INTERNATIONAL STANDARD

ISO/IEC 12089

First edition
1997-12-15

Information technology — Computer graphics and image processing — Encoding for the Image Interchange Facility (IIF)

Technologies de l'information — Infographie et traitement de l'image — Codage pour les accessoires pour l'échange de l'image (IIF)

Reference number
ISO/IEC 12089:1997(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 12089 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 24, *Computer graphics and image processing*.

Information technology — Computer graphics and image processing — Encoding for the Image Interchange Facility (IIF)

1 Scope

This International Standard defines the encoding rules which shall apply to the representation of IPI-IIF image data. The IPI-IIF data format is defined in ISO/IEC 12087-3, called „Image Interchange Facility (IIF)". It is Part 3 of the Image Processing and Interchange International Standard, defined in ISO/IEC 12087. The IPI-IIF facilitates the interchange of digital images. It consists of two major parts:

(1) the IPI-IIF data format (IIF-DF) definition, whose syntax is described using ASN.1;

(2) the IPI-IIF gateway definition, whose functionality is described by an application programmers interface.

The IPI-IIF is based on the definition described in Part 1, *Common Architecture for Imaging* (CAI) of the ISO/IEC 12087.

Due to the fact that the syntax of the IIF-DF is expressed using the *Abstract Syntax Notation One* (ASN.1), defined by ISO/IEC 8824, this standard makes use of the *Basic Encoding Rules* (BER) for ASN.1, by referring to ISO/IEC 8825 for the definition of encoding rules.

NOTE - A rationale for the introduction of new encoding rules in addition to those defined by the BER is given in clause 4.

Reference shall be made to this International Standard, and its definitions shall be employed, whenever images are interchanged, according to the IIF-DF, defined in ISO/IEC 12087-3.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8632:(all parts),	*Information technology - Computer graphics - Metafile for the storage and transfer of picture description information.*
ISO/IEC 8824:1990,	*Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).*
ISO/IEC 8825:1990,	*Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*
ISO/IEC 8825-2:1996,	*Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*
ISO/IEC 12087-1:1995,	*Information technology - Computer graphics and image processing - Image Processing and Interchange (IPI) - Functional specification - Part 1: Common architecture for imaging.*
ISO/IEC 12087-2:1994,	*Information technology - Computer graphics and image processing - Image Processing and Interchange (IPI) - Functional specification - Part 2: Programmer's imaging kernel system application programme interface.*
ISO/IEC 12087-3:1995,	*Information technology - Computer graphics and image processing - Image Processing and Interchange (IPI) - Functional specification - Part 3: Image Interchange Facility (IIF).*

NOTE - Some ISO Standards are technically aligned with CCITT Recommendations, in particular the ASN.1 Standard (ISO Standards 8824/8825 and CCITT Recs. X.208/X.209). The differences between the International Standard definitions and the CCITT definitions are quite small, and should not affect interoperability between implementations written against either document. Within this part of ISO/IEC 12087, the ISO Standards are referenced whenever possible.

3 Definitions and abbreviations

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
frc	fraction
lsb	least significant bit
lsB	least significant Byte
msb	most significant bit
msB	most significant Byte
s	sign

NOTE - For definitions and abbreviations concerning the Image Processing and Interchange Standard (IPI), refer also to clause 3 of ISO/IEC 12087-1, ISO/IEC 12087-2 and ISO/IEC 12087-3.

4 Encoding rules for the IIF syntax entities

The encoding of syntax entities shall conform to the *Basic Encoding Rules* (BER) for ASN.1 - ISO/IEC 8825.

NOTE - Using the BER encoding overheads may occur. In particular, the encoding of a large pixel data field can produce considerable space and processing time overhead, when every pixel is represented as an elementary ASN.1 data entity, consisting of a „tag“ and a „length“ field that proceeds the „value“ field.

For this reason this International Standard describes additional encoding methods that may be applied to pixel fields. These methods may neither be regarded as extensions, nor as changes to the tag-length-value concept of the BER. Instead, they only describe how to interpret the data contained in the elementary ASN.1 type OCTET STRING, when this elementary type was used to encode an entire field of pixel values (instead of just one value). Thus, these encoding rules may rather be regarded to lie „on top of“ BER encoding/decoding tools.

This International Standard defines additional encodings for space-efficient representation of pixel fields. They are outlined in clause 5 in conjunction with additional IIF syntax entities which describe the degree of freedom for the selection of an encoding method.

5 IIF syntax entities for the representation of pixel fields

The syntax is expressed in ASN.1 (Abstract Syntax Notation One), according to ISO/IEC Standard 8824, "Specification of Abstract Syntax Notation One (ASN.1)." ASN.1 is a formal description language. It defines a set of primitive data types, such as `INTEGER`, `ENUMERATED`, and `REAL` and provides a facility to construct new elements with their own typing inherent in the structure using the constructors `SEQUENCE`, `SEQUENCE OF` and `CHOICE`. This allows for new data types to be defined which are uniquely recognisable within an application. To make these definitions more readable, textual labels may be associated with the elements in a constructor type. In order to distinguish different occurrences of the same type within one constructor, various types of tags are provided that may be associated with the constructor's elements.

Within the semantic description each element (which is either a primitive data type or a constructed type) is called *syntax entity*. According to ASN.1, the names of the syntax entities begin with capital letters. Syntax entities consist of a number of *component*s. According to ASN.1, the component labels begin with lower case letters.

In the following, ASN.1 code is indicated by `courier font`. All syntax rules are preceded by a semantics statement. Some rules are succeeded by constraints statements. The rules are ordered in prefix form.

IIF module declaration *IIFEncoding*

Semantics

IIFEncoding is the name of the ASN.1 module responsible for the encoding of fields of pixel values to be used mainly by the IIF-DF. Besides the full syntax (given by the *PixelFieldEncoding* entity), the module also exports the *BooleanEncoding, IntegerEncoding, FixedPointEncoding, RealEncoding, ComplexEncoding* and the *EnumeratedEncoding* entities. This provides other ASN.1 notated applications with direct access to these sub-objects. No objects are imported.

In order to obtain the full syntax for the module specification, the term `<<declarations>>` needs to be replaced with the syntax portions of all subsequent syntax entities within this clause.

```
IIFEncoding {iso(1) standard (0) ipi-encoding(12089)
             iif-encoding(1) }

DEFINITIONS ::=

BEGIN

EXPORTS

PixelFieldEncoding,
BooleanEncoding.
IntegerEncoding,
FixedPointEncoding,
RealEncoding,
ComplexEncoding,
EnumeratedEncoding;

IMPORTS;

<<declarations>>

END
```

Constraints

None.

IIF syntax entity No. 1001 *PixelFieldEncoding*

Semantics

The *PixelFieldEncoding* entity is used to represent a field of pixel values whose structure is defined by an external image structure definition.

NOTE - In case of the IIF-DF, this image structure definition is given by the *ImageStructure* entity.

The *byte-order-swapped* component indicates the sequential order in which consecutive octets appear within the corresponding *pixel-value* component. The definitions of subsequent data types apply after octet swapping, if specified.

Possible values are:

„0" The physical order matches the significance:

```
  7   6   5   4   3   2   1   0
---------------------------------
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
---------------------------------
```

„1" The octets are swapped:

```
  7   6   5   4   3   2   1   0
---------------------------------
| 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
---------------------------------
```

NOTE -When using ASN.1, machine dependencies, such as byte swapping are managed by the BER. However, within the *PixelFieldEncoding* entity an ASN.1 OCTET STRING is used to represent not just one value, but multiple values of various types such as integer or real. Thus, these machine dependencies need to be managed by the application again.

The *encoding-rules* component determines the encoding rules that apply to the field of pixel values.

The *pixel-values* component represents the field of pixel values by the ASN.1 type BITSTRING.

NOTE -A field element is either the value of a pixel, or the value of an elementary part of a pixel, depending on whether the pixel type is defined to be elementary or compound (e.g., a record of some elementary types).

Syntax:

```
PixelFieldEncoding ::= SEQUENCE
   {
   byte-order-swapped   [0] INTEGER (0..1),
   encoding-rules       [1] EncodingRules,
   pixel-values         [2] BITSTRING
   }
```

Constraints

The number field elements contained in the *pixel-values* component must match the number of array elements declared by the corresponding image structure definition.

IIF syntax entity No. 1002 *EncodingRules*

Semantics

The *EncodingRules* entity provides a generic method for the specification of the packed encoding of pixel values. The *uniform-encoding* component describes the encoding of heterogeneous pixel fields, while the *hierarchical-encoding* component facilitate the description of the complex hierarchical encodings.

EXAMPLE - Given a 2D image, which consists of two bands, called *hi* and *lo*, whereby hi has three times the resolution of *lo* in both dimensions: 192x192 pixels for *hi*, and 64x64 pixels for *lo*. According to its image structure definition, the image is represented pixel-interleaved, i.e. it consists of 3 by 3 blocks of *hi* pixels followed by a single *lo* pixel, followed by another block of *hi* pixels, etc. Let us now look at two cases

1) Let us assume that both bands have 8 bit unsigned integer pixels:

In this case the pixel values of both bands are represented in the same way. Thus, on the level of the encoding specification it is not necessary to express the complex pixel interleaved organisation. Instead it is sufficient to specify a plain (1-dimensional) sequence of 8 bit integers. The iteration component is being set to 49960 = 192x192 + 64x64. The bit pad component is set to zero, since no padding bits occur in the sequence.

```
EncodingRules
    uniform-encoding
        iteration-and-alignments          = SEQUENCE OF
            1                           = IterationAndAlignment
                explicit-iteration      = INTEGER                    40960
                alignment               = Padding
                    bit-pad             = INTEGER                    0
        components-encoding             = SEQUENCE OF
            1                           = ElementEncoding
                non-negative-integer    = UnsignedIntegerEncoding
                    number-of-bits      = INTEGER                    8
```

2) Let us assume that hi band consists of 6 bit integers and the low band consists of 4 octet floats.

In this heterogeneous case a hierarchical encoding specifications is required, if each block of pixel elements is aligned to a 2 octet boundary. The first level describes the loop over pixel blocks. The second level describes the representation within one block.

```
EncodingRules
    hierarchical-encoding
        iteration-and-alignment             = IterationAndAlignment
                explicit-iteration          = INTEGER                    64
                alignment                   = Padding
                    octet-boundary          = INTEGER                    2
        components-encoding                 = SEQUENCE OF
            1                               = ComplexEncoding
                elementary-component        = ElementEncoding
                    non-negative-integer    = UnsignedIntegerEncoding
                        number-of-bits      = INTEGER                    6
            2                               = ComplexEncoding
                elementary-component        = ElementEncoding
                    real                    = RealEncoding
                        ieee-basic-single   = NULL
```

Syntax

```
EncodingRules ::= CHOICE
   {
   uniform-encoding        [0] UniformEncoding,
   hierarchical-encoding   [1] HierarchicalEncoding
   }
```

Constraints

None.

IIF syntax entity No. 1003 *UniformEncoding*

Semantics

The *UniformEncoding* entity describes the encoding of uniform encoded pixel fields. The *iteration-and-alignment* component is used to describe the place and size of padding bits. It is organised as a sequence of *IterationAndAlignment* entities in order to support alignment in multiple dimensions. The first entity determines the outer most loop over the field elements. The last entity indicates the inner-most loop.

EXAMPLE - Give a field of elements for which the following iteration and alignment is specified for (for the exact description of the Padding entity see below):

IterationAndAlignment$_1$:	*explicit-iteration = 64,*	*padding = 2 bit*
IterationAndAlignment$_2$:	*explicit-iteration = 128,*	*padding = 6 bit*

According to this specification the field consists of 128 packed encoded elements, followed by a 6 bit pad. This sequence is iterated 64 times, followed by a 2 bit pad at the end of the entire field.

The *components-encoding* component is used to describe the encoding of the field elements for which the above mentioned iteration and alignment is specified.

In order to facilitate the description of encodings of heterogeneous pixel fields (e.g. fields which consists of alternating sequences of integer and real pixel values) the *components-encoding* is organised as sequence of *ElementEncoding* entities, each which describes the encoding of one pixel component.

Syntax

```
UniformEncoding ::= SEQUENCE
    {
    iteration-and-alignment      [0] SEQUENCE OF IterationAndAlignment,
    components-encoding          [1] SEQUENCE OF ElementEncoding
    }
```

Constraints

None.

IIF syntax entity No. 1004 *HierarchicalEncoding*

and

IIF syntax entity No. 1005 *ComplexEncoding*

Semantics

The *HierarchicalEncoding* entity provides the ability to construct hierarchical encoding descriptions through its recursive definition. The *iteration-and-alignment* component is used to describe the place and size of padding bits. The *components-encoding* provides the ability to construct hierarchical encoding definitions. In the *ComplexEncoding* entity two alternatives are provided:

- The *elementary-component* component is the encoding of an elementary pixel value without further levels of hierarchy.
- The *complex-encoding* component allows to represent a complex encoding specification recursively via the *HierarchicalEncoding* entity.

Syntax

```
HierarchicalEncoding ::= SEQUENCE
    {
    iteration-and-alignment      [0] IterationAndAlignment,
    components-encoding          [1] SEQUENCE OF ComplexEncoding
    }

ComplexEncoding ::= CHOICE
    {
    elementary-component         [0] ElementEncoding,
    complex-component            [1] HierarchicalEncoding
    }
```

Constraints

None.

IIF syntax entity No. 1006 *IterationAndAlignment*

Semantics

The *IterationAndAlignment* entity is used to represent one pair of iteration and alignment information. The iteration may be specified either explicit or implicit, given by the *explicit-iteration* and *implicit-iteration* components, respectively.

- In case of implicit iteration, the iteration goes along with the geometric dimension of the pixel array specified in the abstract image structure definition of this image.

  Example - Given an image with consists of 720 pixels per line and 512 lines. Implicit iteration then means that the alignment takes place after each line, i.e. after 729 field elements.

- In case of explicit iteration, the iteration is defined explicitly without necessary reflecting the geometric dimensions of the image.

  Example - Given an image with consists of 720 pixels per line and 512 lines. An explicit iteration which is set to, for instance, 360 means that the alignment takes place twice per line, i.e. after 360 field elements.

Syntax

```
IterationAndAlignment ::= SEQUENCE
   {
   CHOICE
     {
     explicit-iteration    [0] INTEGER,
     implicit-iteration    [1] NULL
     },
   alignment               [2] Padding
   }
```

Constraints

For explicit alignment the number of field elements which result from the product of all *explicit-iteration* components must correspond to the number of pixel values specified by the type definition associated with the image.

For implicit alignment the number of *IterationAndAlignment* entities, which are contained in the *EncodingRules* entity shall match the number of dimensions of the image.

IIF syntax entity No. 1007 *Padding*

Semantics

The *Padding* entity is used to specify padding bits that occur within the field of pixel values. Two alternatives are provided.

- Using the *bit-pad* component, the number of bits of a pad which occur within the field of pixel values may be specified. No alignments to octet or world boundaries takes place. If the value is set to zero, no padding bits occur.
- Using the *octet-boundary* component, a pad of variable size which occurs within the field of pixel values in order to align the following element to a octet or word boundary, may be specified. The value specifies the number of octets to which the alignment shall take place. If the value is set to zero, no alignment takes place.

Syntax

```
Padding ::= CHOICE
   {
   bit-pad        [0] INTEGER (0..MAX),
   octet-boundary [1] INTEGER (0..MAX)
   }
```

Constraints

None.

IIF syntax entity No. 1008 — *ElementEncoding*

Semantics

The *ElementEncoding* entity is used to specify the pixel type dependent encoding method. Besides the pixel types boolean, non-negative integer, signed integer, fixed point, real complex and enumerated, also padding elements may be specified by choosing the *padding-element* component.

Syntax

```
ElementEncoding ::= CHOICE
   {
   boolean                  [0] BooleanEncoding,
   non-negative-integer     [1] UnsignedIntegerEncoding,
   signed-integer           [2] SignedIntegerEncoding,
   real                     [3] RealEncoding,
   fixed-point              [4] FixedPointEncoding,
   complex                  [5] ComplexEncoding,
   enumerated               [6] EnumeratedEncoding,
   padding-element          [7] Padding
   }
```

Constraints

Every type specific representation that has been chosen within the *PixelFieldEncoding* entity must match the pixel representation defined in the type definition associated with the image.

NOTE - In Case of the IIF-DF, this information is given by the ElementaryPixelStructure entity. The following table shows the data fields that may be used to represent a particular pixel type.

ElementaryPixelStructure	ElementEncoding
boolean	BooleanEncoding
non-negative-integer	UnsignedIntegerEncoding
signed-integer	SignedIntegerEncoding
real	RealEncoding, or FixedPointEncoding
fixed-point	RealEncoding, or FixedPointEncoding
complex	ComplexEncoding,
enumerated	EnumeratedEncoding,

IIF syntax entity No. 1009 BooleanEncoding

Semantics

The *BooleanEncoding* entity is used to determine the encoded representation of the boolean values that occur within the field of pixels values at the locations specified by the *EncodingRules* entity and its subentities.

In contrast to the other pixel data types, no degrees of freedom are provided for the encoded representation of boolean values.

Syntax

```
BooleanEncoding ::= NULL
```

Constraints

Refer to the *ElementEncoding* entity.

IIF syntax entity No. 1010 UnsignedIntegerEncoding

Semantics

The *UnsignedIntegerEncoding* entity is used to determine the encoded representation of the unsigned integer values that occur within the field of pixels values at the locations specified by the *EncodingRules* entity and its subentities.

The following rules hold true for the representation of these values:

- The *number-of-bits* component is used to specify the size of an integer in the number of bits.
- The low order (least significant) bit is called bit 0, the high-order (most significant) bit is called bit n-1 whereby n is the value of the *number-of-bits* component.
- The bits have to be interpreted according to the following equation:

$$\text{pixel-value} = \sum_{i=0}^{n-1} \text{bit}(i)*2^{i}$$

Syntax

```
UnsignedIntegerEncoding ::= number-of-bits INTEGER
```

Constraints

Refer to the *ElementEncoding* entity.

IIF syntax entity No. 1011 **SignedIntegerEncoding**

Semantics

The *SignedIntegerEncoding* entity is used to determine the encoded representation of the signed integer values that occur within the field of pixels values at the locations specified by the *EncodingRules* entity and its subentities.

The following rules hold true for the representation of these values:

- The *number-of-bits* component is used to specify the size of an integer in the number of bits.
- The low order (least significant) bit is called bit 0, the high-order (most significant) bit is called bit n-1 whereby n is the value of the *number-of-bits* component.
- The bits have to be interpreted according to the following equation:

$$\text{pixel-value} = \sum_{i=0}^{n-2} \text{bit}(i)*2^{i} - \text{bit}(n-1) * 2^{(n-1)}$$

Syntax

```
SignedIntegerEncoding ::= number-of-bits INTEGER
```

Constraints

Refer to the *ElementEncoding* entity.

IIF syntax entity No. 1012 RealEncoding

Semantics

The *RealEncoding* entity is used to determine the encoded representation of real values that occur within the field of pixels values at the locations specified by the *EncodingRules* entity and its subentities.

Real numbers are encoded according to the IEEE Standard P754. This standard defines two encoding precisions: *single* and *double*. Furthermore, it provides an *extended* encoding mode in addition to the *basic* one. In the later mode, no degrees of freedom exist. In the former mode, all degrees of freedom are controlled by the *IeeeExtendedEncoding* entity. The selection of the various modes is provided by the components *ieee-basic-single, ieee-basic-double, ieee-extended-single, ieee-extended-double.*

Syntax

```
RealEncoding ::= CHOICE
    {
    ieee-basic-single       [0] NULL,
    ieee-basic-double       [1] NULL,
    ieee-extended-single    [2] IeeeExtendedEncoding,
    ieee-extended-double    [3] IeeeExtendedEncoding
    }
```

Constraints

Refer to the *ElementEncoding* entity.

IIF syntax entity No. 1013 IeeeExtendedEncoding

Semantics

The *IeeeExtendedEncoding* entity provides control over parameters for the encoding of real values according to the IEEE P754 standard. The *precision* component specifies the number of significant bits. The *e-max* and *e-min* component determine the maximum and minimum exponent respectively. The *exponent-bias* and *exponent-width* component specify the exponent bias and the number of bits for the exponent, respectively. The *format-width* component determines the total number of bits.

For a detailed description of these parameters, refer to the IEEE Standard P754.

Syntax

```
IeeeExtendedEncoding ::= SEQUENCE
   {
   precision          [0] IMPLICIT INTEGER,
   e-max              [1] IMPLICIT INTEGER,
   e-min              [2] IMPLICIT INTEGER,
   exponent-bias      [3] IMPLICIT INTEGER,
   exponent-width     [4] IMPLICIT INTEGER,
   format-width       [5] IMPLICIT INTEGER
   }
```

Constraints

Refer to the *ElementEncoding* entity.

IIF syntax entity No. 1014 FixedPointEncoding

Semantics

The *FixedPointEncoding* entity is used to determine the encoded representation of the real values of limited range that occur within the field of pixels values at the locations specified by the *EncodingRules* entity and its subentities.

The following rules hold true for the representation of these values:

- The field elements are integer values which are encoded as specified by the *encoding-value* component.
- The original pixel values are determined by the equation:

  $$pixel\text{-}value = (encoding\text{-}value + offset) * factor$$

  whereby *encoding-value* represents the value of the field element, *offset* represents the value of the *offset* component and *factor* represents the value of the *factor* component.

Syntax

```
FixedPointEncoding ::= SEQUENCE
    {
    offset          [0] IMPLICIT REAL,
    factor          [1] IMPLICIT REAL,
    encoded-value   [2] SignedIntegerEncoding,
    }
```

Constraints

Refer to the *ElementEncoding* entity.