the storage of the image. The two images are stored using the same format under the assumption that if you can read the image, then you can read the postage stamp. If the original image is color mapped, DO NOT average the postage stamp, as you will create new colors not in your map.

Color Correction Table - Field 27 (2K Bytes):

The Color Correction Table is a block of 256 x 4 SHORT values, where each set of four contiguous values are the desired A:R:G:B correction for that entry. This allows the user to store a correction table for image remapping or LUT driving. Since each color in the block is a SHORT, the maximum value for a color gun (i.e., A, R, G, B) is 65535, and the minimum value is zero. Therefore, BLACK maps to 0, 0, 0, 0 and WHITE maps to 65535, 65535, 65535, 65535.

TGA FILE FOOTER

1. A TGA Reader should begin by determining whether the desired file is in the Original TGA Format or the New TGA Format. This is accomplished by examining the last 26 bytes of the file (most operating systems support some type of SEEK function). Reading the last 26 bytes from the file will either retrieve the last 26 bytes of image data (if the file is in the Original TGA Format), or it will retrieve the TGA File Footer (if the file is in the New TGA Format).

2. To determine whether the acquired data constitutes a legal TGA File Footer, scan bytes 8-23 of the footer as ASCII characters and determine whether they match the signature string:

TRUEVISION-XFILE

This string is exactly 16 bytes long and is formatted exactly as shown above (capital letters), with a hyphen between "TRUEVISION" and "XFILE." If the signature is detected, the file is assumed to be in the New TGA format and MAY, therefore, contain the Developer Area and/or the Extension Area fields. If the signature is not found, then the file is assumed to be in the Original TGA format and should only contain areas 1 and 2; therefore, the byte format for the TGA File Footer is defined as follows:

Bytes 0-3:	The Extension Area Offset
Bytes 4-7:	The Developer Directory Offset
Bytes 8-23:	The Signature
Byte 24:	ASCII Character "."
Byte 25:	Binary zero string terminator (0x00)

Note: DEVELOPERS ARE NOT REQUIRED TO READ, WRITE OR USE THE EXTENSION OR DEVELOPER AREAS; THEY ARE OPTIONAL. EVEN IF THESE AREAS ARE NOT USED, IT IS RECOMMENDED THAT A TGA FILE FOOTER STILL BE INCLUDED WITH THE FILE. Please see page 19 for more information about the TGA File Footer.

Byte 0-3 - Extension Area Offset - Field 28

The first four bytes (bytes 0-3, the first LONG) of the TGA File Footer contain an offset from the beginning of the file to the start of the Extension Area. Simply SEEK to this location to position to the start of the Extension Area. If the Extension Area Offset is zero, no Extension Area exists in the file.

Byte 4-7 - Developer Directory Offset - Field 29

The next four bytes (bytes 4-7, the second LONG) contain an offset from the beginning of the file to the start of the Developer Directory. If the Developer Directory Offset is zero, then the Developer Area does not exist.

Byte 8-23 - Signature - Field 30

This string is exactly 16 bytes long and is formatted exactly as shown below capital letters), with a hyphen between "TRUEVISION" and "XFILE." If the signature is detected, the file is assumed to be of the New TGA format and MAY, therefore, contain the Developer Area and/or the Extension Area fields. If the signature is not found, then the file is assumed to be in the Original TGA format.

TRUEVISION-XFILE

Byte 24 - Reserved Character - Field 31

Byte 24 is an ASCII character "." (period). This character MUST BE a period or the file is not considered a proper TGA file.

Byte 25 - Binary Zero String Terminator - Field 32

Byte 25 is a binary zero which acts as a final terminator and allows the entire TGA File Footer to be read and utilized as a "C" string.

IMAGE TYPES

DATA TYPE 1 - COLOR-MAPPED IMAGES

Application:

This data type is used for storing color-mapped images (e.g., VDA/D, TARGA M8 or Color-Mapped TrueVista images). Images stored in this format can be retrieved and displayed rapidly; however, a full-screen (256 x 200) VDA image requires 51K bytes of storage. This format is desirable where storage and display time are critical but where file size is not.

This file format also provides a compact way to store limited-color True-Color TARGA images. Software can easily display color-mapped images in real-color display modes by mapping pixels when it copies them to the display memory.

File Format Requirements:

Field 2 - Color-Map Type (1 Byte):

This value MUST be 1 (0x01) for this type.

Field 3 - Image Type (1 Byte):

This value is 1 (0x01) for this data type.

DATA TYPE 2 - TRUE-COLOR IMAGES

Application:

This data type is used for storing images where each pixel is represented by its red, green, and blue values (e.g. ICB, TARGA 16, TARGA 24, TARGA 32, and TrueVISTA images). This format is useful where storage and display time are critical and where file size is not.

File Format Requirements:

Field 2 - Color-Map Type (1 Byte):

This value MUST be 0 (0x00). TIPS∀ from Truevision sets this value to 1 and stores a color pallet in this area. It is wise to check the image type to know when a color-map is needed, otherwise ignore the information. When new releases of Truevision software products are made available they will meet the TGA standard file format and no longer put pallet information in the color-map area.

Field 3 - Image Type (1 Byte):

This value must be 2 (0x02) for this data type.

DATA TYPE 3 - BLACK AND WHITE (UNMAPPED) IMAGES

Application:

This data type is used for storing raster images where each pixel can be represented by a grey-scale value (e.g., the TARGA 8, TARGA M8 and some TrueVista modes).

This file format is used by the demonstration programs distributed with the TARGA 8.

File Format Requirements:

Field 2 - Color-Map Type (1 Byte):

This value is always Zero (0x00) for unmapped images.

Field 3 - Image Type (1 Byte):

This value is 3 (0x03) for this data type.

DATA TYPE 9 - RUN-LENGTH ENCODED (RLE), COLOR-MAPPED IMAGES

Application:

This data type is used for storing images where each pixel is represented by a color map index. The information is encoded so that successive pixels with the same color value are run-length encoded. This format is designed for use with computer-generated (as opposed to captured) images which have large contiguous sections containing the same colors.

File Format Requirements:

Field 2 - Color Map Type (1 Byte):

This value is always 1 (0x01) for color-mapped images.

Field 3 - Image Type (1 Byte):

This value is 9 (0x09) for this data type.

DATA TYPE 10 - RUN-LENGTH ENCODED (RLE), TRUE-COLOR IMAGES

Application:

This data type is used for storing raster images where each pixel is represented by its red, green, and blue components. The infromation is encoded so that successive pixels with the same color value are run-length encoded.

Use this format for storing ICB, TARGA (16, 24, and 32), and TrueVISTA (16-bit RGB and 32-bit RGB modes) images. This type was primarily designed for computer generated (as opposed to captured) images which have large sections containing the same colors.

File Format Requirements:

Field 2 - Color Map Type (1 Byte):

This value is always Zero (0x00) for True-Color images.

Field 3 - Image Type Code (1 Byte):

This value is 10 (0x0A) for this data type.

DATA TYPE 11 - RUN-LENGTH ENCODED (RLE), BLACK AND WHITE IMAGES

Application:

This data type is used for storing raster images where each pixel is represented by a grey level. The information is encoded so that successive pixels with the same color value are run-length encoded.

This format is used for storing black and white TARGA 8 and TARGA M8 images and was primarily designed for use with computer-generated (as opposed to captured) images which have large sections containing the same gray values.

File Format Requirements:

Field 2 Color Map Type (1 Byte):

This value is always Zero (0x00) for True-Color images.

Field 3 Image Type Code (1 Byte):

This value is 11 (0x0B) for this data type.

RUN-LENGTH ENCODING OF IMAGES

Some of the Image Types described above are stored using a compression algorithm called Run-length Encoding. This type of encoding is used with file types 9 (Run-length Encoded Color-mapped Images), 10 (Run-length Encoded True-Color Images) and 11 (Run-length Encoded Black-and-white Images).

Run-length encoding takes advantage of the fact that many types of images have large sections in which the pixel values are all the same. This situation occurs more often in graphic images as opposed to captured, video images. In those images where large areas contain pixels of the same value, run-length encoding can greatly reduce the size of the stored image.

Run-length encoded (RLE) images comprise two types of data elements: Run-length Packets and Raw Packets.

The first field (1 byte) of each packet is called the Repetition Count field. The second field is called the Pixel Value field. For Run-length Packets, the Pixel Value field contains a single pixel value. For Raw Packets, the field is a variable number of pixel values.

The highest order bit of the Repetition Count indicates whether the packet is a Raw Packet or a Run-length Packet. If bit 7 of the Repetition Count is set to 1, then the packet is a Run-length Packet. If bit 7 is set to zero, then the packet is a Raw Packet.

The lower 7 bits of the Repetition Count specify how many pixel values are represented by the packet. In the case of a Run-length packet, this count indicates how many successive pixels have the pixel value specified by the Pixel Value field. For Raw Packets, the Repetition Count specifies how many pixel values are actually contained in the next field. This 7 bit value is actually encoded as 1 less than the number of pixels in the packet (a value of 0 implies 1 pixel while a value of 0x7F implies 128 pixels).

Run-length Packets should never encode pixels from more than one scan line. Even if the end of one scan line and the beginning of the next contain pixels of the same value, the two should be encoded as separate packets. In other words, Run-length Packets should not wrap from one line to another. This scheme allows software to create and use a scan line table for rapid, random access of individual lines. Scan line tables are discussed in further detail in the Extension Area section of this document.

As an example of the difference between the two packet types, consider a section of a single scan line with 128, 24-bit (3 byte) pixels all with the same value (color). The Raw Packet would require 1 byte for the Repetition Count and 128 pixels values each being 3 bytes long (384 bytes). The total number of bytes required to specify the chosen data using the Raw Packet is, therefore, 385 bytes. The Run-length Packet would require 1 byte for the Repetition Count and a single, 3-byte pixel value. The total number of bytes required to specify the chosen data using the Run-length Packet is, therefore, just 4 bytes!

Run-Length Packet:

Run-length packets are composed of two parts. The first is a repetition count and the second is the pixel value to repeat.

Field Name	Packet Type MUST BE 1 for run-length	Pixel Count (number of pixels encoded by this packet -1)	Pixel Data The common pixel value to be used
Field Size	1 bit	7 bits	Pixel Depth (field 5.5)

Table 3 - Run-length Packet

1. Pixel Count: The pixel count can range from 0 to 127. Since a run-length packet never encodes zero pixels, and in order to make use of all of the values available with seven bits, a zero (0) in Pixel Count is defined to mean that 1 pixel is encoded by the packet. A 1 in Pixel Count means that 2 pixels are actually contained in the packet. In other words, the value in Pixel Count is always 1 less than the actual number of pixels encoded in the packet. Thus, you may encode between 1 and 128 pixels with a single packet. If you have a run which is longer than 128 bytes, you must encode it using multiple packets.

Note: The high-order bit of this subfield is always set to 1 to indicate that the packet type is Run-length.

2. Pixel Data: This field contains a single pixel value to be repeated. The number of bits in this field is specified in Field 5.5 of the TGA File Header.

If 19 contiguous pixels in a scan line had the value 0x36 (assuming 8-bits-per-pixel) then the Run-length Packet would be encoded as follows:

Packet Type	Pixel Count	Pixel Data
1	0010010 (0x12)	00110110 (0x36)

Remember that the Repetition Count contains 1 less than the actual number of pixels being encoded. Since we were encoding 19 (0x13) pixels, a value of 18 (0x12) was placed in the repetition count. Also, since this is a Run-length Packet, the high-order bit is set to 1. This changes the Repetition Count byte from a 0x12 to a 0x92. The resulting packet is, therefore, 0x9236.

Raw Packet (i.e., Non-Run-Length Encoded):

The raw packet is always composed of two fields. The first field is the Repetition Count and the second field is the Pixel Data field.

Field Name	**Packet Type** MUST BE 0 for raw packet	**Pixel Count** (number of pixels encoded by this packet -1)	**Pixel Data**
Field Size	1 bit	7 bits	(Pixel depth * Pixel Count + 1)

Table 4 - Raw Packet

1. Pixel Count: The pixel count can range from 0 to 127. Since a raw packet never encodes zero pixels, and in order to make use of all of the values available with seven bits, a zero (0) in Pixel Count is defined to mean that 1 pixel is included in the packet. A 1 in Pixel Count means that 2 pixels are actually contained in the packet. In other words, the value in Pixel Count is always 1 less than the actual number of pixels contained in the packet. Thus, you may encode between 1 and 128 pixels with a single packet. If you have a run which is longer than 128 bytes, you must encode it using multiple packets.

Note: The high-order bit of this subfield is always set to 0 to indicate that the packet type is a Raw Packet.

2. Pixel Data: Non-Run-Length Encoded pixel data. The pixels will be displayed in the order that they are stored.