CIFF

Specification on Image Data File

Version : 1.0 revision 4

Date : December 24, 1997

Revision history

CIFF Specification On Image Data File, Feb 12,1997 (Japanese)

CIFF Specification On Image Data File, May 12,1997 (English, this document)
Revision 2, July 1, 1997 ImageFormat, ImageSpec property became mandatory.

Revision 3, December 19, 1997 Following bugs have been corrected.

On page 24

kTC_ReleaseMethod has been corrected to kTC_SR_ReleaseMethod

kTC_releaseTiming has been corrected to kTC_SR_ReleaseTiming

Revision 4, December 24, 1997

On page 1

The name of this format is corrected to Camera Image File Format.

The storage orders of length and offset field in following figures are corrected.

On page 3 Fig : Heap

On page 5 Fig : HierarchicalHeap

On page 7 Fig : storage format = kInRecordEntry

On page 8 Fig : storage format = kInHeapSpace

Trade marks:

Table of Contents

1 Purpose of Document

This document describes the format (CIFF: Camera Image File Format) of the data-files generated by Canon digital cameras.

2 Format Overview

The data format is defined so as to make it easy for the player (i.e., camera or host-computer application) to play back the images and/or audio generated by the camera.

The format must be one that can easily be adopted into the currently existing market and infrastructure. For this reason, the de facto standards of JFIF and WAVE are used, and the former has been extended for use with digital cameras.

To support these extensions (i.e., the additional information specific to digital cameras that is added to the JFIF file format[1], as described below), sample software shall be provided openly to player vendors. By providing these samples, we can prevent the evolution of different "dialects" of the format.

As for the storage of additional information specific to digital cameras, attention was given to making it easy for various manufacturers to add their own original extensions.

A digital camera is a device that produces multiple image and audio files. For this reason, in addition to the contents of the individual files themselves, we also defined a system of organizing the location of files to make it easy for a player to determine where each file is stored. This system is defined in a separate document: "CIFF Specification on Directory/File organization and File Handling Protocol."

3 Structure of Document

In the separate "Fundamental Data Model" section, we define the data structure used to store the digital-camera data (structure name: "heap")

For information on the JFIF standard, see document "JPEG File Interchange Format version 1.01 Eric Hamilton C-Cube Microsystems Dec. 10, 1991."

In the separate "File Format" section, we define the method used for storing the above-mentioned heap within a JPEG[2] data stream, and the various attribute settings related to the photographed images.

In the separate "Decisions Concerning Extension" section, we define how various companies participating in CIFF will go about adding to the standard to handle standard/common or independent

extensions.

In the separate "Verification of Compatibility" section, we define how various companies participating in CIFF will go about verifying generated data files.

For information on the JPEG standard, see "ISO IEC DIS 10918-1."

4 The Fundamental Data Model

In the JPEG files defined as CIFF, data (such as "date picture was taken") generated by the digital camera is stored in a format known as a "**HierarchicalHeap**" (described below). In this section, we will describe this format's content.

In order to make the organization of properties as general as possible, we will define the data structure called a "heap." A heap is a container for multiple variable-length records.

The type and structure of the content of each record is identified by a corresponding "type code." Thus,

by defining new type codes, it is possible to define new properties.

The type code and storage location of each record is stored in a single record of the OffsetTbl (described below).

By making the heap hierarchical, we allow for easy organization of data. A heap "hierarchicalized" in this way is referred to as a HierarchicalHeap.

The adoption of the hierarchical structure also allows a search-priority to be controlled. In other words, properties in higher layers can be searched more quickly than those in deeper layers. An example use of this is to have "date-taken" data and other frequently-accessed data located on a higher hierarchy-layer.

4.1 Heap

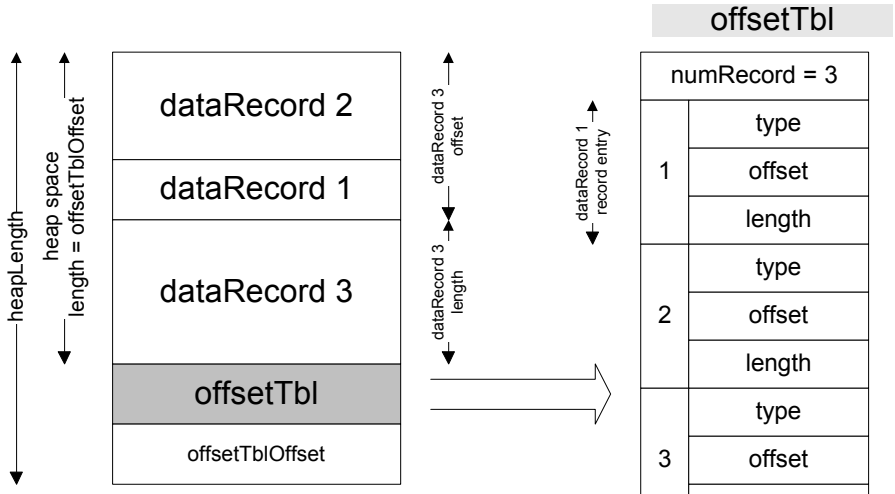


Fig : Heap

4.1.1 heapLength

heapLength indicates the overall length of the entire heap. In general, this value is already known, either from the file size or JPEG marker size, or from whatever other object containing the heap. Note that no information regarding the length of the heap is given within the actual heap data structure itself.

4.1.2 offsetTblOffset (=heapSpaceLength)

In the final position at the end of the heap, the position of **offsetTbl** is stored, as an offset relative to the start of the heap.

The space between the start of the heap and **offsetTbl** is referred to as the "heap space." Thus, **offsetTblOffset** is equal to the size of the heap space.

4.1.3 dataRecord

Data is stored one set at a time in individual data records.

The contents' structure of each record is identified by its type code. The type code for each record is noted in that record's entry in the OffsetTbl, described below.

The storage-position and length information for each data record is stored in that record's entry in the

OffsetTbl, described below.

The type, storage-position and length information for each data record is stored in a **recordEntry** corresponding to each data record in **offsetTbl**.

Each data record holds an index value, which associates it with a single RecordEntry within the **offsetTbl**. We will refer to this type of index value as a "**recordIndex**" below. A recordIndex is a numeric value beginning with 1. The value zero is therefore illegal when used as a recordIndex.

4.1.4 offsetTable

The **offsetTbl** begins with a field called a "**numRecord**," which indicates the number of records present in the heap. This value can legally be zero; in other words, a heap with no contents is not an illegal heap.

The **offsetTbl** is an array used for referring to a **recordEntry** given the corresponding index value. A **recordEntry** holds the information needed in order to get to the actual record data.

Each **recordEntry** holds the type, length, and storage-location information (i.e., offset value from the beginning of the heap) for the corresponding record.

If the record information is sufficiently short, it is possible to store the values inside the "**offset**" and "**length**" fields. See "Record Storage Method," below.

4.2 HierarchicalHeap

- The HierarchicalHeap data structure is a structure elaborated from a heap data structure by rendering it hierarchical. The records it consists of are either a "**single data record" (containing actual concrete values), or a "heap record" (i.e., data collections)**.
- The "type" field in each recordEntry contains a sub-field that indicates whether that entry is a single record, or a heap record.

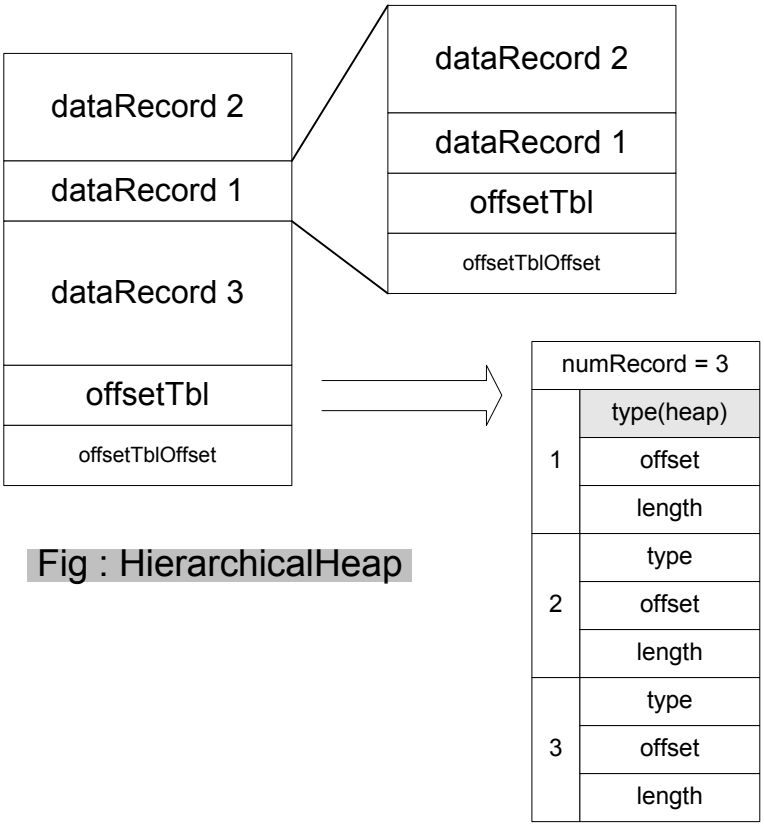


Fig : HierarchicalHeap

4.3 Record Storage Method

The "type" field in each recordEntry includes data concerning the storage location of the record data. It specifies one of the following[3]:

- kInRecordEntry
- kInHeapSpace

4.3.1.1 storage format = kInRecordEntry :

In order to save time and space, the data contents is recorded directly in the off**setTbl** itself, instead of in the heap.

The data contents are stored in the recordEntry's "off**set**" and "**length**" fields. No data whatsoever is stored in heap-space.

Normally, this type of data must have a defined size of 8 or less (i.e., the length of the "offset" and "length" fields). When this is not the case, datum defining the length of the data type must be present. That is, it must include -- within itself -- data allowing its length to be recognized, as in the example of a null-terminated string.

When parsing records using this storage method, a heap-parsing library will normally treat the length as having a value of 8.

If the length is less that 8, a "0" should be stored in the unused bytes.

[3] The present specifications are defined for the purpose of handling data in which each record is a few 10s of bytes in length, and the enti HierarchicalHeap itself is on the level of several kilobytes long.

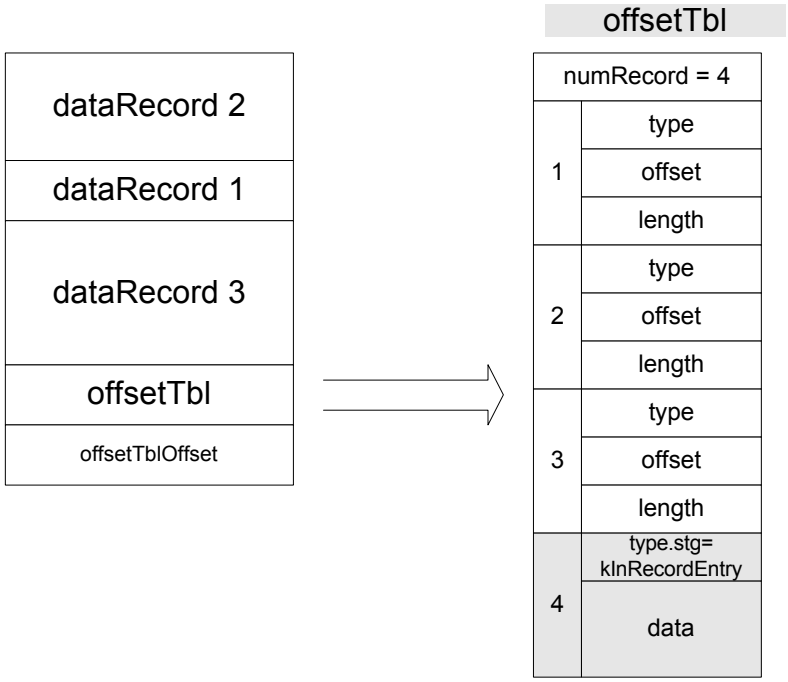


Fig : storage format = kInRecordEntry

4.3.1.2 storage format = kInHeapSpace :

The data contents are located in the heap itself.

The **length** and offset values within the **recordEntry** indicate the data's position and size within the heap.

All "**heap record**" (**data collection**) entries are stored in this format.

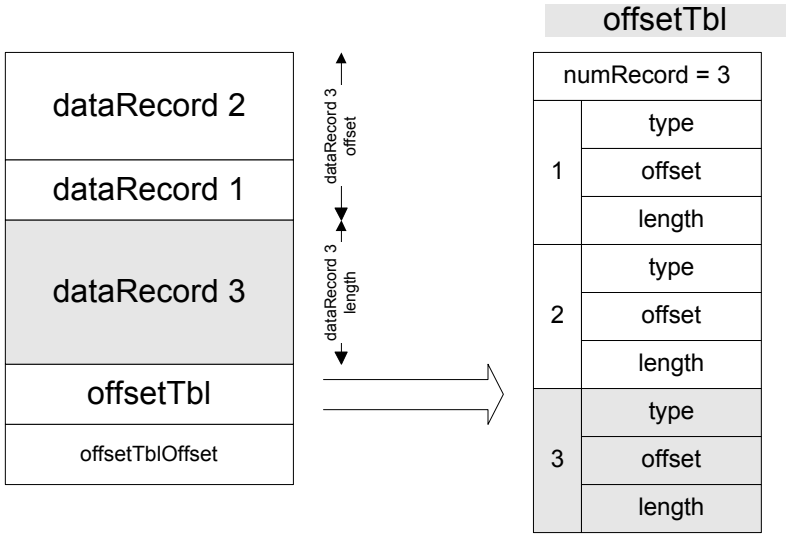


Fig : storage format = kInHeapSpace

4.4 The Heap File Format

Files that have a heap as their basic structure are referred to as having a "heap file" format. This data format is used for storing an image's attribute data inside a JPEG data stream.

In addition, we consider that this structure itself will be used as a general-purpose file format. In such cases, subType fields stored in the files' headers are used to identify the types of data stored in them.

For definitions of the fundamental data types that appear in the explanation below (i.e., DC_UINT32, an unsigned 32-bit integer), refer to the Appendix.

Also, support is mandatory for items accompanied by the symbol (M) in the explanation.

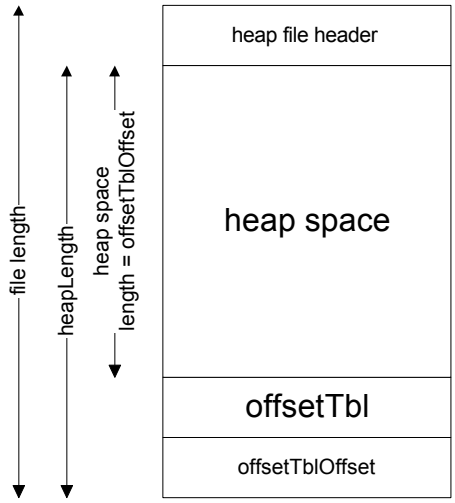


Fig : Heap file structure

4.4.1 Heap File Header(M)

The format of the heap file header is as follows:

```
typedef struct{
  char          byteOrder[2];/* 'MM' for Motorola,'II' for Intel */
  DC_UINT32     headerLength;/* length of header (in bytes) */
  char          type[4];
  char          subType[4];
  DC_UINT32     version; /* higher word: 0x0001, Lower word: 0x0002 */
  DC_UINT32     reserved1;
  DC_UINT32     reserved2;
} HEAPFILEHEADER, *pHEAPFILEHEADER;
```

byteOrder: Indicates the numerical significance-order of bytes: (MM: big-endian; II: little-endian). This applies to the entire heap file.

headerLength: The length of the header. In other words, the distance from the start of the file to the beginning of the "heap-space" in bytes.

type: Indicates the file type. The value stored here is "heap."

subType: A sub-classification of the file's type. For files holding JPEG data, store "JPGM" here. In addition to this value, the following values are currently reserved: "CCDR," "TIFP," and "ARCH."

version: Holds the file's version number. The most significant 16 bits hold the primary version, and the least significant 16 bits hold the sub-version. For CIFF, the former is 0x0001 and the latter is 0x0002.

4.4.2 offsetTbl

The format of the offsetTbl is as follows:

```
typedef DC_UINT16 TYPECODE;
typedef struct {
  TYPECODE      typeCode;/* type code of the record */
  DC_UINT32     length;/* record length */
  DC_UINT32     offset;/* offset of the record in the heap*/
}DC_RECORD_ENTRY;
```

typeCode: Indicates the record's type. Details are given in the following section.

length: Indicates the record's length (note: this is only true when the storage method is kStg_InHeapSpace).

offset: Indicates the location of the record within the heap (offset to location, in bytes) (note: this is only true when the storage method is kStg_InHeapSpace).

```
typedef struct {
  DC_UINT16 numRecords;/* the number tblArray elements */
  DC_RECORD_ENTRY tblArray[1];/* Array of the record entries */
}DC_OFFSETTBL;
```

numRecord: Indicates the number of entries present in the table. **numRe**cord = 0 is a legal situation: there need not be any member-entries present within the list.

tblArray[1] represents multiple entries. When **numRe**cord = 0, this item is not stored.

4.4.2.1 typeCode

The "typeCode" field in the DC_RECORD_ENTRY is a 16-bit unsigned integer, with the bits assigned as indicated below:

4.4.2.1.1 bit 15-14 type . stg

The storage method is defined as follows:

00: kStg_InHeapSpace

01: kStg_InRecordEntry

10: reserved

11: reserved

4.4.2.1.2 bit 13-11 type . dataType

The record type is defined as follows:

Within a single data record, alignment information is used to make it easier to generalize byte-order conversion between little-endian and big-endian orders.

For complex data (e.g., mixtures of 2-byte and 4-byte values), the contents is defined as byte alignment, and the byte-order conversion is left up to program code which has knowledge about the relevant data type.

000: single data record (byte alignment)

001: single data record (byte alignment character string)

010: single data record (2 byte alignment)

011: single data record (4 byte alignment)

100: single data record (arbitrary structure, treated as BYTE alignment)

101: heap record

110: heap record

111: reserved

4.4.2.1.3 bit 10-0 type . IDCode

The data-identifier codes are defined as follows:

4.4.3 offsetTblOffset

Off**setTblO**ff**set**'s data type is a 4-byte unsigned integer. It stores the offset from the start of the heap to the offsetTbl.

4.4.4 Special Purpose Data Records

Two special record types are defined to enable easier heap management:

4.4.4.1 null record (M)

stg.typeID: kStg_InHeapSpace | kTC_Null

alignment: 1

co**ntents**: Reserves an unused recordEntry field within the offsetTbl.

fo**rmat** : Stores "0s" in the length and offset fields in the DC_RECORD_ENTRY entry.

4.4.4.2 free record (M)

stg.typeID: kStg_InHeapSpace | kTC_Free

alignment: 1

co**ntents**: Reserves an unused region within the heap-space.

4.4.5 Concerning Data Alignment

For easy handling, all data records are defined with lengths that are multiples of two bytes. Thus, for example, if a string were an odd number of bytes long, it would have an extra byte appended to its length to make it an even number of bytes.

Given this rule, there are no units of data that are located at positions indicated by odd-numbered offsets, either in a heap or in a HeapFile.

5 File Format

5.1 The Base File Set

In CIFF, a pair consisting of a full view image and its thumbnail are stored with each record as its own independent JPEG file. In addition, when an audio annotation file is present, it is stored in the Windows® WAVE format. These three files comprise the standard files making up a CIFF image object. The term "image object" is used here to represent a concept including all of the information for a single image.

For details of rules for the naming of individual files and storage locations, see the separate document entitled "CIFF Specification on File/Directory organization and File Handling Protocol."

In the strictest sense, the decision as to whether thumbnail images should be stored as separate JPEG files is left up to the vendor, but consideration should be given to the fact that there is no guarantee that a player (i.e., a camera or a PC application) will be capable of producing a reduced-size image from the main image itself and then displaying it to the user.

For details of the internal format of WAVE files, refer to Microsoft Corporation documentation. For audio-file naming conventions, however, refer to the separate document entitled "CIFF Specification on File/Directory organization and File Handling Protocol."

5.2 JPEG (Full View Image)

JPEG data stream

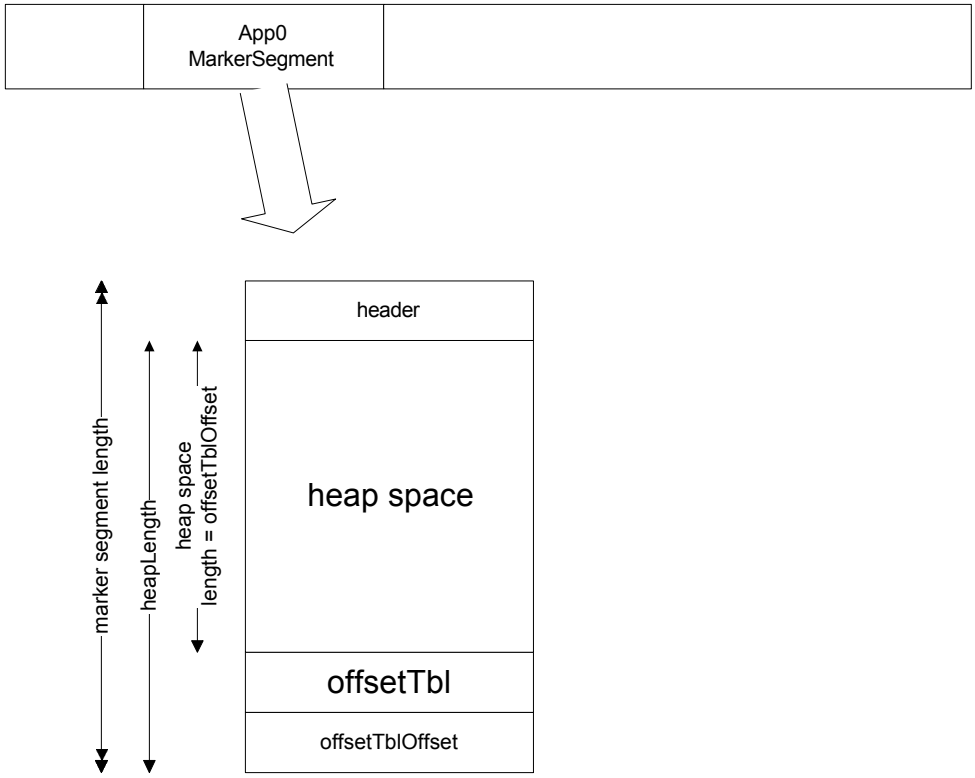


Fig : heap inside JPEG marker segment

The JPEG files generated by CIFF cameras have either JFIF version 1.01 or 1.02 APP0 markers.

JFIF version 1.01 APP0 markers store DPI information. The DPI values should clearly define the ratio between the X- and Y-axis lattice scales (M), and include information on what the manufacturer considers to be an appropriate print size. It is assumed that these DPI values will be used as defaults by the application software.

Another APP0 marker is positioned immediately after the JFIF marker. This marker is for holding information specific to that particular digital camera, and contains an entire heap-file structure, of which the topmost layer is an **ImageProps** heap (described below)[4]. The header's "**type**" field contains a "**HEAP**" and its "**subtype**" field contains a "**JPGM**".

5.3 JPEG (Thumbnail Image)

The Thumbnail JPEG has the same structure as the full view image. It stores a CIFF APP0 marker identical to that of the full view image. This means that it is possible to obtain information related to the full view image even from the thumbnail alone.

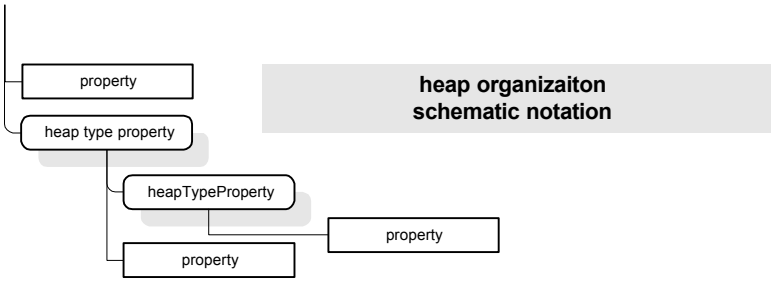
The image sizes are to have either 80*64 or 80*60 as their standard. In other words, the replay function is required, at the very least, to be able to play back thumbnails of this size. It is also acceptable for a camera to generate thumbnail which has a size up to 96*96, however it is not assured to be supported by players.

For file names, the first three characters should be THM. For further details, see the separate document entitled "CIFF Specification on File/Directory organization and File Handling Protocol."

[4] Note that, since it is being stored in a JPEG APP0 marker segment, information on the 64KB scale-level cannot be stored for the capability of replaying thumbnails of approximately 96*96 size. Nonetheless, this will depend on the replay function, therefore, the capability for such a replay cannot be guaranteed.

5.4 ImageProps heap

The diagram below shows the convention for defining property-positioning data. The rectangle indicates an individual data record, and the shadowed round-cornered rectangle indicates a heap record.



Within each record defined in the structure, the various members are stored in their locations in a "packed" state. That is, the members each occupy an absolute minimum amount of space.

For definitions of the fundamental data types (i.e., DC_UINT32, an unsigned 32-bit integer), refer to the Appendix.

The names of the individual properties below are marked (M) if their support is mandatory, and (R) if it is recommended. Those with neither of these markings are optional. Those marked (TBD) are currently under consideration (i.e., To Be Determined), and indicate that minor changes and/or the addition of more information should be expected.

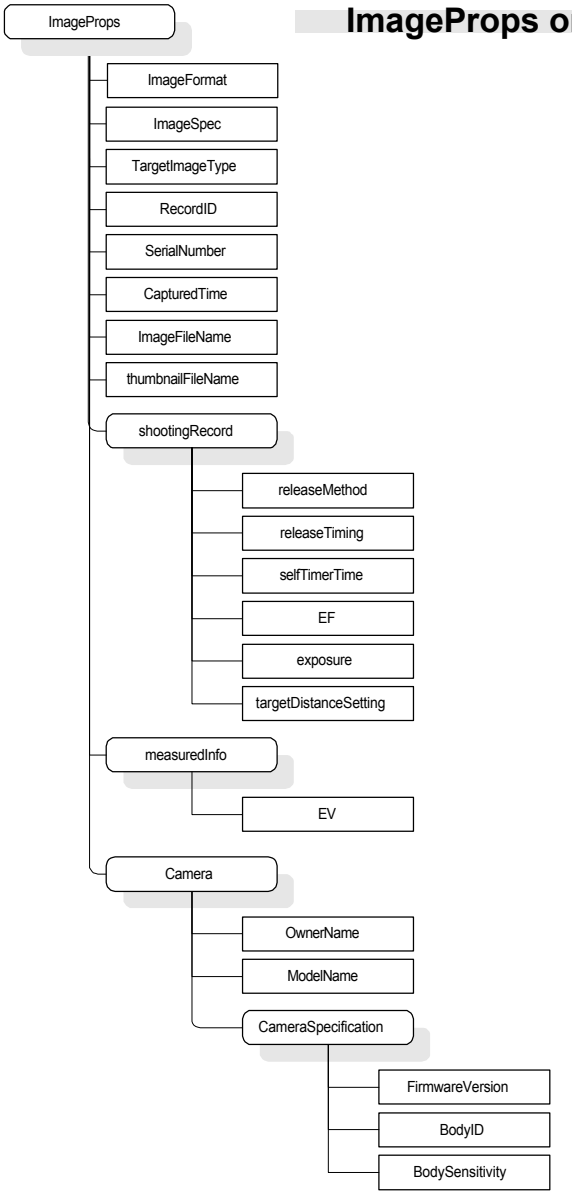
When any of the records defined below are absent, the information it represents is considered to be indeterminate. Note, however, that properties marked (M): Mandatory, are required to be present.

The ImageProps heap's property storage locations are as follows.

In the diagram below, an "ImageProps" heap is present in the topmost layer, but it should be noted that when this is the case (i.e., for JPEG), there will not be any record entries corresponding to the "ImageProps."

In the explanation below, each "**stg.typeID**" **label** indicates that particular property's storage method and the name of its data-type ID. Properties that are of fixed lengths greater than 8 bytes must be stored in heap-space. If they are 8 bytes or fewer, it is possible to store them in the record entry itself, instead. For character strings and other variable-length data, the decision as to where to store them can also be made at runtime, on the basis of their length. In other words, the identification of the data should fundamentally be conducted on the basis of bits 0-10 (i.e., type.datatype) of the TypeCode, and this is handled independent of the information regarding the storage method. Below, the definitions of **stg.typeID** are shown in a typical form.

ImageProps organizaiton



5.4.1 ImageFormat (M)

stg.typeID: kStg_InRecordEntry | kTC_ImageFormat

alignment: 4

contents: This defines the format of the full view-image file.

It should be noted that the information stored in the JPEG file of the thumbnail image is information concerning the full view image. In other words, information concerning the thumbnail image itself is not stored; only that concerning the corresponding full view image is. This property is present for this reason, and is significant particularly when the principal image is not in JPEG format.

format :

```
typedef struct {
  DC_UINT32 fileFormat;
  DC_FLOAT32 targetCompressionRatio;
}DC_IMAGEFORMAT;
```

fileFormat : This defines the image format of the principal image.

Most-significant 16 bits: 1: JPEG; 2: Reserved (for CanonCRW format)

Least-significant 16 bits: Compression type 0: lossy JPEG (that in which the value in the JPEG quantization table is not 1); 1: none (for CRW); 2: non-quantization JPEG (that in which 1 has been used for the value in the JPEG DCT quantization table); 3: in the Canon PS600, lossy/non-quantization has been toggled to correspond with the toggling of the picture/text switch.

targetCompressionRatio : This holds the target compression ratio in bits per pixel. Note that this is a target value, not the compression result.

5.4.2 ImageSpec(M)

stg.typeID: kStg_InHeapSpace | kTC_ImageSpec

alignment: 4

contents: This defines the image format.

format :

```
typedef struct {
  DC_UINT32     imageWidth;
  DC_UINT32     imageHeight;
  DC_FLOAT32    pixelAspectRatio;
  DC_SINT32     rotationAngle;
  DC_UINT32     componentBitDepth;
  DC_UINT32     colorBitDepth;
  DC_UINT32     colorBW;
```

```
}DC_IMAGESPEC;
```

imageWidth: Number of horizontal pixels.

imageHeight: Number of vertical pixel.

pixelAspectRatio: Ratio between the horizontal and vertical size of a pixel (horizontal/vertical).

rotationAngle: This indicates how many degrees in the counter-clockwise direction the original data file should be rotated for correct viewing of the image. If the image has already been rotated in the JPEG format, this value will be zero. When this value is modified, it should be modified in both the thumbnail and full view-image files. Nonetheless, if the two values differ, the player should give final precedence to the data stored in the full view-image file. If this value is then used, for example, to rotate the data stored in the JPEG file structure itself, the imageWidth, imageHeight, pixelAspectRatio and other data should be updated correspondingly. This value should be set to 0 when the image is not sure to be rotated.

componentBitDepth: This indicates the length in bits of each color component. Ex.: 8.

ColorBitDepth: The total amount of all color information. Ex.: 24 (color), or 8 (grayscale).

colorBW:

LS-byte: 0: grayscale; 1: color

2nd byte: Indicates whether post-processing (pixel aspect-ratio conversion) should be applied (value 1) or not (value 0).

For cameras which do not employ square-pixel CCDs, this value should be set to 1 if it is desired that the aspect ratio of the pixels should be converted after the images are recorded. If the converted images are then re-recorded in CIFF JPEG format, this value should be switched to 0.

3rd and 4th bytes: reserved 0

5.4.3 TargetImageType(R)

stg.typeID: kStg_InRecordEntry | kTC_TargetImageType

alignment: 2

c**ontents**: This indicates whether the photograph was taken with the intention of recording a real-world subject, or a written document.

f**ormat** :

```
typedef struct {
  DC_UINT16     targetImagetype;
  DC_UINT16     reserved1;
  DC_UINT16     reserved2;
  DC_UINT16     reserved3;
}DC_TARGET_IMAGE_TYPE;
```

targetImageType: 0:Real-world subject; 1: Written document

5.4.4 RecordID

stg.typeID: kStg_InRecordEntry | kTC_RecordID

alignment: 2

contents: Number of image photos taken since the camera was shipped.

format :

```
typedef struct {
  DC_UINT32     recordID;
  DC_UINT32     reserved1;
}DC_RECORD_ID;
```

recordID: Number of image photos taken since the camera was shipped. (Not including audio recordings.)

5.4.5 SerialNumber

stg.typeID: kStg_InRecordEntry | kTC_SerialNumber

alignment: 4

contents: File number used when camera creates files. (See "Digital Camera 97 Specification on Directory/File Organization." Note that this is an arbitrary value with a ceiling of 5000000. The file name shows only the last five digits thereof.)

format :

```
typedef struct {
  DC_UINT32     serialNumber;
  DC_UINT32     reserved1;
}DC_SERIAL_NUMBER;
```

5.4.6 CapturedTime(M)

stg.typeID: kStg_InRecordEntry | kTC_CapturedTime

alignment: 4

contents: Time photograph was taken.

format :

```
typedef struct {
  DC_UINT32   TimeCount;
  DC_SINT32   TimeZoneCode;
  DC_UINT32   TimeZoneInfo;
}DC_TIME, *pDC_TIME;
```

TimeCount: This is identical to the time_t data type defined in the C-language library. That is, it is the cumulative number of seconds which have passed since midnight exactly of January 1st, 1970.

TimeZoneCode: This is identical to the definition of the value that appears in the C-language library. For Japan, for example, it is -9*3600.

TimeZoneInfo: When bit 31 is "1," that indicates that the TimeZoneCode represents a valid value.

If bit 31 is set to "0," the TimeZoneCode is considered 0. In this situation, since TimeZoneCode is 0, and therefore represents international (Greenwich) standard time, the time calculated from the TimeCount is set in the camera as its local time.

Bits 30 through 0 of TimeZoneInfo represent additional information when bit 31 is valid (i.e., when it is "1"). For example, there are plans for storing city-name and daylight savings time (summer time) indicators in these bits. Currently, these bits contain zeros, indicating a reserved status.

5.4.7 ImageFileName

stg.typeID: kStg_InHeapSpace | kTC_ImageFileName

alignment: 1

contents: Filename of the full view image.

format : **null terminated string**

5.4.8 ThumbnailFileName

stg.typeID: kStg_InHeapSpace | kTC_ThumbnailFileName

alignment: 1

contents: Filename of the thumbnail file.

format : **null terminated string**

5.4.9 Description

stg.typeID: kStg_InHeapSpace | kTC_Description

alignment: 1

contents: Arbitrary character data.

format : **null terminated string**

5.4.10 ShootingRecord heap

stg.typeID: kStg_InHeapSpace | kTC_ShootingRecord

alignment: 1

contents: Heap for storing photo-shooting data.

format : **heap**

5.4.10.1 ShootingRecord.ReleaseMethod(TBD)

stg.typeID: kStg_InRecordEntry | kTC_SR_ReleaseMethod

alignment: 2

contents: Method of taking pictures (i.e., shutter release).

format :

```
typedef struct {
  DC_UINT16     releaseMethod;
  DC_UINT16     reserved1;
  DC_UINT16     reserved2;
  DC_UINT16     reserved3;
}DC_RELEASE_METHOD;
```

releaseMethod : 0: Single shot; 2: Continuous/successive exposures

5.4.10.2 ShootingRecord.ReleaseTiming(TBD)

stg.typeID: kStg_InRecordEntry | kTC_SR_ReleaseTiming

alignment: 2

contents: Release timing.

format :

```
typedef struct {
  DC_UINT16     releaseTiming;
  DC_UINT16     reserved1;
  DC_UINT16     reserved2;
  DC_UINT16     reserved3;
}DC_RELEASE_TIMING;
```

releaseTiming: 0: Priority on shutter; 1: Priority on focus.

5.4.10.3 ShootingRecord.SelfTimerTime(TBD)

stg.typeID: kStg_InRecordEntry | kTC_SelfTimerTime

alignment: 4

contents: Time until release when using auto-release timer function.

format :

```
typedef struct {
```

```
  DC_UINT32     selfTimerTime;
  DC_UINT32     reserved1;
}DC_SELF_TIMER;
selfTimerTime: This indicates the timer time, in millisecond units.
```

5.4.10.4 ShootingRecord.EF(TBD)

stg.typeID: kStg_InRecordEntry | kTC_SR_EF

alignment: 4

c**ontents**: Result of flash use.

f**ormat** :

```
typedef struct {
  DC_FLOAT32    efGuideNumber;
  DC_FLOAT32    efThreshold;
}DC_SR_EF;

efGuideNumber: Flash-emission guide-number. A value of zero indicates that the flash
was not discharged.
efThreshold: EV value set for use in making decisions as to whether or not to use
flash. The flash is discharged if the measured value falls below this value.
```

5.4.10.5 ShootingRecord.Exposure(R)

stg.typeID: kStg_InHeapSpace | kTC_SR_Exposure

alignment: 4

c**ontents**: Exposure information.

f**ormat** :

```
tyepdef struct{
  DC_FLOAT32 exposureCompensation;
  DC_FLOAT32 tv;
  DC_FLOAT32 av;
}CDT_SR_EXPOSURE;

exposureCompensation:  Value used to supplement the exposure value.
tv: TV value.
av: AV value.
```

5.4.10.6 ShootingRecord.TargetDistanceSetting

stg.typeID: kStg_InRecordEntry | kTC_TargetDistanceSetting

alignment: 4

contents: This holds the distance to the subject, in one-millimeter units.

format :

```
tyepdef struct{
  DC_FLOAT32 targetdistanceSetting; /* unit mm */
  DC_FLOAT32 reserved;
};
```

targetdistanceSetting: Distance to the point of focus, in one-millimeter units.

5.4.11 MeasuredInfo heap

stg.typeID: kStg_InHeapSpace | kTC_MeasuredInfo

contents: A heap storing the measurement information gathered at the time the picture was taken

format : heap

5.4.11.1 MeasuredInfo.EV

stg.typeID: kStg_InRecordEntry | kTC_MeasuredInfo

alignment: 4

contents: Measured EV value (Luminance Value).

format :

```
tyepdef struct{
  DC_FLOAT32 ev;
  DC_FLOAT32 reserved;
};
```

ev : EV value.

5.4.12 Camera heap(M)

stg.typeID: kStg_InHeapSpace | kTC_CameraObject

alignment:

contents: Information on the camera that was used to take the picture.

format : heap

5.4.12.1 Camera.OwnerName(R)

stg.typeID: kStg_InHeapSpace | kTC_OwnerName

alignment: 1

contents: The owner-name setting entered into the camera.

format : null terminated string

5.4.12.2 Camera.ModelName(M)

stg.typeID: kStg_InHeapSpace | kTC_ModelName

alignment: 1

contents: The camera model-name.

format : Two null-terminated stings, with an additional null at the end: the first is the manufacturer name (e.g., "Canon") and the next is the model name (e.g., "PowerShot ver1.00"), followed by a null. There will therefore be a total of two consecutive nulls at the end of this entry.

5.4.12.3 Camera. CameraSpecification heap

stg.typeID: kStg_InHeapSpace | kTC_CameraSpecification

alignment: 1

contents: A heap storing information concerning the camera's specifications.

format : **HEAP**

5.4.12.3.1 Camera. CameraSpecification. BodyID

stg.typeID: kStg_InRecordEntry | kTC_BodyID

alignment: 4

contents: The camera's production-identification code (e.g., serial number). This is primarily to help with the manufacturer's service procedures.

format : In principal, the manufacturer's own arbitrary formats is permitted. As a convention, the first 32 bits should represent the serial number, as in the following definition:

```
tyepdef struct{
  DC_UINT32 bodyID;
  DC_UINT32 reserved;
};
```

5.4.12.3.2 Camera. CameraSpecification.BodySensitivity (TBD)

stg.typeID: kStg_InRecordEntry | kTC_BodySensitivity

alignment: 2

contents: The camera's base ISO number.

format :

```
tyepdef struct{
  DC_UINT16 defaultISO;
  DC_UINT16 reserved1;
  DC_UINT16 reserved2;
  DC_UINT16 reserved3;
};
```

5.4.12.3.3 Camera. CameraSpecification.FirmwareVersion(R)

stg.typeID: kStg_InHeapSpace | kTC_FirmwareVersion

alignment: **1**

co**ntents**: The camera's firmware version. This is for service use, and to allow it to be displayed to the end user by the application software.

fo**rmat** : **null terminated string • ie**x. "Firmware version 1.00")

6 Decisions Concerning Extension

Since the information in the HierarchicalHeap is recognized fundamentally on the basis of type codes, extension can be accomplished easily, simply by adding new type codes. Furthermore, the fact that its' structure is hierarchical simplifies the process whereby different vendors can define their own original data.

The currently defined heap (

ImageProps,
ImageProps.Shootingrecord,
ImageProps.MeasuredInfo,
ImageProps.Camera,
ImageProps.Camera.CameraSpecificaiton
)

is called the basic common heap (cf. Diagram shown previously: "ImageProps Organization").

The current basic common heap, and the data records organized therein, shall be considered to define the first version of the common format.

Within the basic common heap, participating vendors may declare additional inter-vendor common records, based on agreements reached in deliberative consultation. Proposals should be made via e-mail, and when all participating companies reach a unanimous agreement, Canon will issue a corresponding formal document. By having the entire process conducted in this manner via e-mail, we will do the utmost to reduce the time taken from proposal to the issuing of the formal document.

Regarding vendor-specific original records, any vendor is welcome to establish such records by setting up a vendor-specific heap under the basic common heap, and placing them therein. The issuing of type codes for vendor-specific heaps will be managed by Canon's contact desk, so as to maintain uniqueness. This issuing will be done automatically, and Canon will not make an inquiry on their contents. Note however, that a limit will be placed on the number of different heap types a single vendor may define (on the order of 10 or so).

When a vendor has established a vendor-specific heap under the basic common heap, the record numbers within the former may be established arbitrarily by that vendor. Nonetheless, we recommend that the regulations for data-type (i.e., data vs. heap), storage method (heap-space vs. record entry), and data alignment follow those set out in this document.

In order to avoid conflicts in the initial "manufacturer's name" string in the mandatory

Camera.ModelName property, it will be necessary for each manufacturer to inform Canon of the exact string they will use for their name. Canon will then automatically check on each name to ensure it has not been previously selected.

7 Verification of Compatibility

A program for parsing the common properties present within ImageProps inside CIFF JPEG files will be provided as a test program from Canon. It is the responsibility of each company to use the program to assure that the data files they generate can be parsed without any problems.

The program will be provided both as an executable and source file. Canon will only test that it runs correctly in a Windows® 32-bit environment; no guarantee will be given that it will run on other platforms. Canon will provide explanation of the source code only in the form of the comments embedded therein, and in an accompanying document; inquiries by e-mail or other means will not be accepted.

The program will be updated every time a new common property is issued, and any time other changes occur that are common to all companies.

Each company is responsible for modifying the source code in accordance with the company-specific properties that it has defined, and for using the resulting program to verify that both the common and company-specific properties can be restored without problem.

Canon will fix bugs in the test program itself, and depending on the situation, may also provide support when issues regarding the program are pointed out by the companies; but fundamentally, Canon will conduct none of the necessary diagnosis based on problem symptoms (e.g., Canon will not determine in what way any particular data is incorrect, etc.).

8 Appendix

8.1 Definitions of Fundamental Data Types

```
char: signed 8 bit integer or character
DC_SINT16: signed 16 bit integer
DC_UINT16: unsigned 16 bit integer
DC_SINT32: signed 32 bit integer
DC_UINT32: unsigned 32 bit integer
DC_BYTE: unsigned 8 bit integer
DC_FLOAT32: 32 bit floating point number
```

8.2 type.dataType and type.IDCode definition

```
#define kStgFormatMask                  0xc000
#define kDataTypeMask                   0x3800
#define kIDCodeMask                     0x07ff
#define kTypeIDCodeMask                 0x3fff

#define kStg_InHeapSpace                0x0000
#define kStg_InRecordEntry              0x4000

#define kDT_BYTE                        0x0000
#define kDT_ASCII                       0x0800
#define kDT_WORD                        0x1000
#define kDT_DWORD                       0x1800
#define kDT_BYTE2                       0x2000
#define kDT_HeapTypeProperty1           0x2800
#define kDT_HeapTypeProperty2           0x3000

#define kTC_WildCard            0xffff
#define kTC_Null                0x0000  /* null record */
#define kTC_Free                0x0001  /* free record */
#define kTC_ExUsed              0x0002
                                //special type for implementation purpose*/

#define kTC_Description                 (kDT_ASCII | 0x0005)
#define kTC_ModelName                   (kDT_ASCII | 0x000a)
#define kTC_FirmwareVersion             (kDT_ASCII | 0x000b)
#define kTC_ComponentVersion            (kDT_ASCII | 0x000c)
#define kTC_ROMOperationMode            (kDT_ASCII | 0x000d)
#define kTC_OwnerName                   (kDT_ASCII | 0x0010)
#define kTC_ImageFileName               (kDT_ASCII | 0x0016)
#define kTC_ThumbnailFileName           (kDT_ASCII | 0x0017)

#define kTC_TargetImageType             (kDT_WORD | 0x000a)
#define kTC_SR_ReleaseMethod            (kDT_WORD | 0x0010)
#define kTC_SR_ReleaseTiming            (kDT_WORD | 0x0011)
#define kTC_ReleaseSetting              (kDT_WORD | 0x0016)
#define kTC_BodySensitivity             (kDT_WORD | 0x001c)

#define kTC_ImageFormat                 (kDT_DWORD | 0x0003)
#define kTC_RecordID                    (kDT_DWORD | 0x0004)
#define kTC_SelfTimerTime               (kDT_DWORD | 0x0006)
#define kTC_SR_TargetDistanceSetting    (kDT_DWORD | 0x0007)
```

```
#define kTC_BodyID                      (kDT_DWORD | 0x000b)
#define kTC_CapturedTime                (kDT_DWORD | 0x000e)
#define kTC_ImageSpec                   (kDT_DWORD | 0x0010)
#define kTC_SR_EF                       (kDT_DWORD | 0x0013)
#define kTC_MI_EV                       (kDT_DWORD | 0x0014)
#define kTC_SerialNumber                (kDT_DWORD | 0x0017)
#define kTC_SR_Exposure                 (kDT_DWORD | 0x0018)

#define kTC_CameraObject                (0x0007 | kDT_HeapTypeProperty1)
#define kTC_ShootingRecord              (0x0002 | kDT_HeapTypeProperty2)
#define kTC_MeasuredInfo                (0x0003 | kDT_HeapTypeProperty2)
#define kTC_CameraSpecificaiton (0x0004 | kDT_HeapTypeProperty2)
```