INTERNATIONAL STANDARD

ISO/IEC 12087-3

First edition
1995-02-15

Information technology — Computer graphics and image processing — Image Processing and Interchange (IPI) — Functional specification —

Part 3:
Image Interchange Facility (IIF)

Technologies de l'information — Infographie et traitement de l'image — Traitement de l'image et échange (IPI) — Spécification fonctionnelle —

Partie 3: Accessoires pour l'échange d'images (IIF)

Reference number
ISO/IEC 12087-3:1995(E)

Contents

Annexes

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 international 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 12087-3 was prepared by the Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO/IEC 12087 initially consists of three parts, under the general title *Information technology — Computer graphics and image processing — Image Processing and Interchange (IPI) — Functional specification*:

- *Part 1: Common architecture for imaging*
- *Part 2: Programmer's imaging kernel system application program interface*
- *Part 3: Image Interchange Facility (IIF)*

Annex A forms an integral part of this part of ISO/IEC 12087. Annexes B to G are for information only.

Introduction

ISO/IEC 12087-1 establishes the conceptual and architectural framework for ISO/IEC 12087. In particular, it defines the types of all image data objects, image-related data objects, and attributes that may be interchanged by means of the IPI-IIF.

ISO/IEC 12087-2 establishes the specification of the Programmer's Imaging Kernel System (IPI-PIKS).

ISO/IEC 12087-3 provides a data format specification and an application program interface specification. The IIF data format may be used for image data interchange in open, heterogeneous environments. It may also serve as a local file format for imaging applications, especially in conjunction with ISO/IEC 12087-2. In future, the IIF data format could be used by telecommunication standards. Examples are future versions of File Transfer, Access, and Management (FTAM), ISO/IEC 8571; the Message Oriented Text Interchange Systems (MOTIS), ISO/IEC 10021 (also known as Message Handling System (MHS), CCITT Recommendation X.400). Thus the IIF data format could become part of application-oriented OSI communications protocols.

Within the IIF data format (IIF-DF), compressed images may be specified and interchanged. For this purpose, the following standards are referenced:

- CCITT Recs. T.4 and T.6 (Facsimile)
- ISO/IEC 11544 (JBIG)
- ISO/IEC 10918 (JPEG)
- ISO/IEC 11172 (MPEG-1)

Image data streams that conform to the encoded representation of compressed image data specified by these standards may be included in the IIF-DF. For instance, a time series image can be represented as an array of time slices, each of which is encoded according to the JPEG Standard. Furthermore, the IIF-DF allows images to be represented through the combination of compressed parts with uncompressed parts. It is also possible to use multiple compression methods within a single IIF-DF-conformant image. For instance, a colour image can be represented as tiled images whereby some tiles are encoded according to the lossy mode of the JPEG Standard and others according to the lossless mode. For detailed information concerning compressed data streams and compression/decompression functionality, refer to 5.3.3 and 7.1.6, respectively.

There are various possibilities for interaction and data exchange between the IPI-PIKS domain and the IPI-IIF domain. Both domains are controlled by the application via application program interfaces (APIs). For a detailed description of the interworking between the IPI-PIKS and the IPI-IIF refer to clause 4 (the IPI-IIF architecture) and clause 7 (the IPI-IIF Gateway functionality). For a description of the relation between the types of objects that may be interchanged by means of the IPI-IIF and those types of objects that may be processed by the IPI-PIKS, refer to clause 6 (the profiles for the IIF data format). Refer also to ISO/IEC 12087-1.

Information technology — Computer graphics and image processing — Image Processing and Interchange (IPI) — Functional specification — Part 3: Image Interchange Facility (IIF)

1 Scope

This part of ISO/IEC 12087 facilitates the interchange of digital images. For this purpose, conceptual, architectural, and functional definitions of the Image Interchange Facility (IPI-IIF) are established. ISO/IEC 12087-3 consists of two major parts, the:

a) IIF data format (IIF-DF) definition (by means of a formal syntax, described according to the Abstract Syntax Notation One (ASN.1) -- refer to clause 5), and the

b) IIF Gateway definition (by means of a manual page description of the functionality of an Application Program Interface (API) -- refer to clause 7).

An IPI-IIF-conformant implementation has to fulfill the functionality specification of the IIF Gateway, as outlined in clause 7. Besides the IIF Gateway, there may be information processing systems (software such as parsers, generators, etc.) which read and/or write the IIF-DF.

The IPI-IIF is based on the definitions described in ISO/IEC 12087-1, the "Common Architecture for Imaging". The IPI-IIF, as a whole, may be characterized briefly as follows:

c) By means of the IIF data format and Gateway, image data objects and image-related data objects are transported to and from application environments.

d) By means of the full PIKS profile of the IPI-IIF data format (i.e., a format for data interchange between IPI-IIF and IPI-PIKS), image data objects and image-related data objects are imported to and exported from the Programmer's Imaging Kernel System (IPI-PIKS), defined in ISO/IEC 12087-2.

e) The IPI-IIF facilitates the storage of image data objects and image-related data objects in a variety of pre-defined storage modalities, including different periodicity organizations, such as pixel-interleaving or band-interleaving.

f) This part of ISO/IEC 12087 defines syntax of image data (and image-related data) streams. The encoding of IIF data types is defined in ISO/IEC 12089. See also 5.3.3.

g) The IPI-IIF supports a concept of standardized conformance profiles. Initially, three conformance profiles are defined within ISO/IEC 12087.

h) An IIF data stream may be stored in devices such as file systems. An IIF data stream may be interchanged and communicated in data networks (e.g., LANs and WANs) or in other data communication facilities. All low-level data storage and transfer is delegated, for instance, to the operating system of the target hardware.

i) The IIF Gateway performs compression and decompression of image data objects using standardized compression and decompression techniques. These techniques are referenced in this part of ISO/IEC 12087. See 1.4.5 and 5.3.3 and 7.5 for further definition.

j) The IIF Gateway is accessible via an API to perform image interchange functions. See clause 7 for a definition of IIF Gateway functionality.

Reference shall be made to this part of ISO/IEC 12087, and its definitions shall be employed, whenever images are interchanged, according to the IPI-IIF, among different imaging applications environments or among imaging devices. The IPI-IIF is applicable to scenarios requiring the interchange of digital images, as outlined in Annex C.

The use of the IIF data format as a superset of the functionality of most of the existing image interchange formats solves the problem of application-independent syntactical and semantical interpretation and understanding of image data.

The IPI-IIF is applicable to image interchange in and among different application domains. The following application areas have been considered:

- Medical imaging
- Remote sensing
- Publishing
- Industrial vision
- Computer graphics arts
- Computer animation
- Scientific visualization
- Mission planning
- Document processing
- Outdoor scene surveillance

The limiting of the IPI-IIF scope to certain application domains is a matter of profiling. This is treated in clause 6.

NOTE - Whether an image interchange format may also be regarded as a device format, depends on the (local) processing power of the device itself. Thus a conceptually "high-level" format which has become an industrial standard page description language for desktop electronic publishing, can be regarded as a device format. The IPI-IIF may well be considered a device format if, for instance, there is an IPI-IIF-compatible printer which is able to receive, process, and hardcopy an image according to the IPI-IIF. In the same sense, it is reasonable to design IPI-IIF-compatible image sources, e.g. IPI-IIF camera systems.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 12087. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 12087 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 2022:1986, *Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques.*

ISO/IEC 8613:1994, *Information processing systems - Text and office systems - Open Document Architecture (ODA) and Interchange Format (ODIF).*

ISO/IEC 8632:1992, *Information processing systems - 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 8879:1986, *Information processing systems - Text and office systems - Standard Generalized Markup Language (SGML).*

ISO/IEC 9069:1988, *Information processing systems - SGML support facilities - SGML Document Interchange Format (SDIF).*

ISO/IEC TR 10000-1:1990, *Information technology - Framework and taxonomy of International Standardized Profiles - Part 1: Framework.*

ISO/IEC TR 10000-2:1994, *Information technology - Framework and taxonomy of International Standardized Profiles - Part 2: Principles and taxonomy for OSI Profiles.*

ISO/IEC 10031-1:1991, *Information technology - Text and office systems - Distributed office application model - Part 1: General model.*

ISO/IEC 10031-2:1991, *Information technology - Text and office systems - Distributed office application model - Part 2: Distinguished object reference and associated procedures.*

ISO/IEC 10918-1:1994, *Information technology - Digital compression and coding of continuous-tone still images - Part 1: Requirements and guidelines.*

ISO/IEC 10918-2: To be published., *Information technology - Digital compression and coding of continuous-tone still images - Part 2: Compliance testing.*

ISO/IEC 11172-1:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbit/s - Part 1: Systems.*

ISO/IEC 11172-2:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbit/s - Part 2: Video.*

ISO/IEC 11172-3:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbit/s - Part 3: Audio.*

ISO/IEC 11544:1993, *Information technology - Coded representation of picture and audio information - Progressive bi-level image compression.*

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 12089:—[1)], *Information technology - Computer graphics and image processing - Encoding for the Image Processing and Interchange Standard (IPI) - Encoding for the Image Interchange Facility (IIF).*

CCITT Rec. G.711(1984), *Coding of analogue signals by pulse code modulation.*

CCITT Rec. G.721(1984), *32 Kbit/s Adaptive Differential Pulse Code Modulation (ADPCM).*

CCITT Rec. T.4(1988), *Standardization of Group 3 Facsimile Apparatus for Document Transmission.*

CCITT Rec. T.6(1988), *Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus.*

CCITT Rec. T.30(1988), *Procedures for Document Facsimile Transmission in the General Switched Telephone Network.*

NOTES

1 All normative references which are common to Parts 1 to 3 of ISO/IEC 12087 are included in ISO/IEC 12087-1. In ISO/IEC 12087-3, only the IIF-specific references are listed.

2 References to documents which are neither ISO/IEC Standards nor CCITT Recommendations are given in Annex G.

3 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.

1) To be published.

3 Definitions and Abbreviations

3.1. Definitions

For the purpose of this part of ISO/IEC 12087, the definitions given in ISO/IEC 12087-1 apply.

3.2. Abbreviations

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CGM	Computer Graphics Metafile
DCT	Discrete Cosine Transform
DOAM	Distributed Office Application Model
DOR	Distinguished Object Reference
FOD	Interchange Format and Representation Profile of Office Documents
FTAM	File Transfer, Access and Management
IIF-DF	Image Interchange Facility - Data Format
IPI	Image Processing and Interchange
IPI-CAI	IPI - Common Architecture for Imaging
IPI-IIF	IPI - Image Interchange Facility
IPI-PIKS	IPI - Programmer's Imaging Kernel System
JBIG	Joint Bi-level Image Experts Group
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
MHS	Message Handling System
MOTIS	Message-Oriented Text Interchange Systems
MPEG	Moving Pictures Experts Group
ODA	Open Document Architecture
OSI	Open Systems Interconnection
PER	Packed Encoding Rules
SDIF	SGML Document Interchange Format
SGML	Standard Generalized Markup Language
WAN	Wide Area Network

4 The IPI-IIF Architecture

4.1 The IPI-IIF Data Format and the IPI-IIF Gateway

As outlined in clause 1, the IPI-IIF consists of the specification of a data format and the specification of functionality. The data format is called Image Interchange Facility Data Format (IIF-DF). It is described in clause 5. Clause 6 describes conformance profiles for the IIF-DF.

The functional component of the IPI-IIF is called IPI-IIF Gateway. It is described in clause 7. The IPI-IIF Gateway functions are called by an application program via a specific API (Application Program Interface). Concerning data interchange, it provides functionality for

a) the import of image data to and export of image data from application, as well as

b) the import of image data to and export of image data from the IPI-PIKS.

Part of the IPI-IIF Gateway functionality deals with the differing complexity of the IPI-IIF data types (clause 6 of ISO/IEC 12087-1) and the IPI-PIKS data types (clause 5 of ISO/IEC 12087-1). The IPI-IIF data types are defined according to the (generic) IPI data types that are introduced in clause 4 and form a superset of the IPI-PIKS data types. The IPI-IIF data types support compound and heterogeneous image structures of arbitrary hierarchical organization whereas the IPI-PIKS data types support five-dimensional images with limited heterogeneity. The IPI-IIF Gateway provides a general mechanism - called attach/detach functionality - to combine simpler IIF structures into more complex ones, and extract simpler structures from more complex ones. Hence, this functionality represents the interface between the IPI-IIF data types (that are interchangeable by means of the IPI-IIF) and the IPI-PIKS data types (that can be processed by the IPI-PIKS, but also interchanged by the IPI-IIF).

The IPI-IIF Gateway function classes are:

c) Gateway control functionality: These functions are used to open and close the IPI-IIF Gateway, to inquire about its status, and to handle errors.

d) Import and export functionality: These functions allow for the import and export of image and image-related data to and from the IPI-IIF Gateway via the application program.

e) Parse and generate functionality: These functions allow for the translation between IPI-IIF data streams (according to the IPI-IIF data format) and IPI-IIF gateway-internal data structures that are accessible and manipulable via IPI-IIF Gateway functions of category f) and g), respectively.

f) Data structure access functionality: These functions allow for the access of parsed image data structures. This includes tree traverse, inquiry, put value, and get value functions.

g) Data structure manipulation functionality: These functions allow for the manipulation of image structures. Create, delete, attach, and detach functions are provided.

h) Compression and decompression functionality: Functions are provided for the compression and decompression of pixel fields according to the standards listed in section 1.4.3.

i) Application-oriented functionality: This category encompasses functions which perform at the same time multiple functional steps according to categories c) to h). Seen from the application, these functions are located on a higher (and thus more application-oriented) level.

The specification of the functions is given in clause 7.

4.2 Interworking between IPI-IIF Gateway and IPI-PIKS

The interworking (i.e. data flow and function calls) of the IIF Gateway and the IPI-PIKS for image data interchange is depicted in Figure 1. The diagram shows three domains, called the "application domain," the "IIF Gateway domain," and the "IPI-PIKS domain." These "domains" indicate whether a certain function or data structure is part of the IPI-PIKS, the IIF Gateway, or the application program, respectively. The arrows between (squeezed) ovals indicate the data flow. Possible interworking situations include:

a) The imaging application program uses only an IIF Gateway, but no IPI-PIKS: The application program may import and export image data which is interchanged using the IIF data stream. On the other hand, the application program has the capability to process image data by means of application-private imaging functions.

b) The imaging application program uses a non-IIF-capable IPI-PIKS and no IIF Gateway: In this case, the IPI-PIKS conformance profiles defined in ISO/IEC 12087-2 describe the import and export functionality of the IPI-PIKS.

c) The imaging application program uses an IIF-capable IPI-PIKS and no IIF Gateway: In this case, the standardized import and export functions of the IPI-PIKS, if called by the application program, produce image data according to the full PIKS profile or the foundation profile of the IPI-IIF data format. The application program will be able to read data streams that conform to the profile of the IPI-IIF data format. The application program, as well as the import and export function of the IPI-PIKS implementation, will have a parser and a generator for the IPI-IIF data format.

d) The imaging application program uses both an IPI-IIF Gateway and an IPI-PIKS: The import/export, parse/generate and data structure access and manipulation functions work according to a) and b), above. Further details are provided by the manual pages for these function descriptions (see clause 7).

Besides, it is possible that the application program does not use the standardized import and export functions, but explicitly ("manually") controls the interworking of the IPI-IIF Gateway and the IPI-PIKS by using only the import and export functions of the IPI-IIF Gateway and the IPI-PIKS utility functions. In this case, image data is transferred by the get_pixel and put_pixel functions from the IPI-PIKS to the application domain, and vice versa.

If both the IPI-IIF Gateway and the IPI-PIKS are developed and supplied as one software unit, the IPI-IIF Gateway and IPI-PIKS APIs may be linked together. In this case, the internal interworking between the IPI-IIF Gateway and the IPI-PIKS may be designed to be more efficient. In either case, private, non-standardized image data input and output may take place as shown by the dashed arrows in Figure 1. It is up to the application program to manage the reading and writing of private image file formats and convert the data into the IPI-IIF data format. See also 7.1.2 for the specification of low-level data exchange mechanisms of the IPI-IIF Gateway.

The IPI-PIKS functions relevant for the interworking of the IPI-PIKS with the IPI-IIF Gateway (defined in ISO/IEC 12087-2) are:

e) input_object: data transfer from the IPI-IIF Gateway to the IPI-PIKS.

f) output_object: data transfer from the IPI-PIKS to the IPI-IIF Gateway.

The IPI-PIKS can write and read image data streams only by means of the above-mentioned input_object/output_object functions. However, an application program may feed data to and from the IPI-PIKS with other utility functions, such as get_pixel, put_pixel, and import/export. For further details, see ISO/IEC 12087-2.

The functionality of both the IPI-IIF Gateway and the IPI-PIKS, as indicated above, is accessible via two separate APIs from the application program.

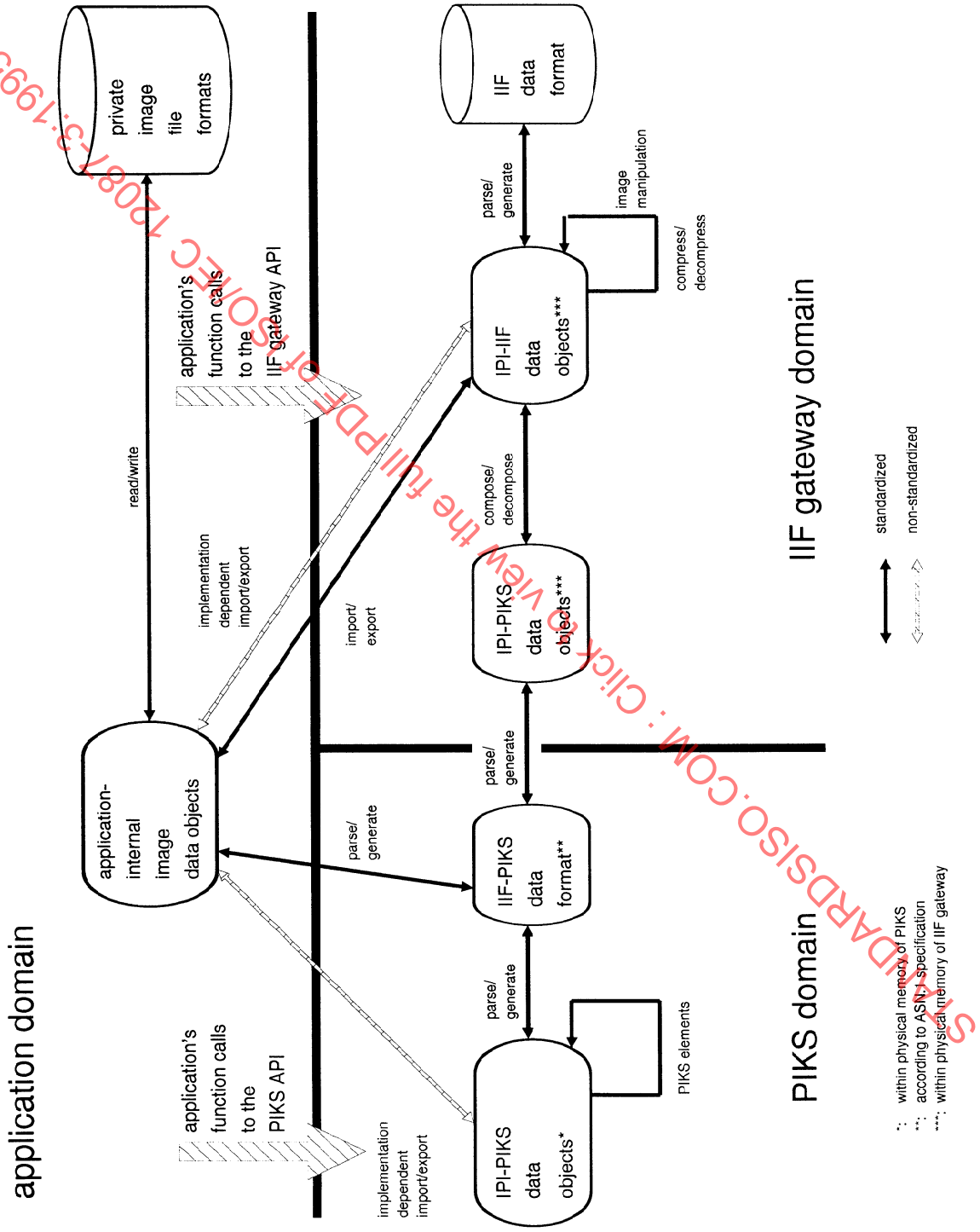

Figure 1 - Interworking of the IPI-IIF Gateway and the IPI-PIKS for image data interchange

5 The IIF data format (IIF-DF)

5.1 Basic features of the IIF-DF

This Clause gives an overview of:

- the kind of data objects which may be expressed by the IIF-DF,
- the kind of formal description method which has been chosen, and
- which encoding rules shall apply, in order to produce IIF-DF-conformant data streams.

Clause 5.3 defines the full IIF-DF syntax by giving the production rules for all data entities. Within this part of ISO/IEC 12087, the rules of the IIF-DF syntax are arranged in preorder traversal (depth-first) with the exception of image attributes, which are described after the non-image data types.

5.1.1 Objects that are expressed in the IIF-DF

The IIF-DF syntax allows description of a sequential format for all data objects that are structured according to the IPI-IIF data types, as defined in ISO/IEC 12087-1, clause 6. Since the IPI-PIKS data types (ISO/IEC 12087-1, clause 5) form a proper subset of the IPI-IIF data types, they are covered by the IIF-DF too. The names of the syntax entities correspond to the names of the data types of ISO/IEC 12087-1. The structure and semantics of these data entities are defined with reference to ISO/IEC 12087-1.

NOTE - ISO/IEC 12087-1 provides IPI-IIF data types and IPI-PIKS data types. Both are derived from the (generic) IPI data types. The IPI-IIF data types represent a very broad approach to image data modelling. The IPI-PIKS data types are oriented by the needs of image processing as defined in ISO/IEC 12087-1.

In addition to the data types defined in ISO/IEC 12087-1, the IIF-DF contains some data entities that specify the placement of pixel data, the ordering of elements of multi-dimensional arrays, and the coded representation of elementary data types, including compression algorithms.

5.1.2 Syntax notation

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 recognizable 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 semantical description of the IPI-IIF data format, 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. An example for a syntax entity is *BandRecord*. Syntax entities consist of a number of *component*s. According to ASN.1, the component labels begin with lower case letters. For example the *BandRecord* entity consists of the *number-of-bands* component, the *data-placement* component, and the *record-components* component. The type of each component is defined by reference to another syntax entity which is either described elsewhere in the IIF-DF syntax or within the ASN.1 standard. For example, the *data-placement* component is represented by the *DataPlacement* entity, which is described within the IIF-DF syntax. The *number-of-bands* component is represented by the *INTEGER* entity, which is an elementary syntax entity. For its definition, refer to ISO/IEC 8824. From the viewpoint

of the *BandRecord* entity, the *DataPlacement* entity is called *subentity*.

NOTE - Only those constructed ASN.1 data types for which a sequential order is defined are used within ISO/IEC 12087. Thus the IIF syntax always describes an unambiguously ordered data stream. However, concerning the placement and periodicity organization of pixel fields, the IIF-DF provides a maximum degree of flexibility.

5.1.3 Encoding of syntax entities

The encoding of IIF syntax entities is defined in ISO/IEC 12089.

NOTES

1 The encoding of IIF syntax entities as defined in ISO/IEC 12089 employs the Basic Encoding Rules for ASN.1 (BER), that are defined in ISO/IEC 8825. The encoding of a large pixel data field according to the BER can produce considerable space and processing time overhead when every pixel is expressed as an elementary ASN.1 data entity, consisting of a "tag" and a "length" field that precedes the "value" field. For this reason, ISO/IEC 12089 provides some methods on how to encode fields of pixel values as single entities in terms of ASN.1. This refers to the *ExternallyDefinedDataUnit* entity.

2 The BER provide a definite form and an indefinite form for the encoding of sequences of entities. Both are indicated by the keyword "SEQUENCE OF." In the definite form the number of content octets is given at the beginning of the sequence. In the indefinite form the number of content octets shall be calculated by parsing the whole sequence until the end-of-sequence token. Within the IIF-DF syntax, for some "SEQUENCE OF" constructs, a *"number-of-..."* component is provided. It specifies the number of entities (e.g., pixel values) that are present in the following sequence. This allows implementations to make use of the number of entities in advance (e.g., for memory allocation), regardless of what encoding method has been chosen.

5.1.4 Rules that are not formally expressed within the IIF syntax

There are rules and constraints within 5.3 that are needed in addition to the IIF syntax to define a proper IIF-DF. These rules and constraints are given by an accompanying text. They need to be checked within the parse and generate functions of the IPI-IIF Gateway, as illustrated in Figure 2.

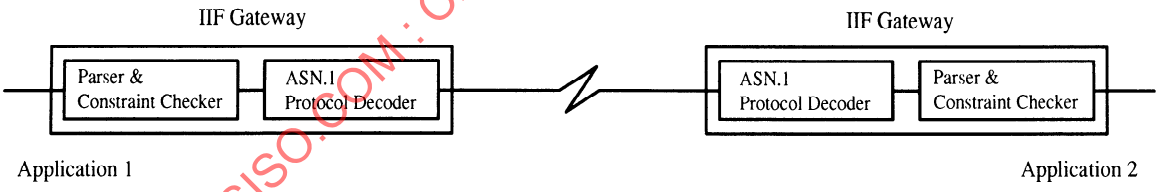

Figure 2 - IIF-DF parser and constraint checker within the IPI-IIF Gateway

NOTE - The IIF syntax is expressed using formal ASN.1 notation, but not all rules or constraints that determine the desired "correctness" of an IIF-DF-conformant data stream can be expressed in terms of this syntax. For example, there is no way to define a syntax that prevents the production of pixel arrays that contain more or fewer pixels than defined in the array's type definition. This deficiency, which is a result of the context-free grammar, is only one example. Other examples of constraints that need to be defined "on top" of the IIF syntax result from the desire for maximum flexibility in pixel data placement. In general, the number of constraints that need to be defined in addition to the syntax is a result of the tradeoff between flexibility and formal elegance. Furthermore, one may distinguish between various levels of constraints. Those constraints that determine, for instance, the completeness of pixel fields, are of a different nature as compared to constraints that deal with inconsistent colour attributes.

5.2 Structure of the IIF-DF syntax

This clause gives an overview of the structure of the IIF-DF syntax. The reader may study the following description of major IIF-DF syntax entities in conjunction with the three example images provided in Annex D and the IIF-DF syntax diagrams provided in Annex F.

5.2.1 Overall structure

The *FullDataFormat* entity is the top-most entity of the IIF-DF syntax tree. All other entities branch from there. There are two types of information contained in the *FullDataFormat* entity. First, there is information identifying the data as conforming to a particular version, conformance level, and application profile of the IIF. This information is contained in the *FormatDescriptor*, *Version*, and *Profile* entities. There is also a place to store some application-specific header information in the *ContentsHeader* entity. Second, there is the data content.

NOTE - Compound syntax entities may consist of an arbitrarily deep tree of sub-entities. Thus the term "something is contained (or stored) in a given syntax entity" does not imply that the information is contained in the named entity directly. It rather means that it is contained in either the entity or one of its subentities.

The data stored in the *ContentsBody* element can be images, image-related data, attributes, annotations, or basic data objects, as defined in ISO/IEC 12087-1. The overall structure of the IIF-DF syntax is shown in Figure 3.

NOTE - Figure 3 does not represent a complete syntax diagram. It is an abstract view of the IIF-DF syntax. Many details are left out. The solid boxes and arrows represent one possible instance of a syntax tree. The dashed boxes and arrows indicate potential alternatives according to the choices provided by the respective syntax entities. For an exact description of the IIF-DF syntax, refer to 5.3.

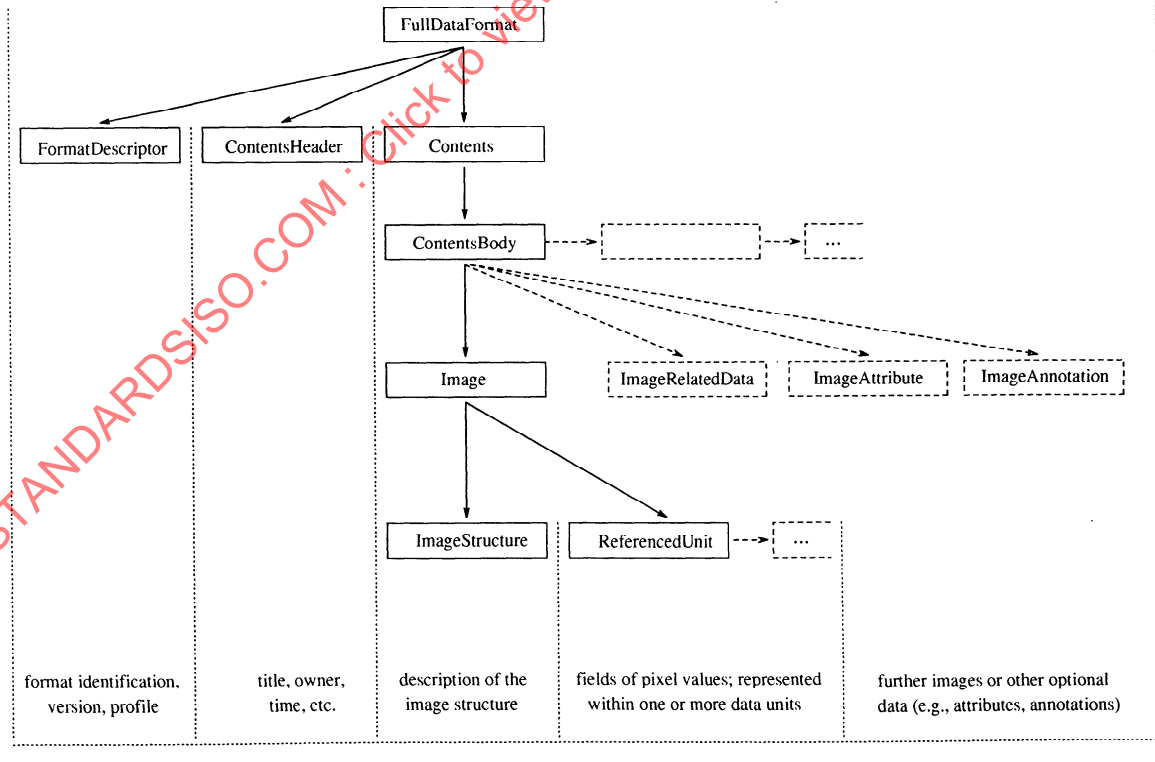

Figure 3 - Overall structure of the IIF-DF syntax.

5.2.2 Image structures

Images are stored in the *Image* entity and are split into structural information and data, called *image-structure* and *image-data*, respectively. As specified in the *ImageStructure* entity, an image can be either a compound image (such as arrays, records, lists or sets of other images) or a fundamental image. A fundamental image is an n-dimensional array of pixels which may consist of one or more bands. Images can have attributes and annotations associated with them.

The *CompoundImageArray* entity is used to specify an n-dimensional array of images; component images are accessed via an n-dimensional index. All the images stored in an array must have identical structures (including attributes, annotations, and image related data). The *CompoundImageRecord* entity is used to specify a record of images. The components are accessed via associated names, called *component-identifiers*. In contrast to arrays of images, the components are not required to have identical image structures. The *CompoundImageList* entity is used to specify a list of images of identical structure which are accessed via their position in the list. The *CompoundImageSet* entity is used to specify a set of images of identical structure. The syntax of the *ImageStructure* entity is recursive, i.e., the array elements, record components etc. are again of *ImageStructure* type.

The *FundamentalImageStructure* entity specifies the structure and sequential organization of an n-dimensional multiband array of pixels, along with associated image-related data, such as look-up tables, histograms, etc. It and its sub-entities provide enough flexibility to specify a wide range of pixel periodicity arrangements (e.g. pixel interleaved, band interleaved, tiled images, etc.). This is specified primarily by the *BandRecord* and *MetricArray* entities as described below.

The *BandRecord* entity is used to specify pixel arrays on a band-by-band basis; this entity is used to specify a band-interleaved image. The *MetricArray* entity is used to specify arrays of pixels or tiles. Pixels can be represented as a collection of bands in the *PixelStructure* entity. Thus, pixel-interleaved images are specified using a *MetricArray* of *PixelStructures*. The syntax of the *MetricArray* and *BandRecord* entities allow for arbitrary recursions. Thus, tiled images may be specified by constructing an array of arrays using the *MetricArray* entity for both hierarchical levels. Band-interleaved tiles are specified by a *MetricArray* containing *BandRecords*. Pixel-interleaved tiles are specified by a *MetricArray* containing *MetricArrays*. The recursive organization within the *FundamentalImageStructure* entity is illustrated in Annex F.

Entities specifying the structure of arrays in the IIF, such as *MetricArrays* and *CompoundImageArrays*, contain components specifying the number of dimensions of the array, the size of the array along each dimension, and a unique identifier for every dimension. The *Serialization* entity which is also part of every array description, indicates which axis varies fastest, second fastest, and so on within the sequence of pixel values.

The various hierarchical levels of an image structure described by the IIF-DF syntax are shown in Figure 4.

NOTE - Figure 4 does not represent a complete syntax diagram. It is an abstract view of the IIF-DF syntax. Many details are left out. The solid boxes and arrows represent one possible instance of a syntax tree. The dashed boxes and arrows indicate potential alternatives according to the choices provided by the respective syntax entities. For an exact description of the IIF-DF syntax, refer to 5.3.

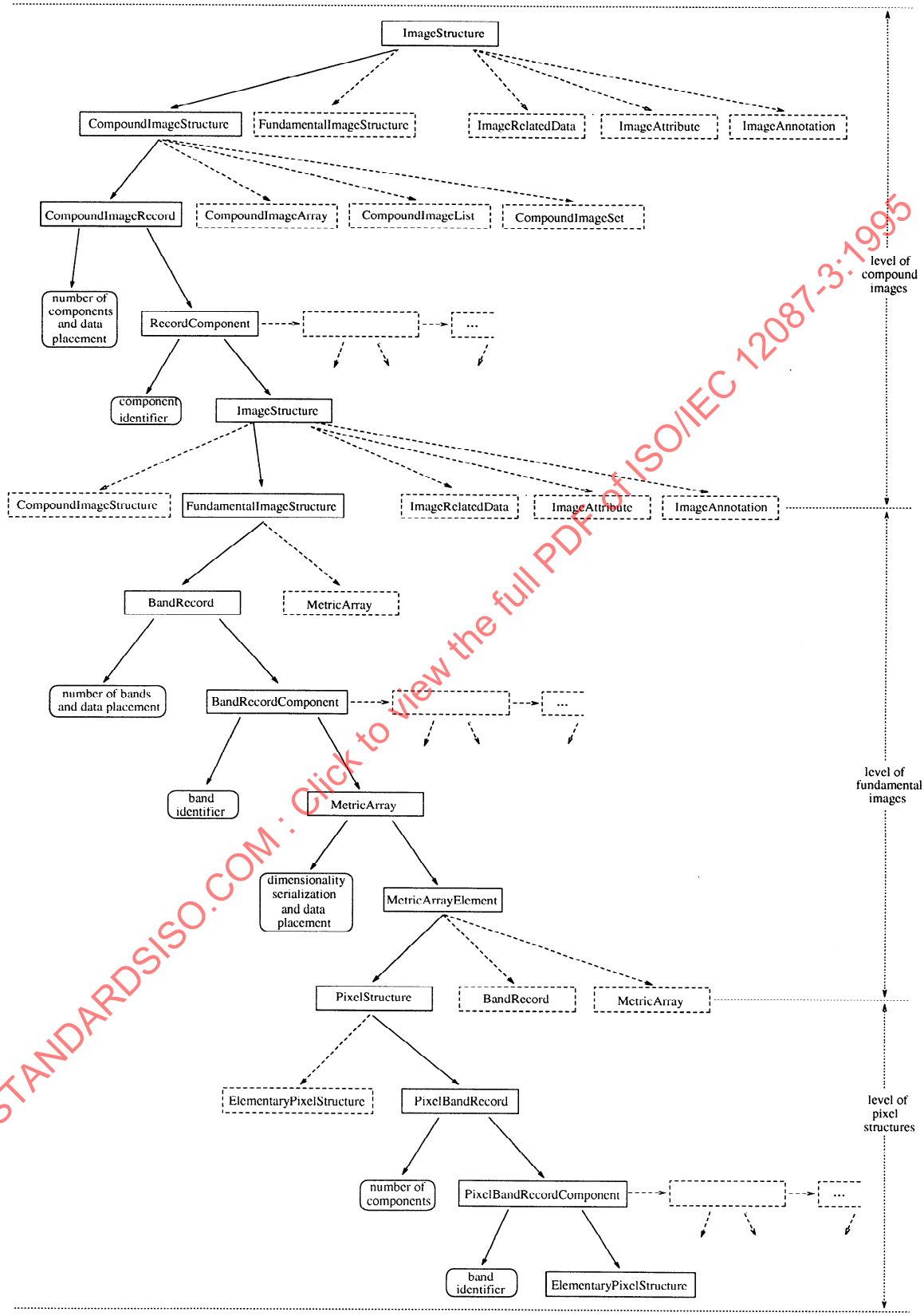

Figure 4 - Representation of image structures within the IIF-DF syntax.

5.2.3 Placement of pixel fields

As described above, the *image-structure* component of the *Image* entity and its subentities only contain information describing the image structure. The associated fields of pixel values are stored in the *image-data* component. Various modes of data placement and various pixel field encodings are permitted. The "atomic portion" of pixel data (a set of data that is always stored as one unit) is called a *ReferencedUnit* (see Figure 3).

The *DataPlacement* entity associated with each of the image structure entities is used to indicate whether the pixel values associated with the image structure are stored together in a single entity, called *ReferencedUnit*, or whether they are distributed into multiple entities. There are three cases that must be delineated.

a) The data associated with the structure is divided into multiple *ReferencedUnit* entities.

b) The data associated with the structure is completely contained in a single *ReferencedUnit* entity; additionally this entity does not contain any data from other structures.

c) The data associated with this structure is contained in a single *ReferencedUnit* entity along with data from other structures.

In either case, the pixel fields of an image are placed at an arbitrary position behind the image structure tree. All of these data sets are modelled in the syntax as a *SEQUENCE OF ReferencedUnit*.

Each *ReferencedUnit* starts with a label that indicates what portion of the pixel values of the image it contains. In principle, the label specifies the path from the root of the image structure tree down to the subentity which describes the structure of the pixel data contained in the referenced unit. For a definition of the *ReferencedUnit* entity and the construction of labels refer to 5.3.3.

NOTE - The labelling of referenced units and all other labellings within the IIF-DF as well are based on semantical identifiers. Using ASN.1 as syntax notation, the syntactical definition of the IIF-DF becomes independent of encoding rules. Thus, mechanisms such as byte counts or byte offsets are excluded.

5.2.4 Encoding of pixel fields

For the encoded representation of pixel fields, four choices are provided within the *SingleDataUnit* entity:

1) The *BuiltinEncodedDataUnit* entity uses elementary ASN.1 syntax entities, such as INTEGER and REAL to encode every pixel value.

2) The *ExternallyDefinedDataUnit* entity uses techniques defined in ISO/IEC 12089 to store the pixels in a single ASN.1 OCTET STRING.

3) The *CompressedDataUnit* entity allows standard compression techniques, such as JBIG and JPEG to be used to encode the pixel information.

4) The *RegisteredDataUnit* entity permits the transportation of registered encodings through the use of an ASN.1 ANY type.

Refer to 5.3.3 for detailed information on each of the above-mentioned encoding alternatives.

5.2.5 Attributes, annotations, and image-related data

The IIF-DF supports the representation of all image attributes and image-related data types that are defined in ISO/IEC 12087-1.

ImageAttribute entities can be attached to image structures or can be represented independently.

MetricDescription entities are used to specify the size and type of coordinate systems from which the image was sampled. *ChannelCharacteristics* entities are used to specify the transfer characteristics, precision, and semantics of the pixel values. The *ColourRepresentation* entity is used to specify colour spaces, both standard and non-standard. Finally, a freeform attribute is provided to allow application specific attributes to be represented.

Within the *MetricDescription* entities identifiers are used to refer to specific image dimensions which are defined within the image structure declaration. Within the *ChannelCharacteristics* and *ColourRepresentation* entities, the same identifier-based mechanism is used to refer to specific image bands. The former identifiers are called *dimension-identifiers*, the latter *band-identifiers*.

ImageRelatedData entities can also be attached to images or represented independently. Types of image related data include look up tables, histograms, regions of interest, neighbourhoods, static arrays, feature lists, and value bounds collections. These data types are used by ISO/IEC 12087-2.

ImageAnnotation entities can also be attached to images or represented independently. Types of pre-registered image annotations include: plain text, structured text according to SGML and ODA, graphics contents according to CGM, and audio data streams according to CCITT Recommendations G.711 and G.721. The exact references are given in 5.3.6. Additional application-specific annotation types supported by the ASN.1 type ANY are subject to registration as defined in 6.2.

5.3 Syntax entities of the IIF-DF

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, with the exception of image attributes, which are described after the non-image data types. The syntax rules, as well as the related semantics and constraints, are divided into the following subclauses:

5.3.1 Entities for the description of the entire IIF-DF

IIF module declaration *IIFDataFormat*
IIF syntax entity No. 001 *FullDataFormat*
IIF syntax entity No. 002 *FormatDescriptor*
IIF syntax entity No. 003 *Version*
IIF syntax entity No. 004 *Profile*
IIF syntax entity No. 005 *ContentsHeader*
IIF syntax entity No. 006 *CharacterString*
IIF syntax entity No. 007 *SpecialCharacterString*
IIF syntax entity No. 008 *Contents*
IIF syntax entity No. 009 *ContentsElement*
IIF syntax entity No. 010 *ContentsBody*

5.3.2 Entities for the description of images

IIF syntax entity No. 101 *Image*
IIF syntax entity No. 102 *ImageStructure*
IIF syntax entity No. 103 *CompoundImageStructure*
IIF syntax entity No. 104 *CompoundImageArray*
IIF syntax entity No. 105 *Dimensionality*
IIF syntax entity No. 106 *DimensionDescription*
IIF syntax entity No. 107 *Identifier*
IIF syntax entity No. 108 *Serialization*
IIF syntax entity No. 109 *DataPlacement*
IIF syntax entity No. 110 *CompoundImageRecord*
IIF syntax entity No. 111 *RecordComponent*
IIF syntax entity No. 112 *CompoundImageList*
IIF syntax entity No. 113 *CompoundImageSet*
IIF syntax entity No. 114 *FundamentalImageStructure*
IIF syntax entity No. 115 *BandRecord*
IIF syntax entity No. 116 *BandRecordComponent*
IIF syntax entity No. 117 *MetricArray*
IIF syntax entity No. 118 *MetricArrayElement*
IIF syntax entity No. 119 *PixelStructure*
IIF syntax entity No. 120 *ElementaryPixelStructure*
IIF syntax entity No. 121 *PixelBandRecord*
IIF syntax entity No. 122 *PixelBandRecordComponent*

5.3.3 Entities for the description of the representation of pixel values

IIF syntax entity No. 201 *ReferencedUnit*
IIF syntax entity No. 202 *DataUnit*
IIF syntax entity No. 203 *SubdividedDataUnit*
IIF syntax entity No. 204 *SingleDataUnit*
IIF syntax entity No. 205 *BuiltinEncodedDataUnit*
IIF syntax entity No. 206 *BuiltinValue*
IIF syntax entity No. 207 *ComplexValue*
IIF syntax entity No. 208 *ExternallyDefinedDataUnit*
IIF syntax entity No. 209 *CompressedDataUnit*
IIF syntax entity No. 210 *RegisteredDataUnit*
IIF syntax entity No. 211 *ExternalReference*
IIF syntax entity No. 212 *ExternalAddress*

5.3.4 Entities for the description of image-related data

IIF syntax entity No. 301 *ImageRelatedData*
IIF syntax entity No. 302 *Histogram*
IIF syntax entity No. 303 *PartitionClass*
IIF syntax entity No. 304 *LookUpTable*
IIF syntax entity No. 305 *RegionOfInterest*
IIF syntax entity No. 306 *BooleanArray*
IIF syntax entity No. 307 *Ellipse*
IIF syntax entity No. 308 *IntervalND*
IIF syntax entity No. 309 *Interval1D*
IIF syntax entity No. 310 *CoordinateND*
IIF syntax entity No. 311 *SetOfCoordinates*
IIF syntax entity No. 312 *NeighbourhoodArray*
IIF syntax entity No. 313 *IndexND*
IIF syntax entity No. 314 *StaticArray*
IIF syntax entity No. 315 *FeatureList*
IIF syntax entity No. 316 *CoordinateAndFeature*
IIF syntax entity No. 317 *ValueBoundsCollection*
IIF syntax entity No. 318 *TransformationMatrix*
IIF syntax entity No. 319 *PixelRecord*
IIF syntax entity No. 320 *PixelRecordComponent*
IIF syntax entity No. 321 *Tuple*

5.3.5 Entities for the description of image attributes

IIF syntax entity No. 401 *ImageAttribute*
IIF syntax entity No. 402 *MetricDescription*
IIF syntax entity No. 403 *DimensionMapping*
IIF syntax entity No. 404 *MeasurementUnit*
IIF syntax entity No. 405 *DeltaVector*
IIF syntax entity No. 406 *MetricTransformation*
IIF syntax entity No. 407 *Domain*
IIF syntax entity No. 408 *ChannelCharacteristics*

IIF syntax entity No. 409 *CompandorDescription*
IIF syntax entity No. 410 *ColourRepresentation*
IIF syntax entity No. 411 *StandardizedSpace*
IIF syntax entity No. 412 *CIEXYZSpace*
IIF syntax entity No. 413 *CIEYxySpace*
IIF syntax entity No. 414 *CIEUVWSpace*
IIF syntax entity No. 415 *CIEYuvSpace*
IIF syntax entity No. 416 *CIELabSpace*
IIF syntax entity No. 417 *CIELuvSpace*
IIF syntax entity No. 418 *CIEXYZCoordinate*
IIF syntax entity No. 419 *LinearRGBSpace*
IIF syntax entity No. 420 *GammaRGBSpace*
IIF syntax entity No. 421 *YIQColourSpace*
IIF syntax entity No. 422 *YUVColourSpace*
IIF syntax entity No. 423 *YCbCrColourSpace*
IIF syntax entity No. 424 *NonStandardizedSpace*
IIF syntax entity No. 425 *NonStandardizedRGB*
IIF syntax entity No. 426 *NonStandardizedIHS*
IIF syntax entity No. 427 *Primaries*
IIF syntax entity No. 428 *CIExyCoordinate*
IIF syntax entity No. 429 *NonStandardizedCMY*
IIF syntax entity No. 430 *NonStandardizedCMYK*
IIF syntax entity No. 431 *NonStandardizedNBand*
IIF syntax entity No. 432 *ColourBand*
IIF syntax entity No. 433 *TestColour*
IIF syntax entity No. 434 *PIKSControl*

5.3.6 Entities for the description of image annotations

IIF syntax entity No. 501 *ImageAnnotation*
IIF syntax entity No. 502 *Location*

5.3.7 Entities for the description of basic data objects

IIF syntax entity No. 601 *BasicDataObject*
IIF syntax entity No. 602 *BasicDataType*
IIF syntax entity No. 603 *CompoundDataType*
IIF syntax entity No. 604 *BasicArray*
IIF syntax entity No. 605 *BasicRecord*
IIF syntax entity No. 606 *BasicRecordComponent*
IIF syntax entity No. 607 *BasicList*
IIF syntax entity No. 608 *BasicSet*
IIF syntax entity No. 609 *ElementaryDataType*

5.3.1 Entities for the description of the entire IIF-DF

IIF module declaration ***IIFDataFormat***

Semantics

IIFDataFormat is the name of the IIF-DF module. Besides the full syntax (given by the *FullDataFormat* entity), the module also exports the *Image*, *ImageRelatedData*, *ImageAttribute*, *ImageAnnotation*, and *BasicDataObject* entities. This provides other ASN.1-notated applications with direct access to these sub-objects.

Two entities are imported from other modules: the *PixelFieldEncoding* entity is imported from ISO/IEC 12089 for the efficient representation of pixel fields and the *DOR* entity is imported from ISO/IEC 10031 for the representation of *Distinguished Object References* (DOR). For a description of the DOR, refer to 5.3.3

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 5.3.

The *FullDataFormat* entity is described here in 5.3.1. Its subentities refer to all other subclauses of 5.3. The *Image* entity is described in 5.3.2. Its subentities refer to 5.3.3 (for the placement of pixel data) and 5.3.4, 5.3.5, and 5.3.6 (for image-related data, attributes and annotations that are directly attached to image objects). The *ReferencedUnit*, *ImageRelatedData*, *ImageAttribute*, *ImageAnnotation* and *BasicDataObject* entities are described in 5.3.3, 5.3.4, 5.3.5, 5.3.6, and 5.3.7, respectively.

Syntax

```
IIFDataFormat {iso standard 12087 iif (3) df (0)} DEFINITIONS ::=

BEGIN

EXPORTS

   FullDataFormat,                                          -- No. 001
   Image,                                                   -- No. 101
   ReferencedUnit,                                          -- No. 201
   ImageRelatedData,                                        -- No. 301
   ImageAttribute,                                          -- No. 401
   ImageAnnotation,                                         -- No. 501
   BasicDataObject;                                         -- No. 601

IMPORTS

   PixelFieldEncoding FROM
   IIFEncoding {iso standard 12089 iif-encoding (1)}
   DOR FROM
   DOR-definitions {joint-iso-ccitt 10031 reference-definition (0)};

<<declarations>>

END
```

IIF syntax entity No. 001 ***FullDataFormat***

and

IIF syntax entity No. 002 ***FormatDescriptor***

Semantics

The *FullDataFormat* entity is the root of the entire IIF-DF syntax. It contains the *format-descriptor*, *contents-header*, and *contents* components.

The *FormatDescriptor* entity contains some meta-information: a *self-identification* component that carries the official identifier of the IPI-IIF syntax, the *profile* component which allows for setting a conformance profile and the *version* component that allows for a textual description of the syntax version.

All image data, image-related data, image annotations, and image attributes are contained in the *contents* component. The *ContentsHeader* entity provides some basic text fields to verbally describe the contents.

Syntax

```
FullDataFormat ::= [APPLICATION 0] IMPLICIT SEQUENCE
   {
   format-descriptor    [0] IMPLICIT FormatDescriptor,          -- No. 002
   contents-header      [1] IMPLICIT ContentsHeader OPTIONAL,   -- No. 005
   contents             [2] IMPLICIT Contents                   -- No. 008
   }


FormatDescriptor ::= SEQUENCE
   {
   self-identification  [0] IMPLICIT OBJECT IDENTIFIER DEFAULT
                            { iso standard 12087 iif (3) df (0)},
                                                                -- No. 706
   version              [1] IMPLICIT Version OPTIONAL,          -- No. 003
   profile              [2] IMPLICIT Profile OPTIONAL           -- No. 004
   }
```

Constraints

If the *Profile* entity is missing, a full syntax without any further constraint set is assumed.

IIF syntax entity No. 003 *Version*

Semantics

The *Version* entity stands for a textual description of the syntax version to which the given data stream conforms. The name of the syntax version is given by the *standard* component and the publication date of the syntax (either the publication date of ISO/IEC 12087-3 or the publication date of a revision of ISO/IEC 12087-3) is given by the *publication-date* component.

NOTE - The specification method for the *publication-date* component conforms to the method that is used within the ISO/IEC 8613 (ODA/ODIF).

Syntax

```
Version ::= SEQUENCE
  {
  standard         [0] IA5String DEFAULT "ISO/IEC 12087-3",    -- No. 722
  publication-date [1] IA5String                               -- No. 722
  }
```

Constraints

None.

IIF syntax entity No. 004 *Profile*

Semantics

The *Profile* entity stands for the description of profiles, specifying that the IIF-DF is restricted to a certain subset. One of the following predefined conformance profiles may be chosen: *full profile*, *full PIKS profile*, and *foundation profile*.

For a definition of these profiles refer to clause 6. Additional application profiles are subject to registration as defined in 6.2.

Syntax

```
Profile ::= IA5String                                    -- No. 702
```

Constraints

For the *Profile* entity, only the values *"full profile"*, *"full PIKS profile"*, and *"foundation profile"* and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 005 *ContentsHeader*

Semantics

The *ContentsHeader* entity provides some common information about the contents of the IIF data stream. No further semantics are defined for the convention of the *title*, *owner*, *date-and-time*, and *message* components. Also, no semantics are defined for the *application-data* component which can be represented using any ASN.1 type.

NOTE - Information fields, such as "processing platform" or "acquisition process" are regarded as too application-specific to be included in this header as a separate entity. This kind of information can either be put into the message field as readable text or handled as an image annotation or image attribute (using either one of the built-in fields or a freeform field).

EXAMPLE - The *application-data* component (as well as any other component that is typed with the ASN.1 type ANY) could be structured by an application in the following way:

```
ApplicationData ::= SET OF
   {
   SEQUENCE
      {
      tag     [0] IMPLICIT INTEGER,
      value   [1] IMPLICIT OCTET STRING
      }
   }
```

Syntax

```
ContentsHeader ::= SEQUENCE
   {
   title             [0] CharacterString OPTIONAL,          -- No. 006
   owner             [1] CharacterString OPTIONAL,          -- No. 006
   date-and-time     [2] GeneralizedTime OPTIONAL,          -- No. 724
   message           [3] CharacterString OPTIONAL,          -- No. 006
   application-data  [4] ANY OPTIONAL
   }
```

Constraints

None.

IIF syntax entity No. 006 ***CharacterString***

and

IIF syntax entity No. 007 ***SpecialCharacterString***

Semantics

The *CharacterString* entity is mainly meant for the description of a text readable by a person. It provides two alternatives. The *standard-characters* component is used for the representation of ASCII characters according to the International Alphabet No. 5. The *special-characters* component is used for all other character sets.

The *SpecialCharacterString* entity supports all different kinds of national and international character sets. The *character-set-escape* component is used to indicate the character set in use. This is done via the internationally standardized escape sequences according to ISO 2022. The *characters* component contains the character string.

Syntax

```
CharacterString ::= [APPLICATION 1] CHOICE
   {
   standard-characters    [0] IMPLICIT IA5String,                -- No. 722
   special-characters     [1] IMPLICIT SpecialCharacterString    -- No. 007
   }

SpecialCharacterString ::= SEQUENCE
   {
   character-set-escape   [0] IMPLICIT OCTET STRING,             -- No. 704
   characters             [1] IMPLICIT OCTET STRING              -- No. 704
   }
```

Constraints

None.

IIF syntax entity No. 008 ***Contents***

and

IIF syntax entity No. 009 ***ContentsElement***

Semantics

The *Contents* entity consists of a sequence of *ContentsElement* entities.

The *ContentsElement* entity provides a sequence of *prolog*, *body*, and *epilog* components. The optional *prolog* and *epilog* components do not carry any semantics in the context of ISO/IEC 12087. The only reason for their presence in this syntax entity in combination with the recursive definition of the *ContentsBody* entity is to prepare the introduction of a segmentation concept, associated with type definitions and segment-specific attributes which may be given in a separate Amendment to ISO/IEC 12087.

Syntax

```
Contents ::= SEQUENCE OF ContentsElement

ContentsElement ::= SEQUENCE
   {
   prolog    [0] ANY OPTIONAL,
   body      [1] IMPLICIT SEQUENCE OF ContentsBody,          -- No. 010
   epilog    [2] ANY OPTIONAL
   }
```

Constraints

The *body* component shall contain only one *ContentsBody* entity.

IIF syntax entity No. 010 *ContentsBody*

Semantics

The *ContentsBody* entity stands for the description of iconic and non-iconic data. The following components may be selected:

- the *image* component contains image structure information and associated pixel data;
- the *image-related data* component contains data that conform to one of the image-related data types as defined in ISO/IEC 12087-1.
- the *image-attributes* component contains data that conform to one of the attribute types as defined in ISO/IEC 12087-1.
- the *image-annotations* component contains data that conform to one of the attribute types as defined in ISO/IEC 12087-1.
- the *basic-data-component* contains data that are structured according to a basic data type as defined in ISO/IEC 12087-1.

Image annotations and image-related data are provided in two ways: bound to images, using the corresponding subentities within the *Image* entity, or separate from images using the *ContentsBody* entity as stated above. The latter way allows for the exchange of non-iconic parameters from and to the IPI-PIKS without any image data.

Syntax

```
ContentsBody ::= CHOICE
    {
    image                [0] Image,                    -- No. 101
    image-related-data   [1] ImageRelatedData,         -- No. 301
    image-attribute      [2] ImageAttribute,           -- No. 401
    image-annotation     [3] ImageAnnotation,          -- No. 501
    basic-data-object    [4] BasicDataObject           -- No. 601
    }
```

Constraints

None.

5.3.2 Entities for the description of images

IIF syntax entity No. 101 ***Image***

Semantics

The *Image* entity consists of the components *image-structure* and *image-data*.

The *image-structure* component stands for the description of the structure of one or more image(s) that is/are contained in the *Image* entity. The structure can conform either to a compound or fundamental image data type. It is expressed as a tree whose root is the *ImageStructure* entity and whose leaves represent elementary data types.

The *image-data* component contains a sequence of data fields to which the image structure sub-components refer. The data fields are called the *ReferencedUnit*.

Syntax

```
Image ::= SEQUENCE
   {
   image-structure     [0] IMPLICIT ImageStructure,            -- No. 102
   image-data          [1] IMPLICIT SEQUENCE OF ReferencedUnit -- No. 201
   }
```

Constraints

Refer to constraints related to the *ReferencedUnit* entity and the *DataPlacement* entity.

IIF syntax entity No. 102 *ImageStructure*

Semantics

The *ImageStructure* entity stands for the description of IPI-IIF data types as defined in ISO/IEC 12087-1. The image data type can be either a *compound-structure* or a *fundamental-structure*.

NOTES

1 The following fields of the "representation attribute", defined in ISO/IEC 12087-1, are represented in the IIF-DF by the *ImageStructure* entity and its subentities: *image size*, *band data type*, *image structure code*.

2 This entity and its subentities do not contain pixel values. They are used to specify the structure of image data according to the data types defined in ISO/IEC 12087-1. (The pixel values are contained in the *ReferencedUnit* entities.)

The semantical distinction between compound image structures and fundamental image structures is as follows:

- A compound image structure consists of arrays, records, or lists, where the dimensions of the arrays do not refer to a coordinate space that shall be used for data presentation. The leaves of a compound image structure express the structure of fundamental images.
- A fundamental image structure consists of arrays and records, where the dimensions of all arrays refer to the same coordinate space, and the record components refer to the bands (of a multi-band image). The arrays of the bands may differ in size and element structure but not in dimensionality. The leaves of a fundamental image structure express the type of elementary pixel values.

An arbitrary number of attributes and image-related data may be attached, using the *image-attributes* and *image-related-data* components. Image annotations may be attached using the *image-annotations* component. The *processing-history* component serves as a textual description of the image's processing history.

Syntax

```
ImageStructure ::= SEQUENCE
   {
   structure-type CHOICE
      {
      compound-structure       [4] CompoundImageStructure,      -- No. 103
      fundamental-structure    [5] FundamentalImageStructure    -- No. 114
      },
   image-attributes      [0] IMPLICIT SEQUENCE OF
                             ImageAttribute OPTIONAL,          -- No. 401
   image-related-data    [1] IMPLICIT SEQUENCE OF
                             ImageRelatedData OPTIONAL,        -- No. 301
   image-annotations     [2] IMPLICIT SEQUENCE OF
                             ImageAnnotation OPTIONAL,         -- No. 501
   processing-history    [3] IMPLICIT CharacterString OPTIONAL -- No. 006
   }
```

Constraints

None.

IIF syntax entity No. 103 *CompoundImageStructure*

Semantics

The *CompoundImageStructure* entity stands for the description of compound image data types, as defined in ISO/IEC 12087-1. It can be an array, a record, a list, or a set.

Syntax

```
CompoundImageStructure ::= CHOICE
    {
    compound-image-array     [0] CompoundImageArray,          -- No. 104
    compound-image-record    [1] CompoundImageRecord,         -- No. 110
    compound-image-list      [2] CompoundImageList,           -- No. 112
    compound-image-set       [3] CompoundImageSet             -- No. 113
    }
```

Constraints

None.

IIF syntax entity No. 104 *CompoundImageArray*

Semantics

The *CompoundImageArray* entity stands for the description of a compound image data type as defined in ISO/IEC 12087-1. The dimensionality of the array, including dimension identifiers and index ranges for every dimension is given by the *dimensionality* component. The sequential order of the array elements is determined within the *serialization* component. The placing of the data values that are associated with this array description is given within the *data-placement* component. The structure of the array elements is given by the *element-structure* component.

NOTE - The *CompoundImageArray* entity contains the *ImageStructure* entity. The *ImageStructure* entity, in turn, contains the *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, and *CompoundImageSet* entities. With this syntactical, recursive construction, image data types of arbitrary complexity may be constructed.

Syntax

```
CompoundImageArray ::= SEQUENCE
    {
    dimensionality          [0] IMPLICIT Dimensionality,      -- No. 105
    serialization           [1] IMPLICIT Serialization,       -- No. 108
    data-placement          [2] IMPLICIT DataPlacement,       -- No. 109
    element-structure       [3] IMPLICIT ImageStructure       -- No. 102
    }
```

Constraints

None.

IIF syntax entity No. 105 *Dimensionality*

Semantics

The *Dimensionality* entity contains the array dimensionality given by the *number-of-dimensions* component. For every dimension, a description, given by the *dimension-descriptions* component, is required.

Syntax

```
Dimensionality ::= SEQUENCE
    {
    number-of-dimensions      [0] IMPLICIT INTEGER (1..MAX),      -- No. 702
    dimension-descriptions    [1] IMPLICIT SEQUENCE OF
                                      DimensionDescription -- No. 106
    }
```

Constraints

The number of *DimensionDescription* entities contained in the sequence shall equal the value of the *number-of-dimensions* component.

IIF syntax entity No. 106 ***DimensionDescription***

and

IIF syntax entity No. 107 ***Identifier***

Semantics

The *DimensionDescription* entity contains a description of one of the array's dimensions. It consists of the *dimension-identifier*, *lower-boundary*, and *upper-boundary* components. The latter two components give the range of the dimension in terms of a lower and an upper boundary. The *upper-boundary* component is optional in order to support an indefinite dimension description. The reader is referred to the *Serialization* entity for the constraints that apply for the indefinite form.

EXAMPLE - An indefinite dimension description may be used to create a moving image consisting of a sequence of frames, where the total number of frames is not known at the time the first frame is being created.

The *Identifier* entity allows for the representation of a dimension identifier in terms of the ASN.1 elementary type IA5String. It is used to associate dimension-specific attributes with image dimensions having the same identifier. The following identifiers have predefined semantics as described in ISO/IEC 12087-2:

"x"	horizontal dimension
"y"	vertical dimension
"z"	depths dimension
"t"	temporal dimension
"b"	spectral dimension

NOTE - The *Identifier* entity is also used by other entities, such as *BandIdentifier* and *ComponentIdentifier*.

Syntax

```
DimensionDescription ::= SEQUENCE
  {
  dimension-identifier [0] IMPLICIT Identifier,                -- No. 107
  lower-boundary       [1] IMPLICIT INTEGER (0..MAX)
                           DEFAULT 0,                          -- No. 702
  upper-boundary       [2] IMPLICIT INTEGER (0..MAX) OPTIONAL -- No. 702
  }

Identifier ::= IA5String                                       -- No. 722
```

Constraints

The number of *DimensionDescription* entities contained in the sequence shall equal the value of the *number-of-dimensions* component in the *Dimension* entity.

The value of the *upper-boundary* component shall not be smaller than the value of the *lower-boundary* component.

See also *Serialization* entity for additional constraints.

IIF syntax entity No. 108 *Serialization*

Semantics

The *Serialization* entity determines the sequential order of array elements. The first descriptor indicates the outer-most loop over array indices. The last descriptor indicates the inner-most loop.

EXAMPLE - Let z be the first descriptor in the IIF data stream with an index range of [1..10], followed by y with an index range of [1..1280] and x with [1..1024]. Then the associated array elements $p(z,y,x)$ have to be interpreted as:

$$p(1,1,1)\ ,\ p(1,1,2)\ ,\ \ldots\ ,\ p(1,1,1024)\ ,\ p(1,2,1)\ ,\ p(1,2,2)\ ,\ \ldots\ ,\ \ldots\ ,\ p(10,1280,1023)\ ,\ p(10,1280,1024)$$

Syntax

```
Serialization ::= SEQUENCE OF Identifier                    -- No. 107
```

Constraints

The number of *Identifier* entities contained in the sequence shall equal the dimensionality of the array to which the serialization entity is attached. The identifier values shall be identical to those declared within the *dimensionality* component that is attached to the array mentioned above.

Only the first *Identifier* entity may have the value of a *dimension-identifier* component which is declared as being infinite within the *DimensionDescription* entity.

IIF syntax entity No. 109 *DataPlacement*

Semantics

The *DataPlacement* entity determines whether pixel values that belong to a certain image structure description are stored all together within one *ReferencedUnit* entity, or whether they are split up into multiple *ReferencedUnit* entities.

The structure that describes an image may be regarded as a tree whose root is the *ImageStructure* entity. All other entities that further specify the substructure (*CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, *CompoundImageSet*, *MetricArray*, and *BandRecord*) form nodes within this tree. The *ElementaryPixelStructure* entities form the leaves of this tree. At any node within this tree, it may be decided to store all pixel values that belong to this node within one *ReferencedUnit* entity.

Three alternatives are provided for the pixel values that belong to the "node" to which this *DataPlacement* entity is attached. The alternatives are associated with the following integer values:

distributed (1): The pixel values that belong to this node are split up into multiple *ReferencedUnit* entities, each of which is associated with one of the substructures of this node.

combined (2): The pixel values that belong to this node are stored all together within one *ReferencedUnit* entity, associated with this level of the image structure tree.

included (3): The pixel values that belong to this node are stored, together with pixel values belonging to other nodes on the image structure tree, as part of a *ReferencedUnit* associated with a higher level node.

EXAMPLE - Given an image structure that consists of a tiled image, expressed as a two-dimensional *MetricArray* of two-dimensional *MetricArrays* of integer pixels.

Then, a "1" on the level of the upper array and a "2" on the level of the lower array indicate that there exist *ReferencedUnit* entities for every tile, each of which contains all pixel values that belong to this tile.

On the other hand, a "2" on the upper level array and (necessarily) a "3" on the lower level array indicate that all pixel values of the whole image are stored within a single *ReferencedUnit* entity. In this case, the *ReferencedUnit* entity consists of a data stream which first contains all pixel values of the first tile, followed by the pixel values of the second tile, etc. There are no delimiters in the data stream that indicate the tile boundaries. Since the upper array is two-dimensional, a tile represents an object in the two-dimensional space. Thus, a serialization needs to be given in order to unambiguously determine, which tile is stored at what position in the data sequence. This is given by the *serialization* component of the upper array description.

NOTE - Depending on the values for the *DataPlacement* entity, references need to be included in the *ReferencedUnit* entities to indicate which *ReferencedUnit* belongs to which node in the image structure tree. For the description of the *ReferencedUnit* entity, refer to 5.3.3.

Syntax

```
DataPlacement ::= INTEGER                                    -- No. 702
                  {
                  distributed(1),
                  combined(2),
                  included(3)
                  }
```

Constraints

For the *DataPlacement* entity, only values in the range of (1..3) and values which have been internationally registered are permitted. Refer to 6.2.

On every path from the root of the image structure tree to a leaf there shall be one and only one node that is marked with a "2." All nodes located above shall be marked with "1" and all nodes located below shall be marked with "3."

IIF syntax entity No. 110 *CompoundImageRecord*

and

IIF syntax entity No. 111 *RecordComponent*

Semantics

The *CompoundImageRecord* entity stands for the description of a compound data type, as defined in ISO/IEC 12087-1. The number of record components is given by the *number-of-components* component. Then the placement of the data that are associated with the record is given by the *data-placement* component. The structure of each record component is given by the *record-components* component.

The *RecordComponent* entity consists of an identifier, called the *component-identifier*, and the structure, called the *component-structure*.

NOTE - The *CompoundImageRecord* entity contains the *ImageStructure* entity. The *ImageStructure* entity, in turn, contains the *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, and *CompoundImageSet* entities. With this syntactical recursive construction, image data structures of arbitrary complexity may be constructed.

Syntax

```
CompoundImageRecord ::= SEQUENCE
    {
    number-of-components   [0] IMPLICIT INTEGER (1..MAX),        -- No. 702
    data-placement         [1] IMPLICIT DataPlacement,           -- No. 109
    record-components      [2] IMPLICIT SEQUENCE OF
                               RecordComponent                   -- No. 111
    }


RecordComponent ::= SEQUENCE
    {
    component-identifier    [0] IMPLICIT Identifier,             -- No. 107
    component-structure     [1] IMPLICIT ImageStructure          -- No. 102
    }
```

Constraints

The number of *RecordComponent* entities contained in the following sequence shall equal the value of the *number-of-components* entity.

The character strings of the *component-identifier* entities shall be different from each other in order to allow for unambiguous referral to record components.

The character strings of the *component-identifier* entities shall not contain the characters "/" and "." because these characters have a special meaning for the *reference-label* components within the *ReferencedUnit* entities.

IIF syntax entity No. 112 — *CompoundImageList*

Semantics

The *CompoundImageList* entity stands for the description of a compound data type, as defined in ISO/IEC 12087-1. The number of list elements is given by the *number-of-elements* component. This component is optional in order to support lists whose length is indefinite at the beginning of the encoding process.

The placement of the data that are associated with the list is given by the *data-placement* component. The structure of the list elements is given by the *element-structure* entity.

NOTES

1 The *element-structure* component only contains a description of the data structure of the list elements but not the elements themselves. Since lists are homogeneous, one *ImageStructure* entity is sufficient to describe the entire list. The list elements are stored in *ReferencedUnit* entities according to the data placement described by the *data-placement* component.

2 The *CompoundImageList* entity contains the *ImageStructure*. The *ImageStructure* entity, in turn, contains the *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, and *CompoundImageSet* entities. With this syntactical, recursive construction, image structures of arbitrary complexity may be constructed.

Syntax

```
CompoundImageList ::= SEQUENCE
  {
  number-of-elements    [0] IMPLICIT INTEGER (1..MAX) OPTIONAL,-- No. 702
  data-placement        [1] IMPLICIT DataPlacement,            -- No. 109
  element-structure     [2] IMPLICIT ImageStructure            -- No. 102
  }
```

Constraints

None.

IIF syntax entity No. 113 *CompoundImageSet*

Semantics

The *CompoundImageSet* entity stands for the description of a compound data type, as defined in ISO/IEC 12087-1. The number of set elements is given by the *number-of-members* component. This component is optional in order to support sets whose cardinality is indefinite at the beginning of the encoding process.

The placement of the data that are associated with the set is given by the *data-placement* component. The structure of the set members is given by the *member-structure* entity.

NOTES

1 The *member-structure* component only contains a description of the data structure of the set members but not the members themselves. Since sets are homogeneous, one *ImageStructure* entity is sufficient to describe the entire set. The set members are stored in *ReferencedUnit* entities according to the data placement, described by the *data-placement* component.

2 The *CompoundImageSet* entity contains the *ImageStructure*. The *ImageStructure* entity, in turn, contains the *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, and *CompoundImageSet* entities. With this syntactical, recursive construction, image structures of arbitrary complexity may be constructed.

Syntax

```
CompoundImageSet ::= SEQUENCE
  {
  number-of-members    [0] IMPLICIT INTEGER (1..MAX) OPTIONAL, -- No. 702
  data-placement       [1] IMPLICIT DataPlacement,             -- No. 109
  member-structure     [2] IMPLICIT ImageStructure             -- No. 102
  }
```

Constraints

None.

IIF syntax entity No. 114 *FundamentalImageStructure*

Semantics

The *FundamentalImageStructure* entity stands for the structural description of fundamental image data type, as defined in ISO/IEC 12087-1. It can be an n-dimensional array of pixel values (given by the *metric-array* component) or a record of bands (given by the *band-record* component) each of which is defined as an n-dimensional array of pixel values.

All bands of a *FundamentalImageStructure* entity are assumed to refer to the same coordinate space. Hence, the *FundamentalImageStructure* entity may comprise a collection of pixel arrays that represent slices of a multi-spectral image or a simple colour image but it may not comprise a collection of pixel arrays that have been acquired at different locations or show different objects. The latter kind of compound image structure needs to be modelled with the *CompoundImageStructure* entity.

NOTES

1 Besides other image types, the *FundamentalImageStructure* entity covers elementary image data types, as defined in ISO/IEC 12087-1. For the definition of the relation between compound, fundamental, and elementary image types, refer to ISO/IEC 12087-1.

2 Because the *FundamentalImageStructure* entity allows multi-band images to be described, these images need not be modelled as compound images.

Syntax

```
FundamentalImageStructure ::= CHOICE
    {
    band-record     [0] BandRecord,                    -- No. 115
    metric-array    [1] MetricArray                    -- No. 117
    }
```

Constraints

None.

IIF syntax entity No. 115 *BandRecord*

Semantics

The *BandRecord* entity stands for the description of a record of image bands, each of which is identified by its component identifier. The number of bands is given by the *number-of-bands* component. The placement of the pixel values that belong to this structural description is determined by the *data-placement* component. The structure of each band is given by the *record-components* component.

Because the pixel values belonging to an image structure definition are ordered according to the sequence in which the structural definition of the image is given, the level at which an image is split into bands has an influence on the sequential order of pixel values.

EXAMPLE - An image that is described as a *metric-array* of *band-records* of three pixels forms a pixel-interleaved image whereas an image that is described as a *band-record* of three *metric-arrays* of pixels forms a band-interleaved image.

NOTE - The *BandRecord* entity is organized in conformance with the *CompoundImageRecord* entity. The term "band" as part of the name of the *BandRecord* entity reinforces the fact that the record components always have the meaning of bands of a multi-band image.

Syntax

```
BandRecord ::= SEQUENCE
  {
  number-of-bands     [0] IMPLICIT INTEGER (1..MAX),          -- No. 702
  data-placement      [1] IMPLICIT DataPlacement,             -- No. 109
  record-components   [2] IMPLICIT SEQUENCE OF
                              BandRecordComponent             -- No. 116
  }
```

Constraints

The number of *BandRecordComponent* entities contained in the *record-components* component shall equal the value of the *number-of-bands* entity.

IIF syntax entity No. 116 *BandRecordComponent*

Semantics

The *BandRecordComponent* entity stands for the description of an image band which is part of a multi-band image, described by the *BandRecord* entity. The *BandRecordComponent* consists of the *band-identifier* component which is used to associate attributes describing the nature of the band with this band. Furthermore, it consists of the *component-structure* component which gives the structure of the band.

Syntax

```
BandRecordComponent ::= SEQUENCE
   {
   band-identifier          [0] IMPLICIT Identifier,          -- No. 107
   component-structure      [1] IMPLICIT MetricArray          -- No. 117
   }
```

Constraints

The character strings of the *band-identifier* entities shall be unique in order to allow unambiguous referral to record components.

The character strings of the *band-identifier* entities shall not contain the characters "/" and "." because these characters have a special meaning for the *reference-label* components within the *ReferencedUnit* entities.

IIF syntax entity No. 117 *MetricArray*

Semantics

The *MetricArray* entity stands for the description of an n-dimensional array. The dimensionality, (including the dimension identifiers and index ranges for every dimension) is given by the *dimensionality* component. The sequential order of the array elements is determined within the *serialization* component. The placing of the data values that are associated with this array description is given within the *data-placement* component. The structure of the array elements is given by the *element-structure* component.

NOTES

1 The structure of the *MetricArray* entity is similar to the structure of the *CompoundImageArray*. The term "metric" as part of the name of the *MetricArray* entity reinforces the fact that the dimensions of this array are assumed to refer to a coordinate space.

2 The term "dimensionality of the *MetricArray* entity" is used in subsequent entities. This always refers to the value of the *number-of-dimensions* component within the *Dimensionality* entity.

Syntax

```
MetricArray ::= SEQUENCE
    {
    dimensionality          [0] IMPLICIT Dimensionality,      -- No. 105
    serialization           [1] IMPLICIT Serialization,       -- No. 108
    data-placement          [2] IMPLICIT DataPlacement,       -- No. 109
    element-structure       [3] MetricArrayElement            -- No. 118
    }
```

Constraints

None.

IIF syntax entity No. 118 *MetricArrayElement*

Semantics

The *MetricArrayElement* entity stands for the description of the element structure of *MetricArray* entities.

Three alternatives are provided. In the *pixel-structure* case, the array elements are pixels, in the *band-record* case, the array elements have multiple bands, in the *metric-array* case, the array elements are arrays themselves. The latter method can be used to describe tiled image structures.

EXAMPLE - Given a pixel array of 2048 by 3072 pixels that shall be organized as a 2D array of tiles that have a size of 256 by 256 pixels. Therefore, the *FundamentalImageStructure* entity needs to be organized as a 2D metric array with the index ranges (1..8) and (1..12) for the x and y dimensions, respectively. Then the metric array's element structure needs to be declared as a 2D array with the index range (1..64) for both x and y dimensions.

NOTES

1 Due to the fact that arrays are homogeneous collections of elements, all tiles of a tiled image which are modelled as arrays of arrays have the same size and structure.

2 For a rationale behind the various ways of organizing *FundamentalImageStructure* entities into metric arrays and band records, refer also to the description of the *BandRecord* and *FundamentalImageStructure* entities.

Syntax

```
MetricArrayElement ::= CHOICE
    {
    pixel-structure          [0] PixelStructure,          -- No. 119
    band-record              [1] BandRecord,              -- No. 115
    metric-array             [2] MetricArray              -- No. 117
    }
```

Constraints

The values of all *dimension-identifier* components that are declared within the subentities of a *MetricArrayElement* entity need to match the values of the *dimension-identifiers* which are declared within the upper entities of a *FundamentalImageStructure* entity.

IIF syntax entity No. 119 ***PixelStructure***

and

IIF syntax entity No. 120 ***ElementaryPixelStructure***

Semantics

The *PixelStructure* entity stands for the description of the structure of a pixel. Two alternatives are provided: in the *elementary-pixel* case, a pixel of elementary data type can be described, in the *compound-pixel* case, a pixel that consists of multiple components can be described.

The *ElementaryPixelStructure* entity stands for the description of an elementary data type that characterizes the structure of a pixel. According to the elementary data types which are defined in ISO/IEC 12087-1, the following alternatives, associated with integer values, are provided: *boolean, non-negative integer, signed integer, real, complex,* and *enumerated.*

NOTE - No pixel values but only type information is expressed by the *PixelStructure* entity and its subentities.

Syntax

```
PixelStructure ::= CHOICE
   {
   elementary-pixel-structure  [0] ElementaryPixelStructure,   -- No. 120
   compound-pixel-structure    [1] PixelBandRecord             -- No. 121
   }


ElementaryPixelStructure ::= INTEGER                           -- No. 702
                             {
                             boolean(1),
                             non-negative-integer(2),
                             signed-integer(3),
                             real(4),
                             complex(5),
                             enumerated(6)
                             }
```

Constraints

For the *ElementaryPixelStructure* entity only values in the range of (1..6) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 121 ***PixelBandRecord***

and

IIF syntax entity No. 122 ***PixelBandRecordComponent***

Semantics

The *PixelBandRecord* entity stands for the description of a pixel that consists of a record of bands, each of which is identified by its component identifier. The number of bands is given by the *number-of-bands* component. The structure of each band is given by the *record-components* component.

NOTE - The *PixelBandRecord* entity equals the *BandRecord* entity, except for the *data-placement* component. This component is missing on the pixel structure level due to the fact that it makes no sense to split up the subcomponents of pixels into multiple *ReferencedUnits*.

The *PixelBandRecordComponent* entity stands for the description of a component of a compound pixel, described by the *PixelBandRecord* entity. The *PixelBandRecordComponent* consists of the *band-identifier* component which may be used to refer to an attribute that describes the nature of the band. Furthermore, it consists of the *band-structure* component which gives the structure of this band of a compound pixel.

Syntax

```
PixelBandRecord ::= SEQUENCE
  {
  number-of-components  [0] IMPLICIT INTEGER (1..MAX),        -- No. 702
  record-components     [1] IMPLICIT SEQUENCE OF
                            PixelBandRecordComponent          -- No. 122
  }


PixelBandRecordComponent ::= SEQUENCE
  {
  band-identifier       [0] IMPLICIT Identifier,              -- No. 107
  component-structure   [1] IMPLICIT ElementaryPixelStructure -- No. 120
  }
```

Constraints

The number of *PixelBandRecordComponent* entities contained in the sequence shall equal the value of the *number-of-components* entity.

The character strings of the *band-identifier* entities shall be different from each other in order to allow unambiguous referral to record components.

The character strings of the *band-identifier* entities shall not contain the characters "/" and "." because these characters have a special meaning for the *reference-label* components within the *ReferencedUnit* entities.

5.3.3 Entities for the description of the representation of pixel values

IIF syntax entity No. 201 ***ReferencedUnit***

Semantics

The *ReferencedUnit* entity stands for the description of a unit of pixel values that is marked with a label to indicate to which image structure it belongs. The label is given by the *reference-label* component. The pixel values are contained in the *pixel-values* component.

As stated in the description of the *DataPlacement* entity, the structure that describes an image may be regarded as a tree whose root is the *ImageStructure* entity. All other entities that further specify the substructure (*CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, *CompoundImageSet*, *MetricArray*, and *BandRecord*) form nodes within this tree. The *ElementaryPixelStructure* entities form the leaves of this tree.

For the referral mechanism applied within the *reference-label* components, the following rules are specified:

a) Any node in the tree can be identified by the names of all edges that lead from the root to this node. The names of the edges are determined by the image structure description. They vary depending on the type of the nodes:

   a1) For records (given by the *CompoundImageRecord*, *BandRecord*, and *PixelBandRecord* entities), the names of the edges that lead from the record description to one of its components are defined as the names of the record components, given by the *component-identifier* and *band-identifier* components.

   a2) For arrays (given by the *CompoundImageArray* and *MetricArray* entities), the names of the edges that lead from the array description to one of its elements are defined as the multidimensional indices. The index values for every dimension are ordered according to the *serialization* component that is part of the array description. They are separated with "." characters.

   a3) For lists (given by the *CompoundImageList* entity), the names of the edges that lead from the list description to one of its list elements are defined by the sequential position of set members in the data stream, assuming that the first element in the sequence is designated to be number "1" and the successor of the *n*th element is assigned to be the *n+1*th element.

   a4) For sets (given by the *CompoundImageSet* entity), the names of the edges that lead from the set description to one of its set members are defined by the sequential position of set members in the data stream, assuming that the first element in the sequence is designated to be number "1" and the successor of the *n*th element is assigned to be the *n+1*th element.

b) In order to create the value of a *reference-label* component, the names of all edges are concatenated, inserting a "/" character in front of every edge name, beginning with a "/" character for the root.

EXAMPLE - Given an image structure that consists of a two-dimensional array of tiles, each of which consists of a record of 3 bands, called "red," "green," and "blue." Then the green band of the second tile in the first row of tiles is identified with the reference label "/1.2/green". The entire tile is identified with "/1.2" and the whole image is identified with "/".

Syntax

```
ReferencedUnit ::= SEQUENCE
   {
   reference-label   [0] IMPLICIT Identifier,                    -- No. 107
   pixel-values      [1] DataUnit                                -- No. 202
   }
```

Constraints

For every node in an image structure tree that has a *data-placement* component which is marked with a "2" (see *DataPlacement* entity), there shall exist one and only one *ReferencedUnit* entity that contains all pixel values that belong to this node and that contains a *reference-label* component that describes the path of this node.

The representation of the pixel values and the number of values that are contained in the *DataUnit* entity shall match the image structure description this *ReferencedUnit* refers to.

IIF syntax entity No. 202 *DataUnit*

Semantics

The *DataUnit* entity can either be a single data field without further subdivision, given by the *single-data-unit* component; a data field that is partitioned into a number of equal-sized portions, given by the *subdivided-data-unit* component; or a reference to a *ReferencedUnit* entity which is stored remotely, given by the *external-data-unit* component.

For the description of the partitioning concept refer to the *SubdividedDataUnit* entity. For the description of the referral scheme to external data sets refer to the *ExternalReference* entity.

Syntax

```
DataUnit ::= CHOICE
   {
   single-data-unit       [0] SingleDataUnit,                -- No. 204
   subdivided-data-unit   [1] SubdividedDataUnit,            -- No. 203
   external-data-unit     [2] ExternalReference              -- No. 211
   }
```

Constraints

See subentities.

IIF syntax entity No. 203 *SubdividedDataUnit*

Semantics

The *SubdividedDataUnit* entity describes a data field that is partitioned into a number of equal-sized portions, with the exception of the last portion which may be smaller in the case of a division with remainder (no filling up with dummy values takes place).

It consists of the *number-of-pixels* component, an integer number that gives the number of elementary pixel values that are contained in a partition. The *partitions* component contains the sequence of all partitions.

The *SubdividedDataUnit* entity provides a mechanism to assist access tools in handling large data sets. This is comparable to the strip concept of some common file formats.

NOTES

1 This concept is based on the counting of pixels but not on the counting of bytes. The physical size of a partition depends on the pixel type and on the encoded representation which is used to store the data.

2 Note also that this concept does not provide a geometrical partitioning within a multidimensional space, as provided, for instance, by tiling mechanisms. Due to the fact that the subdivision takes place at the level of a sequential data stream, the set of pixel values that belong to a partition does not necessarily represent a rectangular region of the n-dimensional pixel array. This has to be taken into account for the representation that is being used for the pixel values.

EXAMPLE - Given a colour image, structured as a 2D array with 256 by 256 pixels. The pixels consist of three elementary integers, one for each colour band. The pixel values for this (pixel-interleaved) image structure shall be partitioned into portions that represent for instance 8 lines of 256 RGB pixels. Then the *number-of-pixels* component needs to be set to 8 * 256 * 3 = 6144, because it counts elementary pixel values. The byte-size of each partition depends on its encoded representation. Different partitions may vary in size due to image compression techniques.

Syntax

```
SubdividedDataUnit ::= SEQUENCE
   {
   number-of-pixels  [0] IMPLICIT INTEGER (1..MAX),            -- No. 702
   partitions        [1] IMPLICIT SEQUENCE OF SingleDataUnit   -- No. 204
   }
```

Constraints

When using a compression scheme to encode the partitions, care has to be taken that the size of the partitions matches the size constraints defined for the selected compression scheme.

IIF syntax entity No. 204 *SingleDataUnit*

Semantics

The *SingleDataUnit* entity contains a pixel field that is associated with a data structure description via the *ReferencedUnit* entity. For the representation of the pixel field, one of the following alternatives shall be chosen:

a) Every pixel value is encoded as one of the elementary ASN.1 built-in types. This alternative is covered by the *builtin-encoded-data-unit* component.

b) All pixel values that belong to the pixel field are encoded as a single entity in terms of ASN.1. The field-internal encoding is then defined externally from the viewpoint of ASN.1. This alternative is covered by the *externally-defined-data-unit* component. The definition of the field-internal encoding is defined externally as described within the semantics description of the *ExternallyDefinedDataUnit* entity.

c) The pixel field is compressed with one of the compression methods available for the IIF-DF. This alternative is covered by the *compressed-data-unit* component.

d) A registered encoding scheme has been chosen that is not defined by this part of ISO/IEC 12087 but by an internationally registered profile. This alternative is covered by the *registered-data-unit* component.

NOTES

1 Alternative a) may produce considerable overhead in space and processing time. For example, the representation of a field of 8-bit integer pixels as *SEQUENCE OF INTEGER* causes a tripling of space overhead because every integer value is preceded by an 8-bit tag field and an 8-bit length field.

2 Alternative b) is only available for a limited set of pixel types (see the *ExternallyDefinedDataUnit* entity and refer to ISO/IEC 12089).

Syntax

```
SingleDataUnit ::= CHOICE
  {
  builtin-encoded-data-unit      [0] BuiltinEncodedDataUnit,    -- No. 205
  externally-defined-data-unit   [1] ExternallyDefinedDataUnit,-- No. 208
  compressed-data-unit           [2] CompressedDataUnit,        -- No. 209
  registered-data-unit           [3] RegisteredDataUnit         -- No. 210
  }
```

Constraints

The alternative chosen to represent a *SingleDataUnit* entity shall match the type definition with which the *SingleDataUnit* entity is associated. For an exact definition, refer to the three *BuiltinEncodedDataUnit*, *ExternallyDefinedDataUnit*, and *CompressedDataUnit* subentities.

IIF syntax entity No. 205 *BuiltinEncodedDataUnit*

Semantics

The *BuiltinEncodedDataUnit* entity represents an entire contents field. In general, it is encoded as a sequence of elementary ASN.1 built-in types, using the *BuiltinValue* entity. For a bitfield, a more efficient alternative is provided, in which the whole field is encoded as a single elementary ASN.1 entity.

Syntax

```
BuiltinEncodedDataUnit ::= CHOICE
    {
    sequence-of-boolean     [0] BIT STRING,                          -- No. 703
    sequence-of-others      [1] SEQUENCE OF BuiltinValue             -- No. 206
    }
```

Constraints

The number of *BuiltinValue* entities contained in the *BuiltinEncodedDataUnit* entity shall match the number of array elements defined by the *DimensionDescription* entities of the corresponding image structure.

See also *BuiltinValue* entity.

IIF syntax entity No. 206 ***BuiltinValue***

and

IIF syntax entity No. 207 ***ComplexValue***

Semantics

The *BuiltinValue* entity stands for the description of an elementary value (in most cases pixel values) that is represented by using the elementary ASN.1 built-in *BOOLEAN*, *INTEGER*, *REAL*, and *IA5String* types.

Complex values are encoded by using two *REAL* values, first the real part and then the imaginary part. This is given by the *ComplexValue* entity.

Syntax

```
BuiltinValue ::= CHOICE
   {
   boolean                    [0] IMPLICIT BOOLEAN,              -- No. 701
   non-negative-integer       [1] IMPLICIT INTEGER (0..MAX),     -- No. 702
   signed-integer             [2] IMPLICIT INTEGER,              -- No. 702
   real                       [3] IMPLICIT REAL,                 -- No. 709
   complex                    [4] IMPLICIT ComplexValue,         -- No. 207
   enumerated                 [5] IMPLICIT IA5String             -- No. 722
   }


ComplexValue ::= SEQUENCE
   {
   real        [0] IMPLICIT REAL,                                -- No. 709
   imaginary   [1] IMPLICIT REAL                                 -- No. 709
   }
```

Constraints

The representation that has been chosen for the *BuiltinValue* entity shall match the pixel representation specified in the image structure description. Table 1 shows which *BuiltinValue* entities may be used to represent a certain pixel data type.

ElementaryPixelStructure	BuiltinValue component
boolean	*boolean* (or *BIT STRING* for the whole field)
non-negative-integer	*non-negative-integer*
signed-integer	*signed-integer*
real	*real*
complex	*complex*
enumerated	*enumerated*

Table 1 - Representation of pixel data types.

IIF syntax entity No. 208 *ExternallyDefinedDataUnit*

Semantics

The *ExternallyDefinedDataUnit* entity stands for the description of a field of pixel values that is encoded according to an external encoding scheme. The scheme is described in ISO/IEC 12089.

NOTE - Neither a change nor an extension to the Basic Encoding Rules for ASN.1 (ISO/IEC 8825) to encode the *ExternallyDefinedDataUnit* entity is provided within ISO/IEC 12089. Instead, the basic principle is to express an entire field of pixel values as a single OCTET STRING. Thus the substructure of this field (i.e., the pixel values) is invisible at the ASN.1 level. ISO/IEC 12089 provides aligned and packed pixel representations.

Syntax

```
ExternallyDefinedDataUnit ::= PixelFieldEncoding                -- No. 801
```

Constraints

The representation which has been chosen for an *ExternallyDefinedDataUnit* entity shall match the pixel representation defined in the image structure definition. For this constraint, refer to ISO/IEC 12089.

The number of field elements contained in one *ExternallyDefinedDataUnit* entity shall conform to the image structure description to which the *ReferencedUnit* entity refers to, using the *reference-label* component.

Because complex values are to be encoded by using the float or double field (interpreting the field as consisting of pairs of float/double values), the number of field elements is twice as great as the number of declared pixels.

IIF syntax entity No. 209 *CompressedDataUnit*

Semantics

The *CompressedDataUnit* entity specifies a compressed data field. The field of pixel values is represented by the *data* component. The description of the field elements is given by the *data-representation* component. It implies a certain decompression method that needs to be applied to the data field during an interpretation of the IIF-DF stream in order to extract the pixel data. This decompression method and the encoded representation of the field elements are defined in the referenced documents. The named integer values, defined for the *data-representation* component, have to be interpreted as follows:

- The values *fax-t4(1)* and *fax-t6(2)* indicate that the field of pixel values is represented according to the Facsimile Standards, defined in CCITT Recommendations T.4 and T.6, respectively.
- The value *jbig(3)* indicates that the field of pixel values is represented according to the compressed data field specification, defined in ISO/IEC 11544 (JBIG).
- The value *jpeg(4)* indicates that the field of pixel values is represented according to the compressed data field specification defined in ISO/IEC 10918 (JPEG). The JPEG data stream may contain multiple bands, called C_i. In this case the i-th band within the JPEG data stream is associated with the i-th component of the respective *BandRecord* entity.
- The value *mpeg1(5)* indicates that the field of pixel values is represented according to the compressed audio/video data field specification defined in ISO/IEC 11172 (MPEG-1).

Additional data representations are subject to registration, as defined in 6.2.

NOTES

1 It is possible to compress parts of a compound (e.g. a multi-band) image, while leaving other parts (e.g., bands) uncompressed. It is even possible to compress individual partitions of a *DataUnit* entity, while leaving others uncompressed. Multiple (different) compression methods can also be used within a single IIF-DF.

2 Because a compression scheme may have several degrees of freedom, all parameters that have been set to compress a data stream have to be transported together with the compressed data to allow unambiguous decompression. In the case of the schemes named above, these data are contained within the syntax of the compressed data stream, (e.g., the interchange format in the case of the JPEG Standard). Thus, there is no need to describe these entities explicitly on the level of the IIF syntax. However, as described in clause 7, the functions of the IIF Gateway that deal with data compression need to handle those parameters that specify the compression method.

EXAMPLES

1 A time series image can be compressed with JPEG by applying the compression method to each time slice separately.

2 A monochrome image which is decomposed into image tiles can be compressed using the JPEG lossless mode for some tiles and the lossy mode for the other tiles.

3 A greyscale image which is decomposed into one bi-level image for each bit of intensity accuracy can be compressed using JBIG for each individual bi-level image plane.

Syntax

```
CompressedDataUnit ::= SEQUENCE
    {
    data-representation   [0] IMPLICIT INTEGER                    -- No. 702
                                        {
                                        fax-t4(1),
                                        fax-t6(2),
                                        jbig(3),
                                        jpeg(4),
                                        mpeg-1(5)
                                        },
    data                  [1] OCTET STRING                        -- No. 704
    }
```

Constraints

For the *data-representation* entity, only values in the range of (1..5) and values which have been internationally registered are permitted. Refer to 6.2.

NOTE - The compressed data fields which are provided by the *CompressedDataUnit* entity are used to represent fields of pixel values. The pixel values belong to an image whose structure is described by an *ImageStructure* entity and its subentities. Every standard mentioned above has its own limitation concerning the image structure that may be represented by means of the standard. For example, the JPEG Standard describes an encoded representation of multiband images but does not describe any representation of time-variant images. Thus, an image structure that describes a time-variant image does not fit together with a single JPEG-encoded field of pixel values. Note that this does not mean, time-variant images cannot be encoded using JPEG fields in any case. In fact, it is possible to represent a time-variant image as 1D (time) array of 2D colour images, where every slice is encoded as a separate JPEG field. The following list comprises constraints that can be stated for the compressed data fields concerning image structure limitations.

a) The (one- or two-dimensional) CCITT T.4 encodings are limited to image structures that consist of two-dimensional arrays. The pixel data type shall be Boolean. The dimension that corresponds to the inner loop over the indices is assumed to correspond to the horizontal axis. The index range for this dimension shall be 1728, 2048, or 2423.

b) The two-dimensional CCITT T.6 encoding is limited to image structures that consist of two-dimensional arrays. The pixel data type shall be Boolean. The dimension that corresponds to the inner loop over the indices is assumed to correspond to the horizontal axis.

c) The field of pixel values stored in a *CompressedDataUnit* entity represents a single facsimile page in terms of the Group 3 and Group 4 facsimile services.

d) For the conversion of Group 3 facsimile messages to the IIF-DF, the CCITT T.30 parameter "resolution" which may be set to *fine* or *coarse* should be mapped to the image attribute *DimensionDescription*, because it determines the pixel aspect ratio.

e) The JBIG encoding is limited to image structures that consist of two-dimensional arrays. The pixel data type shall be Boolean or non-negative integer (according to the generalization of the JBIG Standard which was previously limited to binary images).

f) The JPEG encoding is limited to the following image structure (also refered to as "JPEG source image data structure"): The image structure may either be an array of records of elementary pixels or a record of arrays of elementary pixels. The array dimensionality shall be 2 and the elementary pixel data type is defined to

have either 8 or 12 bits for the lossy compression modes or 2 to 16 bits for the lossless compression modes. Record components need to have identical pixel precisions. Note that the lossy mode of JPEG is designed for the compression of continuous tone images only.

g) The MPEG-1 encoding is limited to the following image structure: The array dimensionality shall be 3, where the dimension that corresponds to the outer loop over the indices is assumed to be the time axis. The pixel data type shall be a record of a two by two array of 8-bit luminance (Y) values and two elementary 8-bit chrominance (Cr and Cb) values. The luminance array shall have less or equal 768 * 576 pixels. The time axis shall be mapped to a metric measurement unit attribute which tells that the offset is less or equal to 1/30 seconds.

IIF syntax entity No. 210 *RegisteredDataUnit*

Semantics

The *RegisteredDataUnit* entity contains a field of pixel values which are encoded according to a registered encoding scheme.

The *registration-id* component allows for the identification of the encoding method. The registration procedure is described in 6.2.

The *pixel-values* component contains all pixel values. From the ASN.1 perspective, they can be represented by any syntax entity.

EXAMPLE - The *RegisteredDataUnit* could be used for the registration of advanced compression schemes which result from a combination or an enhancement of existing compression standards.

Syntax

```
RegisteredDataUnit ::= SEQUENCE
    {
    registration-id     [0] IMPLICIT INTEGER,                    -- No. 722
    pixel-values        [1] ANY DEFINED BY registration-id
    }
```

Constraints

For the *registration-id* component, only those values are permitted for which an international registration exists. Refer to 6.2.

The alternative chosen, to represent a *SingleDataUnit* entity, shall match the type definition with which the *SingleDataUnit* entity is associated.

IIF syntax entity No. 211 *ExternalReference*

and

IIF syntax entity No. 212 *ExternalAddress*

Semantics

The *ExternalReference* entity supports a referral mechanism to an externally stored data set within a heterogeneous and distributed computing environment.

NOTE - Within ISO/IEC 12087, this mechanism is exclusively used to refer to a field of pixel values that is not present within the current data stream.

The *object-address* entity is used to describe the address of the external object. Both local environments (e.g., file names in a UNIX file system) and heterogeneous, distributed environments (e.g., addresses of objects that may be obtained via ftp) are supported.

The *object-format* component allows for a description of the format of the externally stored data set. This is done by a unique identifier of the grammar. If this component is absent, the external data portion shall be interpreted as a *ReferencedUnit* entity, according to the IIF-DF syntax.

The *object-internal-id* component is an optional component which may be used to identify a specific object part within the object specified by the *object-address* component (e.g., by giving a byte offset that points from the top of the file to the beginning of the valid data). A concrete interpretation of this component may be specified within an application profile. Application profiles are subject to registration, as defined in 6.2.

The *ExternalAddress* entity provides two alternatives for the representation of an external object address. The *structured-address* component provides a unique location and name of the external object, according to ISO/IEC 10031, Distributed Office Application Model (DOAM), Part 2: Distinguished Object Reference (DOR). The *cleartext-address* component may be used for an informal representation. A concrete interpretation of this component may be specified within an application profile. Application profiles are subject to registration, as defined in 6.2.

Syntax

```
ExternalReference ::= SEQUENCE
  {
  object-address      [0] ExternalAddress,                          -- No. 212
  object-format       [1] IMPLICIT OBJECT IDENTIFIER OPTIONAL,-- No. 706
  object-internal-id  [2] IMPLICIT CharacterString OPTIONAL    -- No. 006
  }

ExternalAddress ::= CHOICE
  {
  structured-address    [0] DOR,                                    -- No. 802
  cleartext-address     [1] IMPLICIT CharacterString                -- No. 006
  }
```

Constraints

None.

5.3.4 Entities for the description of image-related data

IIF syntax entity No. 301 ***ImageRelatedData***

Semantics

The *ImageRelatedData* entity stands for the description of image-related data, as defined in ISO/IEC 12087-1. The following types of image-related data are provided: match point, look-up table, histogram, region of interest, neighbourhood, and feature list.

The *identifier* component is used to associate the image-related data with those parts of an image structure which have the same identifier.

The *usage* component allows for a verbal description of the way the image-related data type is used.

The types of image-related data named above are given by the *histogram*, *look-up-table*, *region-of-interest*, *neighbourhood-array*, *static array*, *feature-list*, *value-bounds-collection*, *matrix*, *pixel-record*, and *tuple* components.

Syntax

```
ImageRelatedData ::= SEQUENCE
  {
  identifier                 [0] IMPLICIT Identifier,          -- No. 107
  usage                      [1] ANY OPTIONAL,
  data-type CHOICE
    {
    histogram                [2] Histogram,                    -- No. 302
    look-up-table            [3] LookUpTable,                  -- No. 304
    region-of-interest       [4] RegionOfInterest,             -- No. 305
    neighbourhood-array      [5] NeighbourhoodArray,           -- No. 312
    static-array             [6] StaticArray,                  -- No. 314
    feature-list             [7] FeatureList,                  -- No. 315
    value-bounds-collection  [8] ValueBoundsCollection,        -- No. 317
    matrix                   [9] TransformationMatrix,         -- No. 318
    pixel-record            [10] PixelRecord,                  -- No. 319
    tuple                   [11] Tuple                         -- No. 321
    }
  }
```

Constraints

None.

IIF syntax entity No. 302 ***Histogram***

and

IIF syntax entity No. 303 ***PartitionClass***

Semantics

The *Histogram* entity stands for a histogram, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The number of partition classes is given by the *number-of-classes* component. A description of the partition classes' format is given by the entity *class-description*. In general, this is the data type definition of the pixels to be classified. Then, all partition classes and counts are given by the *classes-and-counts* component.

NOTE - This entity allows one to describe partition classes that are associated with images which have compound pixels (e.g., records of elementary data types).

The *PartitionClass* entity stands for the description of a partition class and the corresponding count.

The partition class is given by two sequences of elementary data values. One is called the *lower-boundary* and the other is called the *upper-boundary*. Both sequences together form an interval in an n-dimensional parameter space given by the pixel data type. All values *v* within the interval

lower-boundary <= *v* < *upper-boundary*

belong to the partition class except for the upper-most partition class where the value of the *upper-boundary* is included in the interval.

Syntax

```
Histogram ::= SEQUENCE
  {
  number-of-classes    [0] IMPLICIT INTEGER (1..MAX),           -- No. 702
  class-description    [1] BasicDataType,                       -- No. 602
  classes-and-counts   [2] IMPLICIT SEQUENCE OF PartitionClass -- No. 303
  }

PartitionClass ::= SEQUENCE
  {
  lower-boundary    [0] IMPLICIT SEQUENCE OF BuiltinValue,     -- No. 206
  upper-boundary    [1] IMPLICIT SEQUENCE OF BuiltinValue,     -- No. 206
  count             [2] IMPLICIT INTEGER (0..MAX)              -- No. 702
  }
```

Constraints

The value of the *number-of-classes* component shall equal the number of *PartitionClass* entities within the *classes-and-counts* component.

The *class-description* component shall match the pixel data type defined for the *FundamentalImageStructure* entity, with which the histogram is associated.

The number of *BuiltinValue* entities within the *lower-boundary* sequence shall equal the number of

elementary data types of which a compound (pixel) data type (defined by the *class-description* component) is constructed. The same condition holds for the number of *BuiltinValue* entities within the *upper-boundary* component.

The value of the *upper-boundary* component shall not be smaller than the value of the *lower-boundary* component.

EXAMPLE - Let an image structure consist of an array of pixels that are records of three integers. Then, a histogram associated with this pixel array (via the image-related data entity) has to have a class description identical to the pixel data type. All partition classes contain three integer values as the lower boundary and three integer values as the upper boundary.

IIF syntax entity No. 304 *LookUpTable*

Semantics

The *LookUpTable* entity stands for the description of a look-up table, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The *number-of-input-dimensions*, *input-data-types*, *number-of-output-dimensions*, and *output-data-types* components describe the look-up table's format.

The *number-of-input-dimensions* component describes the number of independent elementary input values which are used to index the table. The *input-data-types* component describes the data type of these input values as a whole. The *number-of-output-dimensions* component describes the number of independent elementary output values. The *output-data-types* component describes the data type of these output values as a whole. The *number-of-entries* component gives the total number of look-up table entries. All input and output values are given by a sequence of *BuiltinValue* entities, as defined in 5.3.3.

Both *input-data-types* and *output-data-types* are represented by the *BasicDataType* entity which may be either a compound data type or an elementary data type. Input and output vectors may be specified by records, represented by the *BasicRecord* entity. Thus, every component declaration encompasses a component identifier. These identifiers are used to bound colour space declarations to look-up tables.

The *input-data-types* component may contain real and complex as elementary data types. In these cases, an interpolation method needs to be declared. The *interpolation-method* component may be used for a textual description. If it is absent, a linear interpolation function is assumed for integer and real data types; for complex data types, the interpolation method is application-dependent.

NOTE - Because the input data type may also be a compound data type (e.g., a record), the look-up table provides a general mapping of input vectors to output vectors.

Both *input-values* and *output-values* components form sequences of elementary values. Depending on the input and output dimensionality, a table index and a table entry may consist of multiple elementary values. The *i*th index (represented by one or more elementary values) within the *input-values* component is semantically bound to the *i*th entry (represented by one or more elementary values) within the *output-values* component.

EXAMPLES

1 Let us assume, the output values of a look-up table are to be interpreted as YIQ colour values. This fact may be expressed by the IIF-DF syntax using the following identifier settings:

ImageStructure:	*band-identifier*s within *BandRecord*:	"table1-a" "table1-b" "table1-c"
LookUpTable:	*input-data-types: component-identifier*s within *BasicRecord*:	"table1-a" "table1-b" "table1-c"
	*output-data-types: component-identifier*s within *BasicRecord*:	"y" "i" "q"
YIQColourSpace:	*y-band-identifier*: *i-band-identifier*: *q-band-identifier*:	"y" "i" "q"

2 Let us assume, the input values are to be interpreted according to an RGB colour system ("r", "g", "b") where the "r", "g", and "b" bands take integer values in the range of (0..2), (0..4), and (0..1), respectively. The look-up table shown below maps the 24 possible RGB value triples onto YIQ colour values:

ImageStructure:	*band-identifier*s within *BandRecord*:	"r" "g" "b"
LookUpTable:	*number-of-input-dimensions*	3
	input-data-types	*BasicRecord* (3, ("r", non-negative integer), ("g", non-negative integer), ("b", non-negative integer))
	number-of-output-dimensions	3
	output-data-types	*BasicRecord* (3, ("Y", real), ("i", real), ("q", real))
	number-of-entries	24
	input-values	((0,0,0), (0,0,1), ..., (2,3,1))
	output-values	((.0,.0,.0), (.10,.20,.04), ..., (1.0,.0,.0))
YIQColourSpace:	*y-band-identifier*: *i-band-identifier*: *q-band-identifier*:	"y" "i" "q"

Syntax

```
LookUpTable ::= SEQUENCE
  {
  number-of-input-dimensions  [0] IMPLICIT INTEGER (1..MAX),   -- No. 702
  input-data-types            [1] BasicDataType,               -- No. 602
  number-of-output-dimensions [2] IMPLICIT INTEGER (1..MAX),   -- No. 702
  output-data-types           [3] BasicDataType,               -- No. 602
  number-of-entries           [4] IMPLICIT INTEGER (1..MAX),   -- No. 702
  interpolation-method        [5] IMPLICIT CharacterString
                                  OPTIONAL,                    -- No. 722
  input-values                [6] IMPLICIT SEQUENCE OF
                                  BuiltinValue,                -- No. 206
  output-values               [7] IMPLICIT SEQUENCE OF
                                  BuiltinValue                 -- No. 206
  }
```

Constraints

The value of the *number-of-input-dimensions* component shall equal the number of elementary data types which are declared by the *input-data-types* component.

The value of the *number-of-output-dimensions* component shall equal the number of elementary data types which are declared by the *output-data-types* component.

Let n be the value of the *number-of-entries* component, d_{in} the value of the *number-of-input-dimensions* component, and d_{out} the value of the *number-of-output-dimensions* component. Then the number of *BuiltinValue* entities contained in the *input-values* component shall equal $n * d_{in}$ and the number of *BuiltinValue* entities contained in the *output-values* component shall equal $n * d_{out}$.

The data types of the *BuiltinValue* entities contained in the *input-values* sequence shall match the data type specification in the *input-data-types* component.

The data types of the *BuiltinValue* entities, contained in the *output-values* component, shall match the data type specification in the *output-data-types* component.

IIF syntax entity No. 305 ***RegionOfInterest***

Semantics

The *RegionOfInterest* entity stands for the description of a region of interest (ROI), as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2. Five alternatives are provided for the representation of an ROI: a Boolean array, an ellipse, an n-dimensional interval, a polygon, and a set of coordinates.

a) The *array* component allows to represent the ROI by an array of Boolean values. This is the most general representation of an ROI because no constraints exist regarding its shape.

b) The *ellipse* component allows representation of the ROI by the geometrical shape of an ellipse.

c) The *rectangular* component allows representation of the ROI by an n-dimensional interval. This representation can be used to express rectangular regions.

d) The *polygon* component allows representation of the ROI by a sequence of coordinates that form a closed polygon. The region of interest is the inside of the polygon including its boundaries.

e) The *set-of-coordinates* component allows representation of the ROI by naming all coordinates that belong to the ROI. This representation is isomorphic to the Boolean array representation.

The *polarity-reversed* component may be used to define reversed polarity of the region of interest. If *polarity-reversed* is 'true', then the real ROI is the complementary set of the ROI actually described.

The *index-manipulation* component may be used in combination with the ROI types *ellipse*, *rectangular*, and *polygon* for the assignment of array dimensions as specified in ISO/IEC 12087-2.

NOTE - The *array* and *set-of-coordinates* components provide an explicit description of the ROI whereas the other components provide generic descriptions.

Syntax

```
RegionOfInterest ::= SEQUENCE
   {
   roi-type CHOICE
      {
      array              [0] BooleanArray,                    -- No. 306
      ellipse            [1] Ellipse,                         -- No. 307
      rectangular        [2] IntervalND,                      -- No. 308
      polygon            [3] SetOfCoordinates,                -- No. 311
      set-of-coordinates [4] SetOfCoordinates                 -- No. 311
      },
   polarity-reversed     [5] IMPLICIT BOOLEAN DEFAULT FALSE,  -- No. 701
   index-manipulation    [6] IMPLICIT SEQUENCE OF
                             Identifier OPTIONAL               -- No. 107
   }
```

Constraints

The *index-manipulation* component may be used only in combination with the ROI types *ellipse*, *rectangular*, and *polygon*. The number of *Identifier* entities contained in the *index-manipulation* component shall equal the dimensionality of the ROI.

IIF syntax entity No. 306 *BooleanArray*

Semantics

The *BooleanArray* entity stands for the description of a region-of-interest, represented as an array of Boolean values.

The dimensionality of the array (including the index ranges and identifiers for every dimension) is given by the *dimensionality* component.

The sequential order of Boolean values within the *boolean-values* component is defined within the *serialization* component.

The data of the bit array are given by the *boolean-values* component. The order of the *dimension-description* entities defines the order of the bits within the pixel data.

Syntax

```
BooleanArray ::= SEQUENCE
   {
   dimensionality    [0] IMPLICIT Dimensionality,                -- No. 105
   serialization     [1] IMPLICIT Serialization,                 -- No. 108
   boolean-values    [2] IMPLICIT BIT STRING                     -- No. 703
   }
```

Constraints

The number of Boolean values contained in the *boolean-values* component shall match the array's index range defined by the *dimensionality* component.

IIF syntax entity No. 307 *Ellipse*

Semantics

The *Ellipse* entity stands for the description of an elliptical region of interest object.

The *roi-dimensionality* component specifies the overall dimensionality of the region of interest. The *roi-size* component is used to identify an interval in the n-dimensional index space which establishes the extent of the ROI with respect to the origin of the ellipse.

The *ellipse-center* component gives the n-dimensional center point. The *ellipse-axis-length* component gives an n-dimensional vector of ellipse axes which are aligned by the coordinate system.

The *ellipse-dimensionality* component specifies the number of elliptical dimensions of the region of interest object. The *elliptical-dimensions* component is used to name the identifiers of the elliptical dimensions. In all non-elliptical dimensions, the region of interest object is limited by the roi-size component.

Syntax

```
Ellipse ::= SEQUENCE
  {
  roi-dimensionality      [0] IMPLICIT INTEGER,                  -- No. 702
  roi-size                [1] IMPLICIT IntervalND,               -- No. 308
  ellipse-center          [2] IMPLICIT CoordinateND,             -- No. 310
  ellipse-axis-length     [3] IMPLICIT CoordinateND,             -- No. 310
  ellipse-dimensionality [4] IMPLICIT INTEGER (2..MAX),          -- No. 702
  elliptical-dimensions   [5] IMPLICIT SEQUENCE OF Identifier    -- No. 107
  }
```

Constraints

The value of the roi-dimensionality shall match the dimensionality of the *roi-size*, *center*, and *axis-length* components.

The value of the *ellipse-dimensionality* component shall be less or equal to the value of the *roi-dimensionality* component.

The number of *Identifier* entities contained in the *elliptical-dimensions* component shall equal the value of the *ellipse-dimensionality* component.

IIF syntax entity No. 308 *IntervalND*

and

IIF syntax entity No. 309 *Interval1D*

Semantics

The *IntervalND* entity stands for the description of a region-of-interest, based on an n-dimensional interval. The intervals are defined upon the arrays' index ranges, but not upon geometric positions.

The dimensionality is given by the *number-of-dimensions* component, and the interval is defined by a sequence of one-dimensional intervals, given by the *intervals* component.

The *Interval1D* entity consists of two integer values. The first describes the lower boundary, the second describes the upper boundary.

NOTE - The term "dimensionality of the *IntervalND* entity" is used in subsequent entities. It always refers to the value of the *number-of-dimensions* component.

Syntax

```
IntervalND ::= SEQUENCE
   {
   number-of-dimensions  [0] IMPLICIT INTEGER (1..MAX),        -- No. 702
   intervals             [1] IMPLICIT SEQUENCE OF Interval1D   -- No. 309
   }


Interval1D ::= SEQUENCE
   {
   lower-boundary   [0] IMPLICIT INTEGER (0..MAX),             -- No. 702
   upper-boundary   [1] IMPLICIT INTEGER (0..MAX)              -- No. 702
   }
```

Constraints

For the *IntervalND* entity, the number of *Interval1D* entities contained in the *intervals* component shall equal the value of the *number-of-dimensions* component.

IIF syntax entity No. 310 *CoordinateND*

Semantics

The *CoordinateND* entity stands for the description of an n-dimensional index vector. The dimensionality is given by the *number-of-dimensions* entity. The index values may be of type real or integer.

NOTE - The term "dimensionality of the *CoordinateND* entity" is used in subsequent entities. It always refers to the value of the *number-of-dimensions* entity.

Syntax

```
CoordinateND ::= SEQUENCE
   {
   number-of-dimensions    [0] IMPLICIT INTEGER (1..MAX),       -- No. 702
   vector-type CHOICE
      {
      integer-vector       [1] IMPLICIT SEQUENCE OF INTEGER,    -- No. 702
      real-vector          [2] IMPLICIT SEQUENCE OF REAL        -- No. 709
      }
   }
```

Constraints

For the *CoordinateND* entity, the number of integer or real values contained in the sequence shall equal the value of the *number-of-dimensions* component.

IIF syntax entity No. 311 *SetOfCoordinates*

Semantics

The *SetOfCoordinates* entity stands for the description of a region-of-interest, represented as a set of pixel coordinates. The cardinality of the set is given by the *number-of-coordinates* component. The whole set of coordinates is given by the *pixel-coordinates* component.

Syntax

```
SetOfCoordinates ::= SEQUENCE
  {
  number-of-coordinates [0] IMPLICIT INTEGER (0..MAX),        -- No. 702
  pixel-coordinates     [1] IMPLICIT SEQUENCE OF CoordinateND -- No. 310
  }
```

Constraints

For a *set-of-coordinates-roi*, the number of *CoordinateND* entities contained in the *pixel-coordinates* sequence shall equal the value of the *number-of-pixel-coords* component.

The dimensionality of all *CoordinateND* entities shall equal the dimensionality of the image structure to which the region of interest is attached.

IIF syntax entity No. 312 *NeighbourhoodArray*

Semantics

The *NeighbourhoodArray* entity stands for the description of a basic neighbourhood, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The dimensionality of the array (including the index ranges and identifiers for every dimension) is given by the *dimensionality* component. The sequential order of values within the *array-elements* component is defined within the *serialization* component. The elementary data type of the array elements is specified by the *element-structure* component. The data of the neighbourhood array are given by the *array-elements* component.

The *semantic-label* component allows for the selection of one of the following predefined labels which is associated with integer values: *generic*, *dither*, *impulse-response*, *mask*, and *structuring-element*. Additional definitions are subject to registration as defined in 6.2.

The *key-pixel* component defines a reference pixel for the neighbourhood. The key pixel offset is measured with respect to the origin of the neighbourhood array.

The *scale-factor* component may be used for rescaling the neighbourhood entry values. It does not express a rescaling of the neighbourhood's size itself.

Syntax

```
NeighbourhoodArray ::= SEQUENCE
   {
   dimensionality      [0] IMPLICIT Dimensionality,              -- No. 105
   serialization       [1] IMPLICIT Serialization,               -- No. 108
   element-structure   [2] IMPLICIT ElementaryPixelStructure,    -- No. 120
   array-elements      [3] BuiltinEncodedDataUnit,               -- No. 205
   semantic-label      [4] IMPLICIT INTEGER                      -- No. 702
                                    {
                                    generic(1),
                                    dither(2),
                                    impulse-response(3),
                                    mask(4),
                                    structuring-element(5)
                                    },
   key-pixel           [5] IMPLICIT IndexND,                     -- No. 313
   scale-factor        [6] IMPLICIT REAL DEFAULT {1, 10, 0}      -- No. 709
   }
```

Constraints

The dimensionality of the *IndexND* entity shall equal the number of dimensions specified by the *dimensionality* component.

For the *semantic-label* component only values in the range of (1..5) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 313 *IndexND*

Semantics

The *IndexND* entity stands for the description of an n-dimensional index vector. The dimensionality is given by the *number-of-dimensions* entity. The index values are integers.

NOTE - The term "dimensionality of the *IndexND* entity" is used in subsequent entities. It always refers to the value of the *number-of-dimensions* entity.

Syntax

```
IndexND ::= SEQUENCE
  {
  number-of-dimensions    [0] IMPLICIT INTEGER (1..MAX),      -- No. 702
  index-vector            [1] IMPLICIT SEQUENCE OF INTEGER    -- No. 702
  }
```

Constraints

For the *IndexND* entity, the number of integer values contained in the sequence shall equal the value of the *number-of-dimensions* component.

IIF syntax entity No. 314 *StaticArray*

Semantics

The *StaticArray* entity stands for the description of a static array, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The dimensionality of the array (including the index ranges and identifiers for every dimension) is given by the *dimensionality* component. The sequential order of values within the *array-elements* component is defined within the *serialization* component. The data of the neighbourhood array are given by the *array-elements* component.

The *semantic-label* component allows for the selection of one of the following predefined labels: *generic*, *power-spectrum*, *transfer-function*, and *windowing-function*. Additional definitions are subject to registration as defined in 6.2.

Syntax

```
StaticArray ::= SEQUENCE
   {
   dimensionality      [0] IMPLICIT Dimensionality,              -- No. 105
   serialization       [1] IMPLICIT Serialization,               -- No. 108
   element-structure   [2] IMPLICIT ElementaryPixelStructure,   -- No. 120
   array-elements      [3] BuiltinEncodedDataUnit,               -- No. 205
   semantic-label      [4] IMPLICIT INTEGER                      -- No. 702
                                    {
                                    generic(1),
                                    power-spectrum(2),
                                    transfer-function(3),
                                    windowing-function(4)
                                    }
   }
```

Constraints

For the *semantic-label* component only values in the range of (1..4) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 315 ***FeatureList***

and

IIF syntax entity No. 316 ***CoordinateAndFeature***

Semantics

The *FeatureList* entity stands for the description of a feature list, as defined in ISO/IEC 12087-1.

The size of the list is given by the *number-of-coordinates* component, and the list of coordinates, together with the associated features, is given by the *coordinates-and-features* component.

The *CoordinateAndFeature* entity stands for the description of an n-tuple coordinate, given by the *coordinate* component, and of the associated feature, given by the *feature* component.

Syntax

```
FeatureList ::= SEQUENCE
    {
    number-of-coordinates      [0] IMPLICIT INTEGER (1..MAX),    -- No. 702
    coordinates-and-features   [1] IMPLICIT SEQUENCE OF
                                       CoordinateAndFeature -- No. 316
    }


CoordinateAndFeature ::= SEQUENCE
    {
    coordinate    [0] IMPLICIT coordinateND,                     -- No. 310
    feature       [1] ANY
    }
```

Constraints

The number of *CoordinateAndFeature* entities contained in the *coordinates-and-features* component shall equal the value of the *number-of-coordinates* component.

The dimensionality of all *CoordinateND* entities contained in the *CoordinateAndFeature* entity shall equal the dimensionality of the image structure to which the region of interest is attached.

IIF syntax entity No. 317 *ValueBoundsCollection*

Semantics

The *ValueBoundsCollection* entity stands for the description of a value bounds collection, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The *pixel-data-type* component is used to describe the pixel data type of the image to which the value bounds collection is attached.

The *lower-boundary* and *upper-boundary* components give the value bounds collection's value range. Each boundary may form either a single value or multiple values, depending on the above-mentioned pixel data type.

> EXAMPLE - For an image whose pixels consist of a *u* and a *v* band, a value bounds collection may be specified which contains the coordinates of all pixels whose values lie within the interval [$lower_u$ .. $upper_u$] for the *u* component and [$lower_v$ .. $upper_v$] for the *v* band. Therefore, the *lower-boundary* component shall be set to ($lower_u$, $lower_v$) and the *upper-boundary* component shall be set to ($upper_u$, $upper_v$)

The *number-of-coordinates* component gives the number of n-dimensional coordinates which are contained in the *coordinate-list* component.

Syntax

```
ValueBoundsCollection ::= SEQUENCE
 {
 pixel-data-type        [0] BasicDataType,                        -- No. 602
 lower-boundary         [1] IMPLICIT SEQUENCE OF BuiltinValue,    -- No. 206
 upper-boundary         [2] IMPLICIT SEQUENCE OF BuiltinValue,    -- No. 206
 number-of-coordinates  [3] IMPLICIT INTEGER (0..MAX),            -- No. 702
 coordinate-list        [4] IMPLICIT SEQUENCE OF IndexND          -- No. 313
 }
```

Constraints

The number of *IndexND* entities contained in the *coordinate-list* component shall equal the value of the *number-of-coordinates* component.

The dimensionality of all *IndexND* entities contained in the *coordinate-list* component shall equal the dimensionality of the image structure to which the value bounds collection is attached.

The structure of the *pixel-data-type* component shall equal the structure of the pixel data type of the image to which the value bounds collection is attached.

The number of elementary values contained in the *lower-boundary* and *upper-boundary* components shall equal the dimensionality of the pixel data type which is given by the *pixel-data-type* component.

IIF syntax entity No. 318 *TransformationMatrix*

Semantics

The *TransformationMatrix* entity stands for the description of a transformation matrix, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The square matrix is given in homogeneous coordinates as illustrated in the example below.

EXAMPLE - A transformation of a point (x, y) in a two-dimensional coordinate system may be specified by a 3 x 3 matrix as follows:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

whereby $w = 1$ and the transformed coordinates (x', y', w') are obtained as follows:

$$\begin{aligned} x' &= ax + by + cw \\ y' &= dx + ey + fw \\ w' &= gx + hy + iw. \end{aligned}$$

The output vector is normalized as follows:

$$\begin{aligned} x'' &= x'/w' \\ y'' &= y'/w'. \end{aligned}$$

The size of the matrix is given by the *size* component (see constraint below) and all matrix elements are given by the *matrix-elements* component. The elements are either integer or real values. The sequential organization of the matrix elements is such that the column index is the fastest-moving index.

Syntax

```
TransformationMatrix ::= SEQUENCE
   {
   matrix-size                [0] IMPLICIT INTEGER,                -- No. 702
   matrix-elements   CHOICE
      {
      integers                [1] IMPLICIT SEQUENCE OF INTEGER,    -- No. 702
      reals                   [2] IMPLICIT SEQUENCE OF REAL        -- No. 709
      }
   }
```

Constraints

If the *TransformationMatrix* is attached to an *n*-dimensional coordinate system, the value of the *matrix-size* component shall equal *n+1* and the number of real values contained in the *matrix-elements* component shall be $(n+1)^2$.

IIF syntax entity No. 319 ***PixelRecord***

and

IIF syntax entity No. 320 ***PixelRecordComponent***

Semantics

The *PixelRecord* entity stands for the description of a record of pixel values with each value being of one of the elementary data types, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2.

The number of pixel record components is given by the *number-of-components* component. The list of components is given by the *record-components* component.

The *PixelRecordComponent* entity stands for the description of a component of the pixel record. It consists of a component identifier, given by *identifier* component, and the component value, given by the *component-value* component. The latter allows for the representation of values any elementary data type.

Syntax

```
PixelRecord ::= SEQUENCE
    {
    number-of-components    [0] IMPLICIT INTEGER (1..MAX),      -- No. 702
    record-components       [1] IMPLICIT SEQUENCE OF
                                PixelRecordComponent            -- No. 320
    }

PixelRecordComponent ::= SEQUENCE
    {
    identifier              [0] IMPLICIT Identifier,            -- No. 107
    component-value         [1] BuiltinValue                    -- No. 206
    }
```

Constraints

The number of *PixelRecordComponent* entities contained in the sequence shall equal the value of the *number-of-components* entity.

The character strings of the *identifier* entities shall be different from each other in order to allow unambiguous referral to record components.

IIF syntax entity No. 321 *Tuple*

Semantics

The *Tuple* entity stands for the description of a collection of values of the same elementary data type, as defined in ISO/IEC 12087-1 and used in ISO/IEC 12087-2 for IPI-PIKS functional element parameters.

The number of tuple components is given by the *number-of-elements* component. The list of components is given by the *record-elements* component.

Syntax

```
Tuple ::= SEQUENCE
   {
   number-of-elements   [0] IMPLICIT INTEGER (1..MAX),          -- No. 702
   element-structure    [1] IMPLICIT ElementaryPixelStructure, -- No. 120
   values               [2] IMPLICIT SEQUENCE OF BuiltinValue  -- No. 206
   }
```

Constraints

The number of *BuiltinValue* entities contained in the sequence shall equal the value of the *number-of-components* entity.

The data types of the *BuiltinValue* entities contained in the *values* sequence shall match the data type specification in the *element-structure* component.

5.3.5 Entities for the description of attributes

IIF syntax entity No. 401 *ImageAttribute*

Semantics

The *ImageAttribute* entity stands for the description of any of the image-related attributes defined in ISO/IEC 12087-1. For the rationale behind these attribute classes, refer to ISO/IEC 12087-1.

Syntax

```
ImageAttribute ::= CHOICE
   {
   metric      [0] MetricDescription,                    -- No. 402
   channel     [1] ChannelCharacteristics,               -- No. 408
   colour      [2] ColourRepresentation,                 -- No. 410
   control     [3] PIKSControl,                          -- No. 434
   freeform    [4] ANY
   }
```

Constraints

None.

NOTE - Attributes may be attached not only to *FundamentalImageStructure* entities but also to higher-level structures. Thus, a single attribute may be declared for multiple images (e.g., by defining a record of images and attaching the attribute to the record). For this reason, the expression "... shall match all *FundamentalImageStructure* entities the attribute is declared for" is used within the list of constraints defined for the sub-entities of the *ImageAttribute* entity.

IIF syntax entity No. 402 *MetricDescription*

Semantics

The *MetricDescription* entity stands for the description of all metric attributes of an image, as defined in ISO/IEC 12087-1.

NOTE - The *MetricDescription* entity comprises information which belongs to the "representation attribute" class, defined in ISO/IEC 12087-1.

The *coordinate-system* component stands for the description of a reference coordinate system. The supported systems are Cartesian and angular. The orientation of the coordinate systems, with respect to an observer, is defined in ISO/IEC 12087-1. The following alternatives are predefined: *one-dimensional cartesian*, *two-dimensional cartesian*, *three-dimensional cartesian*, *four-dimensional cartesian*, *five-dimensional cartesian*, *other cartesian*, *two-dimensional polar*, *three-dimensional cylindrical*, *three-dimensional spherical*, and *other angular*. Additional definitions are subject to registration (see 6.2).

The *dimension-mappings* component allows expression of a relation between the indices of the dimensions and the coordinate system. The *metric-transform* component allows for the specification of a metric transformation of image data related to the coordinate system. If this component is absent, it is assumed that no transformation is applied to the image data. The *domain* component allows one to specify whether the image data are in the spatial or in the transform domain. If this component is absent, it is assumed that the image data are represented in the original space domain.

Syntax

```
MetricDescription ::= SEQUENCE
   {
   coordinate-system  [0] IMPLICIT INTEGER {                    -- No. 702
                                        cartesian-1d(1),
                                        cartesian-2d(2),
                                        cartesian-3d(3),
                                        cartesian-4d(4),
                                        cartesian-5d(5),
                                        cartesian-other(6),
                                        polar-2d(7),
                                        cylindrical-3d(8),
                                        spherical-3d(9),
                                        angular-other(10)
                                        },
   dimension-mappings    [1] IMPLICIT SEQUENCE OF
                                        DimensionMapping,   -- No. 403
   metric-transform      [2] MetricTransformation OPTIONAL,    -- No. 406
   domain                [3] IMPLICIT Domain OPTIONAL          -- No. 407
   }
```

Constraints

The coordinate system shall match the dimensionality of the *FundamentalImageStructure* entities for which it is declared. This matching depends on the meaning of the dimensions and coordinate axis defined within the *DimensionMapping* entity. Only values in the range of (1..10) and values which have been internationally registered are allowed for the *coordinate-system* component. Refer to 6.2.

IIF syntax entity No. 403 ***DimensionMapping***

Semantics

The *DimensionMapping* entity describes the relation between the coordinate system and the dimensions of the *FundamentalImageStructure* entities for which it is declared.

It consists of the *dimension-identifier* and *coordinate-axis* components in order to indicate which dimension is mapped onto which coordinate axis.

It then contains a description of a physical measurement unit that is optionally associated with the coordinate axis. This component is called *physical-measurement*. If it is missing, the *DimensionMapping* entity may be used to express aspect ratios among the various array dimensions without relating the axis to the physical world.

> EXAMPLE - Given an image that is structured as a three-dimensional array with the dimension identifiers "hor," "vert," and "time" and the index ranges for every dimension declared within the *dimensionality* component of the array. Then, this array can be associated with a coordinate system using three *DimensionMapping* entities, each of which refers to one of the identifiers "hor," "vert," and "time" and relates them to a coordinate axis, e.g., the first, second, and third axis, respectively.

The *origin* component defines the position of the center of the lower boundary index on the coordinate axis. The *single-delta* component gives the distance between two index values in terms of the coordinate axis. Using the *origin* and *single-delta* components, a linear transform can be expressed to give the relation between the indices of the dimension and the physical measurement of the coordinate system.

The *multiple-deltas* component allows for a description of irregular gridding. It provides a sequence of delta values.

The *precision* component specifies the number of decimal places that shall be regarded as significant. It may be used to describe how accurate the index grid is placed (e.g., sampled) along the coordinate axis.

Note that image structures may consist of multiple hierarchies of arrays (e.g., for tiled images). In this case, the *single-delta* component gives the distance between two index values of the lowest level array.

> EXAMPLE - Given a tiled image that is structured as a 2D array of 2D tiles. In order to calculate the size of the image, one first has to calculate the size of a tile (depending on the *delta* component and the index range of the tile). The result determines the element size for the upper level array. Hence, it needs to be multiplied with the number of tiles which is determined by the index range of the upper level array.

The pixel-aspect ratio can be expressed as an n-tuple of *delta* components, depending on the dimensionality of the array. If physical measurement units are provided, a probable difference among the measurement units also has to be taken into account. The "image aspect ratio" can be derived from the pixel-aspect ratio and the index range information provided by the *DimensionDescription* entity.

Syntax

```
DimensionMapping ::= SEQUENCE
  {
  dimension-identifier   [0] IMPLICIT Identifier,                -- No. 107
  coordinate-axis        [1] IMPLICIT INTEGER (1..MAX),          -- No. 702
  physical-measurement   [2] IMPLICIT MeasurementUnit OPTIONAL,-- No. 404
  origin                 [3] IMPLICIT REAL DEFAULT {0, 10, 0}, -- No. 709
  delta-type CHOICE
     {
     single-delta        [4] IMPLICIT REAL,                      -- No. 709
     multiple-deltas     [5] IMPLICIT DeltaVector                -- No. 405
     },
  precision              [6] IMPLICIT INTEGER OPTIONAL           -- No. 702
  }
```

Constraints

The value of the *dimension-identifier* component shall match one of the identifiers declared by the *DimensionDescription* components of the *FundamentalImageStructure* entities the attribute is declared for.

Refer also to the *DeltaVector* entity.

IIF syntax entity No. 404 *MeasurementUnit*

Semantics

The *MeasurementUnit* entity stands for the description of the physical measurement associated with a coordinate axis of the coordinate system. It consists of the *measurement*, *basis*, and *description* components.

The *measurement* component provides the following predefined measurements: *meter, second, degree, hertz,* and *application-specific*. Additional definitions are subject to registration as defined in 6.2.

NOTE - Only metric measurements (according to the international physical SI system) are allowed within ISO/IEC standards. Thus, "inches," for example, are not explicitly provided by the *MeasurementUnit* entity. Inches, feet, etc., are rendered as application-specific units of measurement. For application-specific measurements, a verbal definition can be given using the *verbal-description* component.

The *basis* component expresses a shift in the order of magnitude. Let *i* be the value of the *basis* component. Then, the shift is defined to be 10^i.

EXAMPLE - Millimeters are expressed by setting the *basis* to -3 and the *measurement* to "meter."

NOTE - Associating a dimension of an image structure to "time" as physical measurement provides a way to express synchronicity between parts of an image structure.

Syntax

```
MeasurementUnit ::= SEQUENCE
    {
    measurement          [0] IMPLICIT INTEGER                    -- No. 702
                             {
                             meter(1),
                             second(2),
                             degree(3),
                             hertz(4),
                             application-specific(5)
                             },
    basis                [1] IMPLICIT INTEGER DEFAULT 0,         -- No. 702
    verbal-description   [2] IMPLICIT CharacterString OPTIONAL   -- No. 006
    }
```

Constraints

For the *measurement* component, only values in the range of (1..5) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 405 *DeltaVector*

Semantics

The *DeltaVector* entity stands for the description of a sequence of delta values which apply to one dimension of a digital image. This is used when irregular gridding is desired.

The *i*-th value in the sequence specifies the delta between the *i*-th and the *i+1*-th index point of the respective dimension of the image array.

Syntax

```
DeltaVector ::= SEQUENCE OF REAL
```

Constraints

The number of sub-entities which is contained in the *DeltaVector* entity shall equal *n - 1* whereby *n* stands for the number of array elements that are declared for the respective dimension of the image array, given by the *dimension-identifier* component of the parent entity.

IIF syntax entity No. 406 *MetricTransformation*

Semantics

The *MetricTransformation* entity may be used to specify a transformation of the cartesian coordinate system of an image (described by the *CoordinateSystem* and *DimensionMapping* entities). It can be either a standard transformation, using the *standard-transformation* component, or a user-defined transformation, using the *matrix* component.

A standard scanning pattern can be defined by the *standard-transformation* component. Two alternatives are predefined:

orthogonal mode: the coordinate system and the physical measurement fully define the position of array elements in the coordinate space.

quincunx sweep mode: array elements on odd number "scan lines" have to be interpreted as being placed one-half of the "scanning distance" to the right.

Additional definitions are subject to registration as defined in 6.2.

NOTE - "Scan line" and "scanning resolution" are commonly used terms. According to the wording of this standard, "scan line" stands for all array elements that belong to a certain index in the dimension that defines the second loop over the indices. The term "scanning resolution" equals the delta within the *DimensionMapping* entity.

If the *matrix* component is used to associate a coordinate transformation *T* with an image, then the image data are given with respect to the transformed coordinate system. To obtain the image information at a certain position in the original coordinate system, the respective coordinates need to be transformed according to the transformation matrix *T* first.

Syntax

```
MetricTransformation ::= CHOICE
  {
  standard-transformation [0] IMPLICIT INTEGER                    -- No. 702
                                  {
                                  orthogonal(1),
                                  quincunx(2)
                                  },
  matrix                  [1] IMPLICIT TransformationMatrix       -- No. 318
  }
```

Constraints

For the *standard-transformation* component, only values in the range of (1..2) and values which have been internationally registered are permitted. Refer to 6.2.

See also constraints which apply to the *TransformationMatrix* entity.

IIF syntax entity No. 407 *Domain*

Semantics

The *Domain* entity describes whether the raster data directly specify intensity values or a transformation of such data into the transform domain (e.g., Fourier domain).

The *domain-type* component determines the domain of the raster data. The following alternatives are predefined: *space domain, cosine transform domain, Fourier transform domain, Hadamard transform domain, Hartley transform domain*, and *Karhunen-Loeve domain*. Operators for transformations among these domains are supported by ISO/IEC 12087-2. Additional definitions of specific transform domains are subject to registration as defined in 6.2.

The transform domain can further be described verbally by the *verbal-description* component.

NOTE - The verbal description of a certain unregistered transform domain need not necessarily be understood by other applications.

Syntax

```
Domain ::= SEQUENCE
    {
    domain-type           [0] IMPLICIT INTEGER                    -- No. 702
                                  {
                                  space(1),
                                  cosine(2),
                                  fourier(3),
                                  hadamard(4),
                                  hartley(5),
                                  karhunen-loeve(6)
                                  },
    verbal-description    [1] IMPLICIT CharacterString            -- No. 006
    }
```

Constraints

For the *domain-type* component, only values in the range of (1..1) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 408 *ChannelCharacteristics*

Semantics

The *ChannelCharacteristics* entity stands for the description of image capture-related attributes, as defined in ISO/IEC 12087-1.

The *band-identifier* component is used to associate the channel characteristics information with those bands of an image structure which have the same *band-identifier*. This component is optional. If it is absent, the channel characteristics description is associated with the whole image.

The *verbal-description* component provides a field for application-specific verbal descriptions of the channel characteristics.

The *precision* component specifies the number of decimal places that shall be regarded as significant. It may be used to describe the channel's acquisition accuracy.

The *channel-type* component may be used to express the type of channel that has been used to acquire or generate a channel (i.e., - in the wording of the data types defined in ISO/IEC 12087-1 - a band). The following alternatives are predefined: *bi-level*, *half-tone*, *grey-value*, *colour*, and *feature*. Additional definitions are subject to registration as defined in 6.2.

The *channel-usage* component may be used to express the usage of the channel with respect to the image's other channels (bands). The following alternatives are predefined: *opaque*, *transparent*. Additional definitions are subject to registration as defined in 6.2.

The *background-value* component may be used to express to what pixel value the image's background is assigned.

The *compandor-description* component allows to specify the compandor (compression/expander) model of the given channel.

The *sampling-function* component allows to specify the spatial sampling characteristics of the given channel.

The *application-specifics* component allows for application-specific extensions to the *ChannelCharacteristics* entity in general.

Syntax

```
ChannelCharacteristics ::= SEQUENCE
  {
  band-identifier        [0] IMPLICIT Identifier      OPTIONAL,-- No. 107
  verbal-description     [1] IMPLICIT CharacterString OPTIONAL,-- No. 006
  precision              [2] IMPLICIT INTEGER         OPTIONAL,-- No. 702
  channel-type           [3] IMPLICIT INTEGER                  -- No. 702
                             {
                             bi-level(1),
                             half-tone(2),
                             grey-value(3),
                             colour(4),
                             feature(5)
                             } OPTIONAL,
  channel-usage          [4] IMPLICIT INTEGER                  -- No. 702
                             {
                             opaque(1),
                             transparent(2)
                             } OPTIONAL,
  background-value       [5] BuiltinValue             OPTIONAL,-- No. 206
  compandor-description  [6] IMPLICIT
                             CompandorDescription     OPTIONAL,-- No. 409
  sampling-function      [7] ANY                      OPTIONAL,
  application-specifics  [8] ANY                      OPTIONAL
  }
```

Constraints

For the *channel-type* component, only values in the range of (1..5) and values which have been internationally registered are permitted. Refer to 6.2.

For the *channel-usage* component, only values in the range of (1..2) and values which have been internationally registered are permitted. Refer to 6.2.

For the *background-value* component, the type which is used to represent the value has to match the elementary pixel type of the pixel array to which this channel description is associated.

If a channel type is *colour*, the existence of a *ColourRepresentation* entity is required.

The encoded representation of a field of pixel values (*SingleDataUnit* entity) must provide at least as much bits per pixel as specified by the value of the *precision* component.

IIF syntax entity No. 409 *CompandorDescription*

Semantics

The *CompandorDescription* entity stands for the description of a compandor (compression/expander) model which is part of a channel's quantization characteristics specification:

A channel value v is defined as: $v = G(Q(f(u)))$

where u is the real world value, which is represented by the channel value v.

The function $f(u)$ is a compressor function, which is a transformation between the real world value u and the continuous variable w. The function Q is a uniform quantization function and G is an expander function. For $f(u)$, three different function types are being distinguished:

- linear: $w = a * u + w_0$
- gamma: $w = a * log_{10}(u) - w_0$
- exponential: $w = a * u^n$

The *function-type* component allows one to select the kind of transfer function. The following alternatives are predefined: *linear*, *gamma*, and *exponential*. Additional definitions are subject to registration as defined in 6.2.

The *transfer-factor* component represents the amplification factor a in the above-mentioned formulas. The *function-parameter* component represents the second parameter in one of the above-mentioned formulas. In the case of the *linear* and *gamma* transformations, it represents w_0; in the case of the *exponential* transformation it represents the exponent n. The *expander-function* component represents the function G and defines the mapping between the w space and the channel value v. This is done by a *LookUpTable* entity.

The *application-specific* component allows for application-specific extensions.

Syntax

```
CompandorDescription ::= SEQUENCE
    {
    function-type           [0] IMPLICIT INTEGER                -- No. 702
                                {
                                linear(1),
                                gamma(2),
                                exponential(3),
                                application-specific(4)
                                },
    transfer-factor         [1] IMPLICIT REAL,                  -- No. 709
    function-parameter      [2] IMPLICIT REAL,                  -- No. 709
    expander-function       [3] IMPLICIT LookUpTable OPTIONAL,  -- No. 304
    application-specific    [4] ANY OPTIONAL
    }
```

Constraints

For the *function-type* component, only values in the range of (1..3) and values which have been internationally registered are permitted. Refer to 6.2.

IIF syntax entity No. 410 *ColourRepresentation*

Semantics

The *ColourRepresentation* entity stands for the selection of a colour space including all attributes which are required to unambiguously describe the colour representation. According to the description in ISO/IEC 12087-1, two kinds of colour spaces are provided: standardized spaces that are based on the CIE-1931 colour system, given by the *standardized-space* component and non-standardized spaces, given by the *non-standardized-space* component.

Syntax

```
ColourRepresentation ::= SEQUENCE
   {
   colour-space-type CHOICE
      {
      standardized-space       [0] StandardizedSpace,          -- No. 411
      non-standardized-space   [1] NonStandardizedSpace        -- No. 424
      },
   test-colour   [2] IMPLICIT TestColour OPTIONAL             -- No. 433
   }
```

Constraints

See constraints of the subentities.

IIF syntax entity No. 411 *StandardizedSpace*

Semantics

The *StandardizedSpace* entity stands for the description of a colour space which is based on the 1931 standard colour system according to the CIE standard. The following spaces are provided: CIE spaces, linear RGB spaces, gamma RGB spaces, and luminance-chrominance spaces. For a description of these spaces and the methods on how to perform conversions between the spaces, refer to the corresponding subentities and to ISO/IEC 12087-1.

Syntax

```
StandardizedSpace ::= CHOICE
   {
   cie-xyz                 [0] CIEXYZSpace,                -- No. 412
   cie-yxy                 [1] CIEYxySpace,                -- No. 413
   cie-uvw                 [2] CIEUVWSpace,                -- No. 414
   cie-yuv                 [3] CIEYuvSpace,                -- No. 415
   cie-lab                 [4] CIELabSpace,                -- No. 416
   cie-luv                 [5] CIELuvSpace,                -- No. 417
   linear-rgb              [6] LinearRGBSpace,             -- No. 419
   gamma-rgb               [7] GammaRGBSpace,              -- No. 420
   yiq-colour-space        [8] YIQColourSpace,             -- No. 421
   yuv-colour-space        [9] YUVColourSpace,             -- No. 422
   ycbcr-colour-space     [10] YCbCrColourSpace            -- No. 423
   }
```

Constraints

None.

IIF syntax entity No. 412 ***CIEXYZSpace***

and

IIF syntax entity No. 413 ***CIEYxySpace***

Semantics

The *CIEXYZSpace* entity stands for the description of the CIE *XYZ* colour space which is based on the CIE-1931 colour system. See reference [CIE:1931] in ISO/IEC 12087-1.

The *CIEYxySpace* entity stands for the description of the CIE *Yxy* colour space which is also based on the CIE-1931 colour system. See reference [CIE:1931] in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
CIEXYZSpace ::= SEQUENCE
    {
    x-band-identifier  [0] IMPLICIT Identifier,          -- No. 107
    y-band-identifier  [1] IMPLICIT Identifier,          -- No. 107
    z-band-identifier  [2] IMPLICIT Identifier           -- No. 107
    }

CIEYxySpace ::= SEQUENCE
    {
    yl-band-identifier  [0] IMPLICIT Identifier,         -- No. 107
    xc-band-identifier  [1] IMPLICIT Identifier,         -- No. 107
    yc-band-identifier  [2] IMPLICIT Identifier          -- No. 107
    }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 414 ***CIEUVWSpace***

and

IIF syntax entity No. 415 ***CIEYuvSpace***

Semantics

The *CIEUVWSpace* entity stands for the description of the CIE *UVW* colour space which is based on the CIE-1931 colour system. See reference [CIE:1970] in ISO/IEC 12087-1.

The *CIEYuvSpace* entity stands for the description of the CIE *Yuv* colour space which is based on the CIE-1931 colour system. See reference [CIE:1970] in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
CIEUVWSpace ::= SEQUENCE
    {
    u-band-identifier  [0] IMPLICIT Identifier,          -- No. 107
    v-band-identifier  [1] IMPLICIT Identifier,          -- No. 107
    w-band-identifier  [2] IMPLICIT Identifier           -- No. 107
    }


CIEYuvSpace ::= SEQUENCE
    {
    y-band-identifier  [0] IMPLICIT Identifier,          -- No. 107
    u-band-identifier  [1] IMPLICIT Identifier,          -- No. 107
    v-band-identifier  [2] IMPLICIT Identifier           -- No. 107
    }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 416 ***CIELabSpace***

and

IIF syntax entity No. 417 ***CIELuvSpace***

Semantics

The *CIELabSpace* entity stands for the description of the CIE $L^*a^*b^*$ colour space which is based on the CIE-1931 colour system. See reference [CIE:1976] in ISO/IEC 12087-1.

The *CIELuvSpace* entity stands for the description of the CIE $L^*u^*v^*$ colour space which is based on the CIE-1931 colour system. See reference [CIE:1976] in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

The *white-point* component allows for the description of a white point for the colour space. It is given as a CIE coordinate.

Syntax

```
CIELabSpace ::= SEQUENCE
    {
    l-band-identifier   [0] IMPLICIT Identifier,                -- No. 107
    a-band-identifier   [1] IMPLICIT Identifier,                -- No. 107
    b-band-identifier   [2] IMPLICIT Identifier,                -- No. 107
    white-point         [3] IMPLICIT CIEXYZCoordinate           -- No. 418
    }

CIELuvSpace ::= SEQUENCE
    {
    l-band-identifier   [0] IMPLICIT Identifier,                -- No. 107
    u-band-identifier   [1] IMPLICIT Identifier,                -- No. 107
    v-band-identifier   [2] IMPLICIT Identifier,                -- No. 107
    white-point         [3] IMPLICIT CIEXYZCoordinate           -- No. 418
    }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 418 *CIEXYZCoordinate*

Semantics

The *CIEXYZCoordinate* entity stands for the description of an X,Y,Z coordinate based on the CIE-1931 standard colour system.

Syntax

```
CIEXYZCoordinate ::= SEQUENCE
   {
   cie-x      [0] IMPLICIT REAL,                    -- No. 709
   cie-y      [1] IMPLICIT REAL,                    -- No. 709
   cie-z      [2] IMPLICIT REAL                     -- No. 709
   }
```

Constraints

None.

IIF syntax entity No. 419 *LinearRGBSpace*

Semantics

The *LinearRGBSpace* entity stands for the description of the linear RGB colour space that conforms to one of the following standards: *NTSC*, *EBU*, *SMPTE*, *CCIR-709*.

The *representation* component allows for the selection of one of the standards mentioned above:

NTSC: according to the references [CCIR:1990a] and [CCIR:1990b], given in ISO/IEC 12087-1.

EBU: according to the references [CCIR:1990a], [CCIR:1990b], and [EBU:1975], given in ISO/IEC 12087-1.

SMPTE: according to the reference [DeMarsh:1974], given in ISO/IEC 12087-1.

CCIR-709: according to the reference [CCIR:1990c], given in ISO/IEC 12087-1.

Additional definitions are subject to registration as defined in 6.2.

The *illuminant* component allows for the selection of an illuminant. If this component is missing, the illuminant is defined by the selected colour space standard. The following values are defined (see ISO/IEC 12087-1): *D65* and *C*. Additional definitions are subject to registration as defined in 6.2.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
LinearRGBSpace ::= SEQUENCE
   {
   representation      [0] IMPLICIT INTEGER                    -- No. 702
                           {
                           ntsc(1),
                           ebu(2),
                           smpte(3),
                           ccir-709(4)
                           },
   illuminant          [1] IMPLICIT INTEGER                    -- No. 702
                           {
                           d65(1),
                           c(2)
                           } OPTIONAL,
   r-band-identifier   [2] IMPLICIT Identifier,                -- No. 107
   g-band-identifier   [3] IMPLICIT Identifier,                -- No. 107
   b-band-identifier   [4] IMPLICIT Identifier                 -- No. 107
   }
```

Constraints

For the *representation* component, only values in the range of (1..4) and values which have been internationally registered are permitted. For the *illuminant* component, only values in the range of (1..2) and values which have been internationally registered are permitted. Refer to 6.2.

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 420 *GammaRGBSpace*

Semantics

The *GammaRGBSpace* entity stands for the description of the gamma corrected RGB colour space that conforms to one of the following standards: *NTSC*, *EBU*, *SMPTE*, *CCIR-709*.

The *representation* component allows for the selection of one of the standards mentioned above:

NTSC: according to the reference [CCIR:1990b], given in ISO/IEC 12087-1.

EBU: according to the reference [CCIR:1990b], given in ISO/IEC 12087-1.

SMPTE: according to the reference [DeMarsh:1974], given in ISO/IEC 12087-1.

CCIR-709: according to the reference [CCIR:1990c], given in ISO/IEC 12087-1.

Additional definitions are subject to registration as defined in 6.2.

The *illuminant* component allows for the selection of an illuminant. If this component is missing, the illuminant is defined by the selected colour space standard. The following values are defined: *D65* and *C*.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
GammaRGBSpace ::= SEQUENCE
    {
    representation      [0] IMPLICIT INTEGER                    -- No. 702
                            {
                            ntsc(1),
                            ebu(2),
                            smpte(3),
                            ccir-709(4)
                            },
    illuminant          [1] IMPLICIT INTEGER                    -- No. 702
                            {
                            d65(1),
                            c(2)
                            } OPTIONAL,
    r-band-identifier   [2] IMPLICIT Identifier,                -- No. 107
    g-band-identifier   [3] IMPLICIT Identifier,                -- No. 107
    b-band-identifier   [4] IMPLICIT Identifier                 -- No. 107
    }
```

Constraints

For the *representation* component, only values in the range of (1..4) and values which have been internationally registered are permitted. For the *illuminant* component, only values in the range of (1..2) and values which have been internationally registered are permitted. Refer to 6.2.

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities)

they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 421 *YIQColourSpace*

Semantics

The *YIQColourSpace* entity stands for the description of the video-oriented *YIQ* colour space which is based on the CIE-1931 colour system, according to the reference [NTSC:1954], given in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
YIQColourSpace ::= SEQUENCE
   {
   y-band-identifier      [0] IMPLICIT Identifier,          -- No. 107
   i-band-identifier      [1] IMPLICIT Identifier,          -- No. 107
   q-band-identifier      [2] IMPLICIT Identifier           -- No. 107
   }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 422 ***YUVColourSpace***

Semantics

The *YUVColourSpace* entity stands for the description of the video-oriented *YUV* colour space which is based on the CIE-1931 colour system, according to the reference [CCIR:1990b], given in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
YUVColourSpace ::= SEQUENCE
    {
    y-band-identifier      [0] IMPLICIT Identifier,          -- No. 107
    u-band-identifier      [1] IMPLICIT Identifier,          -- No. 107
    v-band-identifier      [2] IMPLICIT Identifier           -- No. 107
    }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 423 *YCbCrColourSpace*

Semantics

The *YCbCr* entity stands for the description of a video-oriented YC_bC_r colour space which is based on the CIE-1931 colour system. It conforms to the *SMPTE* Standard according to the reference [DeMarsh:1974], given in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

Syntax

```
YCbCrColourSpace ::= SEQUENCE
   {
   y-band-identifier      [0] IMPLICIT Identifier,          -- No. 107
   cb-band-identifier     [1] IMPLICIT Identifier,          -- No. 107
   cr-band-identifier     [2] IMPLICIT Identifier           -- No. 107
   }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 424 *NonStandardizedSpace*

Semantics

The *NonStandardizedSpace* entity stands for the description of a non-standardized colour space. Five colour spaces are provided: *RGB*, *IHS*, *CMY*, *CMYK*, and arbitrary multi-band, given by the *rgb*, *ihs*, *cmy*, *cmyk*, and *n-band* components.

NOTE - The *ChannelCharacteristics* entity may be used in combination with the *NonStandardizedSpace* entity in order to further specify the band characteristics of user-defined colour bands.

Syntax

```
NonStandardizedSpace ::= CHOICE
   {
   rgb       [0] NonStandardizedRGB,                    -- No. 425
   ihs       [1] NonStandardizedIHS,                    -- No. 426
   cmy       [2] NonStandardizedCMY,                    -- No. 429
   cmyk      [3] NonStandardizedCMYK,                   -- No. 430
   n-band    [4] NonStandardizedNBand                   -- No. 431
   }
```

Constraints

None.

IIF syntax entity No. 425 ***NonStandardizedRGB***

and

IIF syntax entity No. 426 ***NonStandardizedIHS***

Semantics

The *NonStandardizedRGB* entity stands for the description of a non-standardized RGB space as described in ISO/IEC 12087-1.

The *NonStandardizedIHS* entity stands for the description of a non-standardized IHS space as described in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

The *primaries* component allows for the description of three primary chromaticities, one for each band.

The *white-point* component allows for the description of a white point for the colour space.

Syntax

```
NonStandardizedRGB ::= SEQUENCE
    {
    r-band-identifier   [0] IMPLICIT Identifier,              -- No. 107
    g-band-identifier   [1] IMPLICIT Identifier,              -- No. 107
    b-band-identifier   [2] IMPLICIT Identifier,              -- No. 107
    white-point         [3] IMPLICIT CIEXYZCoordinate,        -- No. 418
    primaries           [4] IMPLICIT Primaries                -- No. 427
    }

NonStandardizedIHS ::= SEQUENCE
    {
    i-band-identifier   [0] IMPLICIT Identifier,              -- No. 107
    h-band-identifier   [1] IMPLICIT Identifier,              -- No. 107
    s-band-identifier   [2] IMPLICIT Identifier,              -- No. 107
    white-point         [3] IMPLICIT CIEXYZCoordinate,        -- No. 418
    primaries           [4] IMPLICIT Primaries                -- No. 427
    }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 427 *Primaries*

Semantics

The *Primaries* entity stands for the description of three primary chromaticity values, one for each band of the colour space description to which it is attached. The values are given as CIE coordinates.

Syntax

```
Primaries ::= SEQUENCE
    {
    r-primary    [0] IMPLICIT CIExyCoordinate,          -- No. 428
    g-primary    [1] IMPLICIT CIExyCoordinate,          -- No. 428
    b-primary    [2] IMPLICIT CIExyCoordinate           -- No. 428
    }
```

Constraints

None.

IIF syntax entity No. 428 *CIExyCoordinate*

Semantics

The *CIExyCoordinate* entity stands for the description of a x,y coordinate based on the CIE-1931 standard colour system.

Syntax

```
CIExyCoordinate ::= SEQUENCE
   {
   cie-x      [0] IMPLICIT REAL,                    -- No. 709
   cie-y      [1] IMPLICIT REAL                     -- No. 709
   }
```

Constraints

None.

IIF syntax entity No. 429 ***NonStandardizedCMY***

and

IIF syntax entity No. 430 ***NonStandardizedCMYK***

Semantics

The *NonStandardizedCMY* entity stands for the description of a non-standardized subtractive CMY space as described in ISO/IEC 12087-1.

The *NonStandardizedCMYK* entity stands for the description of a non-standardized subtractive CMYK space as described in ISO/IEC 12087-1.

The *"...-identifier"* components are used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

The *verbal-description* component allows for a verbal description of the characteristics of the colour space.

Syntax

```
NonStandardizedCMY ::= SEQUENCE
   {
   c-band-identifier  [0] IMPLICIT Identifier,                 -- No. 107
   m-band-identifier  [1] IMPLICIT Identifier,                 -- No. 107
   y-band-identifier  [2] IMPLICIT Identifier,                 -- No. 107
   verbal-description [3] IMPLICIT CharacterString OPTIONAL    -- No. 006
   }

NonStandardizedCMYK ::= SEQUENCE
   {
   c-band-identifier  [0] IMPLICIT Identifier,                 -- No. 107
   m-band-identifier  [1] IMPLICIT Identifier,                 -- No. 107
   y-band-identifier  [2] IMPLICIT Identifier,                 -- No. 107
   k-band-identifier  [3] IMPLICIT Identifier,                 -- No. 107
   verbal-description [4] IMPLICIT CharacterString OPTIONAL    -- No. 006
   }
```

Constraints

The values of the *"...-identifier"* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 431 ***NonStandardizedNBand***

and

IIF syntax entity No. 432 ***ColourBand***

Semantics

The *NonStandardizedNBand* entity stands for the description of a non-standardized multi-band colour space.

The *number-of-bands* component gives the dimensionality of the colour space. The *band-descriptions* component contains a description of each band.

The *ColourBand* entity stands for the description of one non-standardized colour band.

The *band-identifier* component is used to associate the colour space with those bands declared within the image structure description which have the same *band-identifier*.

The *verbal-description* component allows for a description of the semantical meaning of the colour band.

Syntax

```
NonStandardizedNBand ::= SEQUENCE
   {
   number-of-bands      [0] IMPLICIT INTEGER (1..MAX),          -- No. 702
   band-descriptions    [1] IMPLICIT SEQUENCE OF ColourBand     -- No. 432
   }


ColourBand ::= SEQUENCE
   {
   band-identifier      [0] IMPLICIT Identifier,                -- No. 107
   verbal-description   [1] IMPLICIT CharacterString            -- No. 006
   }
```

Constraints

The value of the *number-of-bands* component shall equal the number of *ColourBand* entities which are contained in the *band-description* component.

The values of the *band-identifier* components shall equal either the values of the corresponding *band-identifier* components (within the pixel data type declaration of the *FundamentalImageStructure* entities) they refer to or they shall equal the values of the component identifiers which are declared within the input and output data types of the look-up tables. For a description of the relation between look-up tables and colour spaces, refer to the *LookUpTable* entity.

IIF syntax entity No. 433 *TestColour*

Semantics

The *TestColour* entity stands for the description of a test colour with respect to a region of interest. The components are called *colour* and *region*, respectively. The region of interest determines the geometrical position of the test colour in the image. The colour is given as CIE coordinate.

The *TestColour* entity serves as a mechanism to prove whether correct colour reproduction has been achieved during a rendering process. For this purpose, colourimetric values need to be taken by measurement at the location marked by the *region* component. For correct colour reproduction, these values have to match the value of the associated *colour* component.

Syntax

```
TestColour ::= SEQUENCE
   {
   colour    [0] IMPLICIT CIEXYZCoordinate,                    -- No. 418
   region    [1] RegionOfInterest                              -- No. 305
   }
```

Constraints

None.

IIF syntax entity No. 434 *PIKSControl*

Semantics

The *PIKSControl* entity stands for the description of attributes for the control of the IPI-PIKS, as defined in ISO/IEC 12087-1.

The *roi* component is used to specify a region of interest (ROI) and the *roi-offset* component is used to specify an ROI offset. The *match-point* component is used to describe a match point.

For the use of these components to control image processing functions, refer to ISO/IEC 12087-2.

Syntax

```
PIKSControl ::= SEQUENCE
   {
   roi               [0] RegionOfInterest,                -- No. 305
   roi-offset        [1] CoordinateND,                    -- No. 107
   match-point       [2] CoordinateND                     -- No. 310
   }
```

Constraints

None.

5.3.6 Entities for the description of image annotations

IIF syntax entity No. 501 ***ImageAnnotation***

Semantics

The *ImageAnnotation* entity stands for the description of text, graphics, audio, or other application-related data. The annotation may be semantically attached to an image as a whole or to a specific location within the image using the *location* component.

The *numeric-identifier* component provides one of two ways to specify how the *contents* component has to be interpreted. The following alternatives are predefined:

"1": *plain text contents*

"2": *structured text contents* according to ISO/IEC 8613, Open Document Architecture (ODA), Office Document Format (FOD) 11 (simple text documents), ISO/IEC 10000.

"3": *structured text contents* according to ISO/IEC 8879, Standard Generalized Markup Language (SGML), represented according to ISO/IEC 9069, SGML Document Interchange Format (SDIF).

"4": *geometric graphics contents* according to ISO/IEC 8632, Computer Graphics Metafile (CGM).

"5": *geometric graphics contents* according to ISO/IEC 8632, Computer Graphics Metafile (CGM), Revision 1992.

"6": *audio contents* according to CCITT Recommendation G.711, encoding type *aLaw*.

"7": *audio contents* according to CCITT Recommendation G.711, encoding type *muLaw*.

"8": *audio contents* according to CCITT Recommendation G.721.

"9": *audio contents* according to ISO/IEC 11172 (MPEG-1).

Additional definitions are subject to registration as defined in 6.2.

As an alternative, the *object-identifier* component may be used to specifies how the *contents* component has to be interpreted by identifying an officially registered object.

The contents of the annotation is either directly represented by the *contents* component, or via a reference to an image data unit, using the *reference-to-image-data* component. In the former case, the syntax of the *contents* component depends on the contents type as specified by either the *numeric-identifier* component or the *object-identifier* component. The latter case may be used to refer to an audio data stream that is stored interleaved with image data within a *ReferencedUnit* entity, e.g., using the MPEG-1 encoding.

NOTE - The semantics contained in an image annotation are only meaningful to certain applications. No meaning is defined throughout ISO/IEC 12087. However, it is assumed that the application-specific semantics contained in a certain image annotation are related to all substructures with which they are associated. If *ImageAnnotation* entities occur, the IIF parser shall switch to the text, graphics, audio, or application-specific processor as indicated by either the *numeric-identifier* or *object-identifier* component.

Syntax

```
ImageAnnotation ::= SEQUENCE
   {
   location                    [0] Location OPTIONAL,          -- No. 502
   contents-descriptor CHOICE
      {
      numeric-identifier       [1] IMPLICIT INTEGER            -- No. 702
                                      {
                                      plain-text(1),
                                      oda-fod-11(2),
                                      sgml(3),
                                      cgm-87(4),
                                      cgm-92(5),
                                      g711-a-law(6),
                                      g711-mu-law(7),
                                      g721(8),
                                      mpeg-1(9)
                                      },
      object-identifier        [2] IMPLICIT OBJECT IDENTIFIER -- No. 706
      },
   contents-type CHOICE
      {
      contents                 [3] ANY,
      reference-to-image-data  [4] IMPLICIT Identifier         -- No. 107
      }
   }
```

Constraints

For the *numeric-identifier* component, only values in the range of (1..9) and values which have been internationally registered are permitted. Refer to 6.2.

For contents of type *plain-text*, the syntax of the *contents* component shall be CharacterString. For the other content types, the syntax depends on the encoded representation provided by the referenced standards:

a) clear text encoded content types shall be represented by CharacterString

b) binary encoded content types shall be represented by OCTET STRING

c) content types whose representations are given by an ASN.1 syntax shall be represented according to this syntax.

The *reference-to-image-data* component may only be chosen in combination with a *ReferencedUnit* entity which contains interleaved data in a well-defined representation.

NOTE - The only well-defined representation for image and audio data defined by this Standard is MPEG-1. Future representations are subject to registration. Refer to 6.2.

IIF syntax entity No. 502 ***Location***

Semantics

The *Location* entity stands for the description of a location within an image. The location may either be a single *point* or a *region*, represented by the *RegionOfInterest* entity.

Syntax

```
Location ::= CHOICE
   {
   point      [0] IndexND,                                  -- No. 313
   region     [1] RegionOfInterest                          -- No. 305
   }
```

Constraints

None.

5.3.7 Entities for the description of basic data object

IIF syntax entity No. 601 *BasicDataObject*

Semantics

The *BasicDataObject* entity covers the description of data that are structured according to the basic data types as defined in ISO/IEC 12087-1. The *basic-data-type* component covers type information. The *data* component covers the data values that belong to the type definition.

Syntax

```
BasicDataObject ::= SEQUENCE
   {
   basic-data-type    [0] BasicDataType,                    -- No. 602
   data               [1] BuiltinEncodedDataUnit            -- No. 205
   }
```

Constraints

None.

IIF syntax entity No. 602 ***BasicDataType***

and

IIF syntax entity No. 603 ***CompoundDataType***

Semantics

The *BasicDataType* entity stands for the description of a basic data type, as defined in ISO/IEC 12087-1. The components are called *compound-data-type* and *elementary-data-type*, respectively.

The *CompoundDataType* entity covers the description of a compound data type, as defined in ISO/IEC 12087-1. It can be an array of basic data types, called a *basic-array*, a record of basic data types, called a *basic-record*, a list of basic data types, called a *basic-list*, or a set of basic data types, called *basic-set*.

Syntax

```
BasicDataType ::= CHOICE
    {
    compound-data-type      [0] CompoundDataType,                    -- No. 603
    elementary-data-type    [1] ElementaryDataType                   -- No. 609
    }


CompoundDataType ::= CHOICE
    {
    basic-array     [0] BasicArray,                                  -- No. 604
    basic-record    [1] BasicRecord,                                 -- No. 605
    basic-list      [2] BasicList,                                   -- No. 607
    basic-set       [3] BasicSet                                     -- No. 608
    }
```

Constraints

None.

IIF syntax entity No. 604 *BasicArray*

Semantics

The *BasicArray* entity stands for the description of an array of basic data types, as defined in ISO/IEC 12087-1. The dimensionality and the index ranges are given by the *dimensionality* component. The sequential order of array elements is determined by the *serialization* component. A description of the array's element structure is given by the *element-type* component.

Syntax

```
BasicArray ::= SEQUENCE
   {
   dimensionality          [0] IMPLICIT Dimensionality,          -- No. 105
   serialization           [1] IMPLICIT Serialization,           -- No. 108
   element-type            [2] BasicDataType                     -- No. 602
   }
```

Constraints

None.

IIF syntax entity No. 605 ***BasicRecord***

and

IIF syntax entity No. 606 ***BasicRecordComponent***

Semantics

The *BasicRecord* entity stands for the description of a record of basic data types, as defined in ISO/IEC 12087-1. The number of record components is given by the *number-of-components* component. Then, all record components are given by the *basic-record-components* component.

The *BasicRecordComponent* entity consists of an identifier, called the *component-identifier*, and the data type, called the *component-data-type*.

Syntax

```
BasicRecord ::= SEQUENCE
   {
   number-of-components      [0] IMPLICIT INTEGER (1..MAX),     -- No. 702
   basic-record-components   [1] IMPLICIT SEQUENCE OF
                                    BasicRecordComponent -- No. 606
   }


BasicRecordComponent ::= SEQUENCE
   {
   component-identifier   [0] IMPLICIT Identifier,            -- No. 107
   component-data-type    [1] BasicDataType                   -- No. 602
   }
```

Constraints

The number of *BasicRecordComponent* entities contained in the following sequence shall equal the value of the *number-of-components* component.

The character strings of the *component-identifier* components shall be different from each other in order to allow unambiguous referral to record components.

IIF syntax entity No. 607 *BasicList*

Semantics

The *BasicList* entity stands for the description of a list of basic data types, as defined in ISO/IEC 12087-1. It contains the number of list elements by the *number-of-elements* component, followed by the *element-type* component which determines the structure of all list elements.

Syntax

```
BasicList ::= SEQUENCE
   {
   number-of-elements    [0] IMPLICIT INTEGER (1..MAX),      -- No. 702
   element-type          [1] BasicDataType                   -- No. 602
   }
```

Constraints

The number of *BasicDataType* entities contained in the following sequence shall equal the value of the *number-of-elements* entity.

IIF syntax entity No. 608 *BasicSet*

Semantics

The *BasicSet* entity stands for the description of a set of basic data types, as defined in ISO/IEC 12087-1. It contains the number of set members by the *number-of-members* component, followed by the *member-type* component which determines the structure of all set members.

Syntax

```
BasicSet ::= SEQUENCE
    {
    number-of-members    [0] IMPLICIT INTEGER (1..MAX),      -- No. 702
    member-type          [1] BasicDataType                   -- No. 602
    }
```

Constraints

The number of *BasicDataType* entities contained in the following sequence shall equal the value of the *number-of-members* entity.

IIF syntax entity No. 609 ***ElementaryDataType***

Semantics

The *ElementaryDataType* entity stands for the description of an elementary data type, as defined in ISO/IEC 12087-1. The following alternatives are provided: *boolean, non-negative integer, signed integer, real, complex,* and *enumeration.*

Syntax

```
ElementaryDataType ::= INTEGER                                    -- No. 702
                       {
                       boolean(1),
                       non-negative-integer(2),
                       signed-integer(3),
                       real(4),
                       complex(5),
                       enumeration(6)
                       }
```

Constraints

For the *ElementaryDataType* entity, only values in the range of (1..6) and values which have been internationally registered are permitted. Refer to 6.2.

6 IPI-IIF Conformance

ISO/IEC 12087-1 establishes the concept of profiles. In general, a profile is used to specify a certain level of conformance that needs to be supported by an information processing system which shall conform to the Standard.

Profiles for the IIF-DF support a variety of image interchange features. The structure of the profiles places restrictions on the complexity of images in order to allow for the development of IIF-related information processing systems (parser, generator, gateway, and the like) that are reduced in complexity.

Two kinds of profiles are to be distinguished:

a) standardized profiles: These are the profiles which are provided by ISO/IEC 12087. They are given in 6.1.

b) registered profiles: These are future profiles which shall fulfill the needs of certain application domains. The guidelines on how a user group may specify and register such a profile is provided in 6.2.

6.1 Standardized profiles for the IIF-DF

The profiles for the IIF-DF are defined by a number of constraints that shall hold for the syntax of the IIF-DF in addition to the list of constraints that holds for the full profile. These constraints are expressed in a verbal manner as outlined in 5.1.4. In order to be syntactically compatible to the full profile of the IIF-DF, the profiles do not contain any changes in syntax.

The following three profiles are standardized:

a) Full profile of the IIF-DF:

This profile fully supports the whole IIF-DF. No further constraints are stated. This profile is a true superset of the other profiles. It is defined in clause 5.

b) Full PIKS profile of the IIF-DF:

This is the only application profile which is standardized within this part of ISO/IEC 12087. It fully supports all image data objects that can be processed by means of the Programmer's Imaging Kernel System (IPI-PIKS). With regard to data structures, the full PIKS profile of the IIF-DF conforms to the IPI-PIKS technical profile, the IPI-PIKS scientific profile, and the IPI-PIKS full profile as described in ISO/IEC 12087-1. The full PIKS profile of the IIF-DF is a true superset of profile c). It is defined in 6.1.1.

c) Foundation profile of the IIF-DF:

This profile forms the foundation profile for the IIF-DF. It meets the requirements of colour image and greyscale applications (e.g., desktop publishing). With regard to data structures, it conforms to the PIKS foundation profile. It is defined in 6.1.2.

6.1.1 Full PIKS profile of the IIF-DF

The full PIKS profile of the IIF-DF is defined by a list of constraints that apply to the syntax rules of the full IIF-DF syntax, described in clause 5. For all other rules and the description of the semantics, refer to clause 5.

Three cases are distinguished: the *band-interleaved* case, the *pixel-interleaved* case and the *homogeneous-bands* case. In the band-interleaved case, images are defined as records of arrays of elementary pixels. In the pixel-interleaved case, images are defined as arrays of records. In both cases the array dimensionality may be either two, three or four. In the homogeneous-bands case, images are defined as arrays of up to five dimensions of elementary pixels. In the latter case, one of the dimensions plays the role of a collection of (homogeneous) bands. No colour or channel attributes may be used in this case, because the syntax supports the mapping of these attributes only to image bands which are defined as record components but not to image bands which are defined via an array index.

The following constraints hold for all three cases:

- **IIF syntax entity *Profile***

  The value of this entity shall be *"full PIKS profile"*.

- **IIF syntax entity *ContentsBody***

  It is not allowed to select the *basic-data-object* component.

- **IIF syntax entity *Image***

  The *image-data* component may contain only one *ReferencedUnit* entity.

- **IIF syntax entity *ImageStructure***

  It is not allowed to select the *compound-image-structure* component.

- **IIF syntax entity *BandRecordComponent***

  In the case of colour images, the values of the *band-identifier* components shall conform to the mappings between colours and image channels which are specified in ISO/IEC 12087-1.

- **IIF syntax entity *MetricArrayElement***

  The *pixel-structure* component shall be selected.

- **IIF syntax entity *ElementaryPixelStructure***

  The value for the *ElementaryPixelStructure* entity shall be in the range of [1..5] in order to prohibit the use of the pixel type *enumerated*.

- **IIF syntax entity *PixelBandRecordComponent***

  In the case of colour images, the values of the *band-identifier* components shall conform to the mappings between colours and image channels which are specified in ISO/IEC 12087-1.

- **IIF syntax entity *DataUnit***

  The *external-data-unit* component shall not be selected.

- **IIF syntax entity *SingleDataUnit***

  The *registered-data-unit* and *compressed-data-unit* components shall not be selected.

- **IIF syntax entity *ImageRelatedData***

  The types of image-related data are limited to *histogram*, *look-up-table*, *region-of-interest*, *neighbourhood-array*, *static-array*, *value-bounds-collection*, *matrix*, *pixel-record*, and *tuple*.

- **IIF syntax entity *Histogram***

  Within the *class-description* component, only the elementary data types Boolean, non-negative integer, signed, integer, real, and complex may be chosen.

- **IIF syntax entity *LookUpTable***

  Within the *input-data-types* component, only the elementary data types non-negative integer and signed integer may be chosen. Within the *output-data-types* component, the elementary data types Boolean, non-negative integer, signed integer, real, and complex or *BasicRecord* entities of the named elementary data types may be chosen.

- **IIF syntax entity *RegionOfInterest***

  For all components, the dimensionality is limited to 5. The *polarity-reversed* component may only be used in combination with ROIs of type *BooleanArray* and *SetOfCoordinates*.

- **IIF syntax entity *NeighbourhoodArray***

  For all components, the dimensionality is limited to 5. Within the *element-structure* component, only Boolean, non-negative integer, signed integer, real, and complex may be chosen.

- **IIF syntax entity *StaticArray***

  For all components, the dimensionality is limited to 5. Within the *element-structure* component, only Boolean, non-negative integer, signed integer, real, and complex may be chosen.

- **IIF syntax entity *ValueBoundsCollection***

  Within the *pixel-data-type* component, only the elementary data types Boolean, non-negative integer, signed integer, real, and complex or *BasicRecord* entities of the named elementary data types may be chosen.

- **IIF syntax entity *PixelRecordComponent***

  Within the *component-value* entity, only Boolean, non-negative integer, signed integer, and real values may be chosen.

- **IIF syntax entity *Tuple***

  Within the *element-structure* component, only Boolean, non-negative integer, signed integer, and real may be chosen.

- **IIF syntax entity *ImageAttribute***

  The *freeform-attribute* component may not be selected.

- **IIF syntax entity *PIKSControl***

  For all components, the dimensionality is limited to 5.

The following constraints hold only for the band-interleaved case:

- **IIF syntax entity *Dimensionality***

  The value of the *number-of-dimensions* component shall be in the range of [2..4].

- **IIF syntax entity *DimensionDescription***

  The values of the *dimension-identifiers* shall be "x," "y," "z," and "t."

- **IIF syntax entity *FundamentalImageStructure***

  The *band-record* component shall be selected.

- **IIF syntax entity *PixelStructure***

  The *elementary-pixel-structure* shall be selected.

The following constraints hold only for the pixel-interleaved case:

- **IIF syntax entity *Dimensionality***

  The value of the *number-of-dimensions* component shall be in the range of [2..4].

- **IIF syntax entity *DimensionDescription***

  The values of the *dimension-identifiers* shall be "x," "y," "z," and "t."

- **IIF syntax entity *FundamentalImageStructure***

  The *metric-array* component shall be selected.

- **IIF syntax entity *PixelStructure***

  The *compound-pixel-structure* component shall be selected.

The following constraints hold only for the homogeneous-bands case:

- **IIF syntax entity *Dimensionality***

  The value of the *number-of-dimensions* component shall be in the range of [2..5].

- **IIF syntax entity *DimensionDescription***

  The values of the *dimension-identifiers* shall be "x," "y," "z," "t," and "b."

- **IIF syntax entity *FundamentalImageStructure***

  The *metric-array* component shall be selected.

- **IIF syntax entity *PixelStructure***

  The *elementary-pixel-structure* shall be selected.

6.1.2 Foundation profile of the IIF-DF

The foundation profile of the IIF-DF is defined by a list of constraints that apply to the syntax rules of the full IIF-DF syntax, described in clause 5. For all other rules and the description of the semantics, refer to clause 5.

Two cases are distinguished: the *band-interleaved* case and the *pixel-interleaved* case. In the band-interleaved case, images are defined as records of 2D arrays of elementary pixels; in the pixel-interleaved case, images are defined as 2D arrays of records.

The following constraints hold for both cases:

- **IIF syntax entity *Profile***

  The value of this entity shall be *"foundation profile"*.

- **IIF syntax entity *ContentsBody***

  It is not allowed to select the *basic-data-object* component.

- **IIF syntax entity *Image***

  The *image-data* component may contain only one *ReferencedUnit* entity.

- **IIF syntax entity *ImageStructure***

  It is not allowed to select the *compound-image-structure* component.

- **IIF syntax entity *Dimensionality***

  The value of the *number-of-dimensions* component shall be 2.

- **IIF syntax entity *DimensionDescription***

  The values of the *dimension-identifiers* shall be "*x*" and "*y*".

- **IIF syntax entity *BandRecord***

  The value of the *number-of-bands* component is limited to 3 for all colour spaces with the exception of the *CMYK* colour space. In the latter case, 4 bands are allowed.

- **IIF syntax entity *BandRecordComponent***

  In the case of colour images, the values of the *band-identifier* components shall conform to the mappings between colours and image channels which are specified in ISO/IEC 12087-1.

- **IIF syntax entity *MetricArrayElement***

  The *pixel-structure* component shall be selected.

- **IIF syntax entity *ElementaryPixelStructure***

  Only the values 1, 2, and 3 are allowed for the *ElementaryPixelStructure* entity in order to restrict the pixel structures to Boolean, non-negative integer, and integer.

- **IIF syntax entity *PixelBandRecord***

  The value of the *number-of-bands* component is limited to 3 for all colour spaces with the exception of the *CMYK* colour space. In the latter case, 4 bands are allowed.

- **IIF syntax entity *PixelBandRecordComponent***

  The values of the *band-identifier* components shall conform to the mappings between colours and image channels which are specified in ISO/IEC 12087-1.

- **IIF syntax entity *DataUnit***

  The *external-data-unit* component shall not be selected.

- **IIF syntax entity *SingleDataUnit***

  The *registered-data-unit* and *compressed-data-unit* components shall not be selected.

- **IIF syntax entity *ImageRelatedData***

  The types of image-related data are limited to *histogram*, *look-up-table*, *region-of-interest*, *neighbourhood-array*, *matrix*, and *tuple*.

- **IIF syntax entity *Histogram***

  Within the *class-description* component, only non-negative integer may be chosen.

- **IIF syntax entity *LookUpTable***

  Within the *input-data-types* component, only the elementary data types non-negative integer and signed integer may be chosen. Within the *output-data-types* component, only Boolean, non-negative integer, and signed integer may be chosen.

- **IIF syntax entity *RegionOfInterest***

  For all components, the dimensionality is limited to 2. The *polarity-reversed* component may only be used in combination with ROIs of type *BooleanArray* and *SetOfCoordinates*.

- **IIF syntax entity *CoordinateND***

  The *real-vector* component shall not be selected.

- **IIF syntax entity *NeighbourhoodArray***

  For all components, the dimensionality is limited to 2. Within the *element-structure* component, only Boolean, non-negative integer, signed integer, and real may be chosen.

- **IIF syntax entity *Tuple***

  Within the *element-structure* component, only Boolean, non-negative integer, signed integer, and real may be chosen.

- **IIF syntax entity *ImageAttribute***

  The *freeform-attribute* component may not be selected.

- **IIF syntax entity *PIKSControl***

  For all components, the dimensionality is limited to 5.

The following constraints hold only for the band-interleaved case:

- **IIF syntax entity *FundamentalImageStructure***

  The *band-record* component shall be selected.

- **IIF syntax entity *PixelStructure***

  The *elementary-pixel-structure* shall be selected.

The following constraints hold only for the pixel-interleaved case:

- **IIF syntax entity *FundamentalImageStructure***

  The *metric-array* component shall be selected.

- **IIF syntax entity *PixelStructure***

  The *compound-pixel-structure* component shall be selected.

6.2 Registered profiles for the IIF-DF

The profiles included in ISO/IEC 12087 are not expected to satisfy all future needs. New application profiles may be registered, when required, in the ISO Register of Graphical Items (see ISO/IEC 9973) by the International Registration Authority (RA) or via International Standardized Profiles (ISP).

Registered profiles for the IIF-DF are constructed by:

a) defining additional application-specific semantics to specific values of one or more IIF-DF syntax entities for which an extension of the semantics is allowed (see 6.2.1) and/or by

b) constraining the full profile of the IIF-DF or one of the other standardized profiles defined in 6.1. This means that in the case of choosing a registered profile in addition to a standardized profile, the constraints defined for both profiles hold true at the same time (see 6.2.2).

6.2.1 Application-specific semantics

Table 2 shows all IIF-DF syntax entities which allow for the registration of application-specific semantics.

syntax entity	component
Profile	---
ImageStructure	processing-history
DataPlacement	---
ElementaryPixelStructure	---
CompressedDataUnit	data-representation
RegisteredDataUnit	registration-id
ExternalReference	object-internal-id
ExternalAddress	cleartext-address
LookUpTable	interpolation-method
NeighbourhoodArray	semantic-label
MetricDescription	coordinate-system
MeasurementUnit	measurement
MetricTransformation	standard-transformation
Domain	domain-type
ChannelCharacteristics	channel-type
ChannelCharacteristics	channel-usage
ChannelCharacteristics	background-value
CompandorDescription	function-type
LinearRGBSpace	representation
LinearRGBSpace	illuminant
GammaRGBSpace	representation
GammaRGBSpace	illuminant
ColourBand	verbal-description
ImageAnnotation	numeric-identifier
ElementaryDataType	---

Table 2 - IIF-DF syntax entities which allow for registration

For components of type INTEGER, registered values shall be in the range of [1000..2000]. The constraints that are contained in the full profile to limit the value range of a certain syntax element to those values for which some semantics are predefined, will be changed if a new value is registered permitting this value to be set.

6.2.2 Constraining methods

The following methods are allowed to form constraints to the IIF-DF syntax:

a) A constraint may state that particular alternatives in a "CHOICE" syntax entity may no longer be permitted.

b) A constraint may state that an "OPTIONAL" component of a syntax entity may no longer be present.

c) A constraint may state that an "OPTIONAL" component of a syntax entity may now be mandatory.

d) A constraint may limit the number of elements within a "SEQUENCE OF" syntax entity.

e) A constraint may limit the allowed values for an elementary syntax entity.

f) A constraint may limit the hierarchical depths of recursive syntax entities.

g) A constraint may state that an "ANY" placeholder shall be replaced by a certain syntax entity (including a semantical description of the new entity).

NOTE - Constraints of category i) may, at first, seem to be extending the syntax. On the syntactical level, this is not true. In fact, this method constrains the syntax because some data streams that conform to the full IIF-DF syntax will no longer conform with the constrained syntax. Also, all data streams that conform with the constrained syntax also conform with the full IIF syntax. On the semantical level, it is true that the specification of the meaning of the new syntax entities that replace ANY entities might be regarded as an extension.

A list of all syntax entities which contain components of type ANY is given in Table 3.

syntax entity	component
ContentsHeader	application-data
ContentsElement	prolog
ContentsElement	epilog
RegisteredDataUnit	pixel-values
ImageRelatedData	usage
CoordinateAndFeature	feature
ImageAttribute	freeform
ChannelCharacteristics	sampling-function
ChannelCharacteristics	application-specifics
CompandorDescription	application-specific
ImageAnnotation	contents

Table 3 - IIF-DF syntax entities which contain components of type ANY

EXAMPLES

1 A constraint that belongs to category a) is: "It is not permitted to choose the *CompoundImageList* entity within the *CompoundImageStructure* entity."

2 A constraint that belongs to category e) is: "The value of the *number-of-dimensions* component which is contained in the *Dimensionality* entity shall be 2". Because of the constraints which are already contained in the definition of the full profile of the IIF-DF for the *Dimensionality* entity, this automatically restricts the number of *DimensionDescription* entities which are contained in the

sequence to 2.

3 A constraint that belongs to category f) is: "The number of *MetricArray* entities which are nested in a *FundamentalImageStructure* entity shall be 1."

4 A constraint that belongs to category g) could be the definition of the syntax of a patient folder that shall be used as the application-specific component of the *ImageAnnotation* entity within the medical imaging community.

6.3 Extension methods

For the IIF-DF, no other extension methods are supported than encompassed by the registration of new application profiles according to the rules specified in 6.2

7 IPI-IIF Gateway functionality

The functionality of the IPI-IIF Gateway is defined according to the interworking scenarios between the application, the IPI-PIKS, and the IPI-IIF Gateway. The IPI-IIF Gateway may be used in a stand-alone mode, as well as in combination with the IPI-PIKS. Both scenarios are described in 4.2.

7.1 Basic categories of IPI-IIF Gateway functions

The IPI-IIF Gateway provides the following categories of functionality:

a) Gateway control and error handling

b) Import and export functionality

c) Parse and generate functionality

d) Data structure access functionality

e) Data structure manipulation functionality

f) Compression and decompression functionality

g) Application-oriented functionality

These categories are described in the subsections below.

7.1.1 IPI-IIF Gateway control and error handling

This category encompasses all functions which are used to open, to close, and to reset the IPI-IIF Gateway, and to inquire about its status. The IIF Gateway is in one of the following three operational states:

{IIF_GATEWAY_CLOSED, IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR}

For each IIF Gateway function, a number of error situations are specified, any of which will cause the error state to be set and the error handling function to be called from the function in which the error was detected. Every IIF Gateway implementation supports this error checking.

The list of IIF Gateway function-caused errors is provided in Annex B. In case of any error which is explicitly defined in this list, the IIF Gateway calls the error handling mechanism. This mechanism provides an interface between the IIF Gateway and the application. The application may either provide its own error handling function or may use the one provided as part of the IIF Gateway.

Any error handling function accepts the following information from the IIF Gateway:

a) identification of the error condition

b) identification of the IIF syntax entity that caused the error

c) identification of the IIF element that called the error handling function

d) the error file identifier

The error handling function provided by the IIF Gateway calls the error logger function using the same set of parameters. The latter writes an error message on the error file. This message consists of the error code and the identifier of the IIF Gateway function in which the error was detected. This two-stage calling of error functions allows the application to supply its own error handling function while still having access to services provided by the error logging function.

The following error numbers are reserved:

a) Unused error numbers less than 2000 are reserved for future standardization

b) Error numbers 2000 to 3999 are reserved for language bindings

c) Error numbers greater than or equal to 4000 are reserved for registration in the ISO International Register of Graphical Items, which is maintained by the Registration Authority.

7.1.2 Import and export functionality

This category encompasses all functions that allow for the opening and closing of data ports and the transfer of image and image-related data from the IPI-PIKS to the IPI-IIF Gateway and vice versa.

The IPI-IIF Gateway does not standardize a specific low-level data exchange mechanism. Rather, each low-level data exchange mechanisms, as typically provided by operating systems, may be shared by the IPI-IIF Gateway.

The IPI-IIF Gateway employs a port concept to cover different low-level data exchange mechanisms. When the IPI-IIF Gateway is writing or reading data, a port identifier is assigned, from and/or to which data transfer takes place.

EXAMPLE - Examples of low-level data exchange mechanisms which are covered by the port concept are data exchange by writing to and reading from files, and data exchange based on pipes or sockets.

The IPI-IIF Gateway will typically operate three types of ports from and to

a) the application program,
b) the IPI-PIKS, and
c) the file system, or the transport interface of a network.

This situation is illustrated in Figure 5.

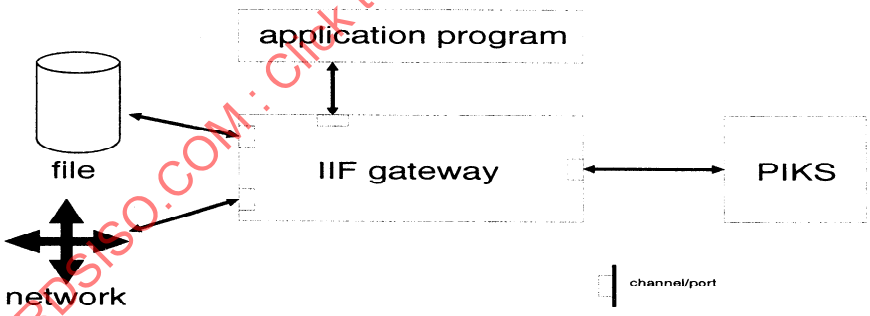

Figure 5 - Exemplary ports of the IPI-IIF Gateway

7.1.3 Parse and generate functionality

This category encompasses two functions which allow for the translation between IIF-DF data streams (according to the IIF data format) and IPI-IIF gateway-internal data structures which are accessible and manipulable via other IPI-IIF Gateway functions.

The read_and_parse_iif_image function reads an IIF-DF data stream, decodes the ASN.1 data entities and

creates an IPI-IIF gateway-internal data structure. The generate_and_write_iif_image function, in turn, arranges the data ordering and creates an encoded IIF-DF data stream from a data structure.

7.1.4 Data structure access functionality

This category encompasses all functions which allow for the access and manipulation of data structures. For the relation between data structures and IIF-DF data streams refer to 7.1.3.

From the viewpoint of the application programmer, a data structure is a tree-like representation of a parsed IIF-DF data stream. The nodes of the tree represent IIF syntax entities. The upper-most node (i.e., the root) of a data structure tree which has been created by parsing an IIF-DF data stream represents the *FullDataFormat* entity. The other nodes represent subentities, such as *FormatDescriptor*, *ContentsHeader*, *Contents*, *Image*, *ImageStructure*, *Dimensionality*, etc. According to the data structure manipulation functionality described in 7.1.5, data structures can be created that represent only a subtree of an entire IIF-DF-conformant image structure tree. Because every data structure represents an entity of the IIF-DF syntax, they are called *IIF syntax entity* within the IPI-IIF Gateway manual pages.

For a list of all IIF-DF syntax entities and their numerical identification numbers, refer to Annex A. Within the access and manipulation functions of the IPI-IIF Gateway, these numbers are used to identify the IIF syntax entities.

Note that the description of the tree-like data structure does not preclude any IPI-IIF gateway-internal data representation. Only the virtual representation in terms of access functionality is described by this part of ISO/IEC 12087.

As described in 5.1.2, every IIF syntax entity consists of a number of components. For example, the *BandRecord* entity consists of the *number-of-bands* component, the *data-placement* component, and the *record-components* component. The type of each component is defined by reference to another syntax entity which is either described elsewhere in the IIF-DF syntax or within the ASN.1 standard. For example, the *data-placement* component is represented by the *DataPlacement* entity, which is described within the IIF-DF syntax. The *number-of-bands* component is represented by the *INTEGER* entity, which is an elementary syntax entity. A list of all IIF syntax entity component names is provided in Annex A.

Note that IIF syntax entities may not be used as *globally unambiguous* identifiers within a data structure tree because one IIF syntax entity may occur at multiple locations within the same data structure tree. The meaning of the IIF syntax entity depends on the location in the tree. For example, the *ImageAttribute* entity allows for the description of image attributes (e.g., colour space). A data structure tree may contain multiple images with different colour spaces.

Furthermore, multiple components of one IIF syntax entity may be represented by the same IIF syntax entity. For example, the *title* and the *owner* component of the *ContentsHeader* entity are both represented by the elementary *IA5String* entity.

Hence, the access of data within a data structure tree that represents an IIF syntax entity is based on a sequential traverse from node to node. It works as follows:

a) All IIF syntax entities are handled by the application program via abstract identifiers (i.e., a handle or a pointer).

b) The type of an IIF syntax entity (i.e., a node of the data structure tree) can be inquired into by passing the identifier of this node to the inquiry function. The output is a numerical value. It describes the type of the IIF syntax entity which is represented by this node. The list of all valid IIF syntax entities and their numerical values is given in Annex A.

c) Every component of an IIF syntax entity can be accessed using the component number. This

number is constructed by appending the three digit syntax entity number with a two digit component count. A complete list of valid syntax entities, valid component names, and associated numbers is given in Annex A. The access function returns an identifier to the node that represents this component. This identifier does not point to a new object but to a subtree of the input object.

d) Some components are marked with the keyword *OPTIONAL*. Others are alternatives of a *CHOICE*. These components do not necessarily exist in the data structure tree. In these cases a NULL is returned and no error is generated.

e) Some components are defined as *SEQUENCE OF* a certain IIF syntax entity. In this case the component encompasses an arbitrary number of these IIF syntax entities. Hence, the access function requires another input parameter telling the index of the IIF syntax entity to be accessed. The entities are numbered starting from zero. This parameter is ignored in all other cases.

f) In order to provide faster access to the data within the tree, component names of subentities (and subsubentities, etc.) may also be used as input parameters. In the case of ambiguities due to multiple occurrences of these component names within a subtree, a NULL is returned and no error is generated.

g) For elementary IIF syntax entities and components that are represented as elementary entities (like, e.g., *INTEGER*, *IA5String*, *REAL*), get value and set value functions are provided.

7.1.5 Data structure manipulation functionality

As described in 7.1.4, IIF syntax entities are represented as tree-like data structures. For manipulations of these data structures, produce, delete, attach, and detach functions are provided. The manipulations work in the following way:

a) A node in the data structure tree may be copied or deleted. Both operations affect all subnodes.

b) A new node may be created. This function requires the numerical value identifying the IIF syntax entity that shall be represented by this node. No values or links to subentities are set by this function.

c) A created node can be filled with components by giving the name of the component and the identifier of the IIF syntax entity which shall represent the component.

NOTE - During the manipulation of data structures using the above-mentioned functions, the application programmer has to assure the semantical correctness of the data. The consistency of reference labels, band-identifiers, component-identifiers etc. is not maintained automatically.

Using the attach/detach functionality step by step, simpler IPI-IIF structures may be combined into more complex, and simpler structures may be extracted from more complex ones. Thus, an interface may be realized between the IPI-IIF data types (that are interchangeable by means of the IPI-IIF) and the IPI-PIKS data types (that can be processed by the IPI-PIKS, but also interchanged by the IPI-IIF). The IPI-IIF data types are oriented by the (generic) IPI data types that are introduced in clause 4 and form a superset of the IPI-PIKS data types. The IPI-IIF data types support compound and heterogeneous data structures of arbitrary hierarchical organization whereas the IPI-PIKS data types support five-dimensional images with limited heterogeneity.

7.1.6 Compression and decompression functionality

This category encompasses image data compression and decompression functions. The algorithms which are provided by the IPI-IIF Gateway are related to the encoded representations of compressed image data allowed within the IIF-DF. As described in 5.3.3, the following standards are referenced:

- CCITT Recommendation T.4
- CCITT Recommendation T.6
- ISO/IEC 11544 (JBIG)
- ISO/IEC 10918 (JPEG)
- ISO/IEC 11172 (MPEG-1)

These standards define compression algorithms which fall into several categories: lossless vs. lossy, bi-level vs. multiple-bits-per-pixel, still images vs. time sequences, data arrangement for sequential vs. progressive build-up. The standards have multiple applications and are very versatile. Therefore, implementations can support different levels of performance and sets of features and still be a conforming implementation of the standard. The conformance levels are indicated by constraints placed on the free parameters of each individual standard.

NOTES

1 Group 3 and Group 4 facsimile are lossless bi-level (or binary) compression standards. They work according to one- and two-dimensional entropy coding schemes, respectively, which were designed to perform best on a set of test images which have been scanned at approximately 200 dots per inch. They are sequential in nature because they compress and decompress images in a raster-scan order. The compression ratio achieved is image data-dependent.

2 JBIG is a lossless bi-level and limited-bits-per-pixel compression standard. On bi-level images scanned at 200 dots per inch, JBIG can be expected to achieve a 20 to 30% compression improvement over Group 4. JBIG can adapt to different statistical image characteristics. It has a sequential mode of operation and a progressive mode. In the latter mode, images of lower resolution are encoded first with a small number of bits and then the image resolution is progressively increased. The previously encoded lower resolutions are used as predictors for the higher resolutions. The JBIG algorithm can be an effective method of losslessly compressing images which contain from one to six bits-per-pixel.

3 The JPEG compression standard was designed for sequential and progressive coding of still colour images. It has both a lossless and a lossy mode. The latter mode is based on a discrete cosine transform (DCT) of 8 by 8 pixel blocks, followed by quantization and entropy encoding. In its lossless mode, it can code from one to sixteen bits-per-pixel. In the lossy mode, eight and twelve bits-per-pixel are allowed. JPEG is very effective at lossy compression of colour images at up to about 30:1 compression ratios and higher. JPEG's progressive mode of operation works either by ordering bit-planes according to their significance or by ordering DC coefficients according to their significance. Furthermore, a hierarchical mode of operation is provided. In this mode, the image is encoded as a pyramid of different resolutions.

4 The MPEG-1 compression standard defined in ISO/IEC 11172 was designed for synchronized video and audio compression at about 1.5 Mbits/sec. The compression ratio is about 100:1 with resulting video quality approximately equal to that of a VHS video cassette player. In contrast to JPEG, the MPEG-1 Standard defines a colour mode (YC_bC_r) and a fixed subsampling ratio among the three bands (4:1:1). Besides the DCT-based encoding of frames, the MPEG-1 algorithm performs motion compensation (based on prediction, interpolation and the use of motion vectors) to achieve the high compression ratio. Three types of pictures are used: intra-pictures, interpolated pictures, and predicted pictures. The encoding scheme is asymmetrical: the encoder is much more complex than the decoder. The MPEG-1 audio signal is compressed to a rate of about 192 or 384 Kbits/sec. The decompressed audio signal is a high fidelity stereo signal.

Within the IPI architecture, image data compression and decompression may take place at only one location: image data compression and decompression is performed by the respective components of the IPI-IIF Gateway. The data import and export processes from and to the IPI-PIKS do not perform image data compression and decompression. This situation is illustrated in Figure 6.

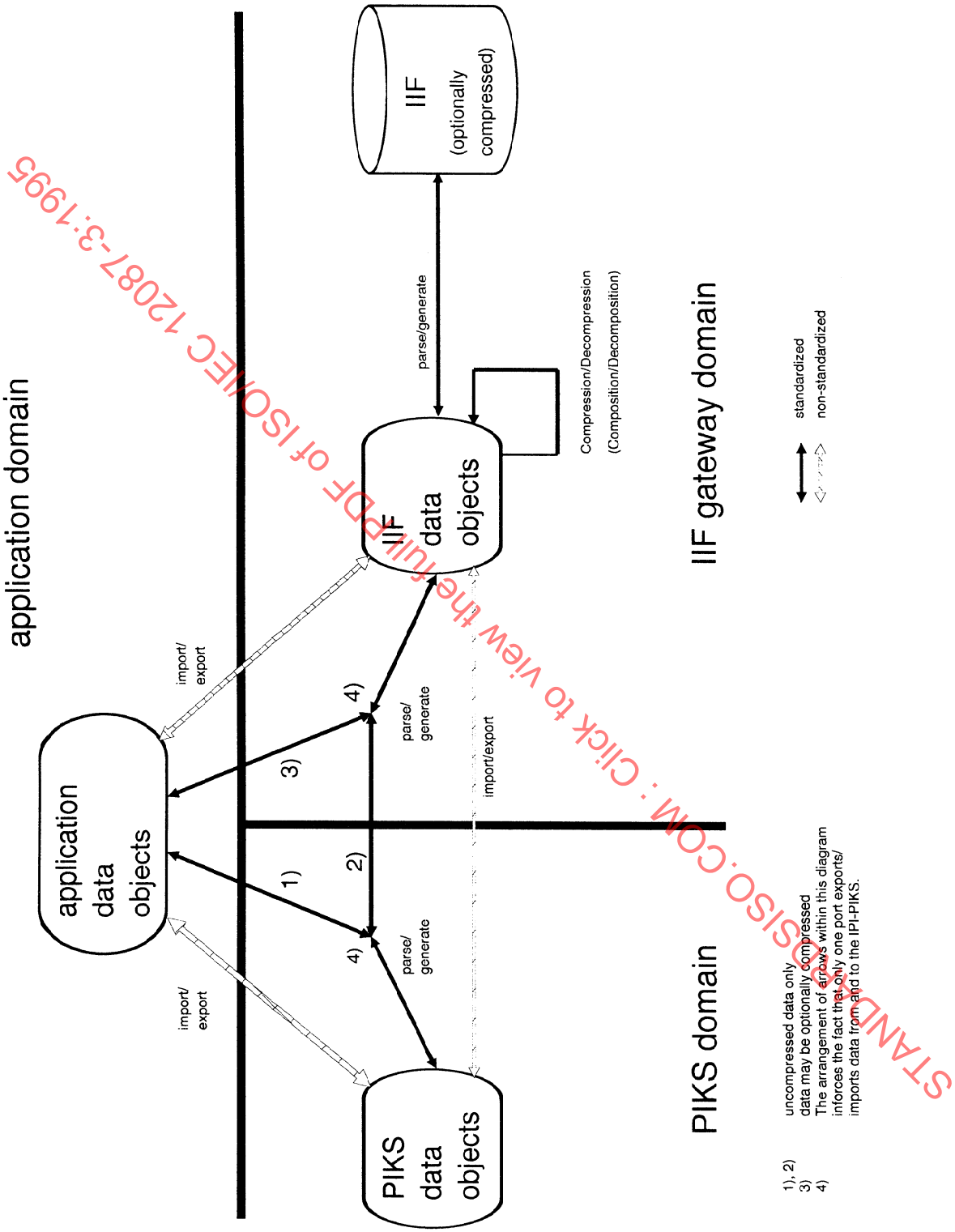

Figure 6 - Location of image compression and decompression processes

Figure 6 reinforces the fact that there is one data path in and out of the IPI-IIF Gateway domain and one path in and out of the IPI-PIKS domain. IPI-PIKS domain data objects may be picked up by the application and/or the IPI-IIF Gateway. The IPI-IIF Gateway domain objects may be picked up by the application and/or the PIKS domain. An implementation of the IPI-PIKS and the IPI-IIF Gateway supplied by a single vendor may have common representations of IPI-PIKS and IPI-IIF Gateway data objects and thus avoid some parse/generate operations.

7.1.7 Application-oriented functionality

This category encompasses all functions which perform multiple functional steps according to the categories of functionality defined in 7.1 at once. From the viewpoint of the application these functions are located on a higher (and thus more application-oriented) level.

7.2 IPI-IIF gateway-internal tables

There is one gateway-internal table to support the control of the IPI-IIF Gateway: the GATEWAY STATUS TABLE.

The first table entry is set to the current IIF status which can be one of three alternatives as defined in 7.1.1. Then the table contains open port entries for every opened port currently under control of the IPI-IIF Gateway. These entries consist of the following fields:

- Port identifier: This is the identifier which has been assigned to the port by the open_port function.
- I/O direction: This parameter specifies the I/O direction of the opened port. The following options are specified:

  { READ, WRITE, READ_AND_WRITE }

- Access type: This parameter specifies whether the port shall support random access (such as a file) or sequential access (such as a socket). The following options are specified:

  { RANDOM, SEQUENTIAL }

- Domain: This parameter specifies the communication domain in which the I/O operation shall take place. The following options are specified:

  { PIKS, NON_PIKS }

  Choosing the *PIKS* option supports optimized data transfer between IPI-PIKS and the IPI-IIF Gateway by allowing the GenerateAndWriteIIFImage and InputObject function to transfer data using an internal representation.

- I/O status: This parameter specifies whether there is any pending I/O operation on this port. The following options are specified:

  { CLEAR, PENDING }

IPI-IIF Gateway functions are provided to write, read, and reset this table. See 7.4 for further details.

7.3 Survey of IPI-IIF Gateway functions

A list of all IPI-IIF Gateway functions and the respective identification numbers is given in Table 4.

Gateway control functionality	Open IIF Gateway	001
	Close IIF Gateway	002
	Reset IIF Gateway	003
	Inquire IIF Gateway Status	004
	Inquire Supported Functionality	005
	Error Handler	006
	Error Logger	007
	Error Test	008
Import and export functionality	Open Port	101
	Close Port	102
	Inquire Port	103
	Reset Port	104
	Export IIF Data to IPI-PIKS	105
	Import IIF Data from IPI-PIKS	106
Parse and generate functionality	Read and Parse IIF Image	201
	Generate and Write IIF Image	202
Data structure access functionality	Inquire IIF Syntax Entity Choice	301
	Inquire IIF Syntax Entity Optional	302
	Inquire IIF Syntax Entity Sequence	303
	Inquire IIF Syntax Entity Number	304
	Inquire IIF Syntax Entity Validity	305
	Inquire IIF Syntax Entity Type Definition	306
	Inquire IIF Syntax Component Type Definition	307
	Get IIF Syntax Entity Component	308
	Get IIF Syntax Entity Value	309
	Put IIF Syntax Entity Value	310
Data structure manipulation functionality	Create IIF Syntax Entity	401
	Copy IIF Syntax Entity	402
	Delete IIF Syntax Entity	403
	Attach IIF Syntax Entity	404
	Detach IIF Syntax Entity	405
	Release IIF Syntax Entity Identifier	406
	Insert IIF Atomic Sequence Element	407
	Delete IIF Atomic Sequence Element	408
Compress and decompress functionality	Compress IIF Syntax Entity	501
	Decompress IIF Syntax Entity	502
Application-oriented functionality	Get Image Subrange	601
	Put Image Subrange	602

Table 4 - Survey of IIF Gateway functions

7.4 IPI-IIF Gateway functionality by manual pages

The specification of the IPI-IIF Gateway functions is given in the following manual pages. The functions are given in alphabetical order.

For the data types of input and output parameters, the same naming conventions are used as within the IPI-PIKS, as shown in Table 5.

abbreviation	semantics
BP	Boolean
NP	non-negative integer
SP	signed integer
RP	real
CP	complex
CS	character string
IP	identifier
EP	enumerated
null	NULL

Table 5 - Data type naming conventions for the IIF Gateway

For the input and output of elementary syntax entity values, the following mapping from ASN.1 entities to function parameter types applies:

ASN.1 entities	function parameters
BOOLEAN	BP
INTEGER	SP
REAL	RP
BIT STRING	IP
OCTET STRING	IP
IA5String	CS
GeneralizedTime	CS
OBJECT IDENTIFIER	CS

Table 6 - Mapping between ASN.1 entities and function parameter types

Every manual page consists of the following parts:

- **Description:**

  This contains a verbal description of the function. For an overview, refer to 7.1.

- **Name:**

  This is the formal name of the function.

- **Function class:**

  This is the name of the function class, e.g., DATA STRUCTURE ACCESS (4)

- **Parameters:**

  Here a list of input and output parameters appears. The types are given in the notation mentioned above.

- **Status:**

  This is the required IPI-IIF Gateway status.

- **Error Codes:**

  All potential errors are listed together with the respective error codes. For an overview of all IPI-IIF Gateway errors, refer to Annex B.

IPI-IIF GATEWAY FUNCTION No. 404 **ATTACH IIF SYNTAX ENTITY**

Description:

An IIF syntax entity is attached as a component to another IIF syntax entity. The entity type of the former IIF syntax entity needs to conform to the component type of the latter syntax entity as defined in the IIF-DF syntax.

The place within the parent entity to which the new entity should be attached, is identified by the "component number" parameter. For syntax entities that consist of only one (unnamed) component, this parameter is ignored.

For components that are organized as *SEQUENCE OF* a certain syntax entity, the "component index" parameter is used to give the index of the syntax entity to be associated with the component. This parameter is ignored in all other cases. The entities are numbered starting from zero.

Entities are restricted to be attached to only one parent entity at a time. Once an entity has been attached, it can still be operated on. Before deleting an entity, it needs to be detached.

> EXAMPLE - Given a *Dimensionality* entity which has been "filled up" with two *DimensionDescription* entities, one for the x and the other for the y axis. In order to attach a third *DimensionDescription* entity (describing the z axis) to the *Dimensionality* entity, the attach_iif_syntax_entity function needs to be called with four input parameters: the identifier of the *Dimensionality* entity, the identifier of the new *DimensionDescription* entity, the number of the component (in this case: *10 501*), and the index "2".

The attach_iif_syntax_entity function provides a general mechanism for the composition of compound IIF syntax entities. For an overview of the access and manipulation functionality provided by the IPI-IIF Gateway, refer to 7.1.4 and 7.1.5.

> NOTE - During the manipulation of data structures using this function, the application programmer has to take care of the semantical correctness of the data. The consistency of reference labels, band-identifiers, component-identifiers etc. is not maintained automatically.

Name:

attach_iif_syntax_entity

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity identifier	IP	in/out
Subentity identifier	IP	in
Component number	SP	in
Component index	NP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

815
Cause: Component name is ambiguous at this level of IIF syntax entity
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

821
Cause: Subentity already attached to another syntax entity
Reaction: Abort operation and issue warning

825
Cause: IIF syntax entity already has a subentity attached to the specified component, or (in case of a CHOICE construct) to another member of this CHOICE.
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 002 **CLOSE IIF GATEWAY**

Description:

This function closes the IPI-IIF Gateway. The IIF GATEWAY STATUS TABLE is set to "IIF_gateway_closed." All IPI-IIF gateway-internal data structures are deleted and all tables are cleared.

Name:

close_iif_gateway

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

None

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 102 **CLOSE PORT**

Description:

This function closes a port for reading and/or writing IIF image data. The input parameter is a port identifier. The respective entry is deleted from the GATEWAY STATUS TABLE.

Name:

close_port

Function class:

IMPORT/EXPORT (2)

Parameters:

Port identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 501 **COMPRESS IIF SYNTAX ENTITY**

Description:

This function performs the compression of an uncompressed field of pixel values given by the "uncompressed pixel field identifier" parameter. This parameter may be represented by one of the IIF syntax entities for uncompressed pixel values (i.e., *BuiltinEncodedDataUnit*, *ExternallyDefinedDataUnit*, and *RegisteredDataUnit*).

The type and the ordering of the source data is described by the "image structure entity identifier" parameter. This parameter represents an *ImageStructure* entity which describes the structure of the given field of pixel values.

NOTE - To avoid ambiguities, this *ImageStructure* entity may not comprise descriptions of other pixel fields not included in the given field of pixel values. Hence, it may not contain a subentity of type *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, *CompoundImageSet*, *BandRecord*, or *MetricArray* for which the *data-placement* component has been assigned to *distributed* (1).

The output of the function is an identifier to a *CompressedDataUnit* entity, containing the compressed pixel values.

The compression mode is determined by the basic compression algorithm (Facsimile, JBIG, JPEG, and MPEG-1) and a list of parameters which differs for each of the basic algorithms. This list only determines the compression mode but not the source image characteristics.

NOTE - Information such as "number of bands" and "array size" for every band is contained in the image structure entity. Thus, it is not necessary to have these source image attributes as extra parameters for the compression function.

For the constraints that apply for each compression parameter with respect to the source image format, refer to the respective standard documents. For an overview of the features and application areas of each compression algorithm, refer to 7.1.6.

The list of compression parameters contains the following items:

a) For the Group 3 facsimile compression, the list of compression parameters contains:

- compression mode	SP	*uncompressed*	(1)
		one-dimensional	(2)
		two-dimensional	(3)

b) For the Group 4 facsimile compression, the list of compression parameters is empty. The encoding always conforms to class 1, the facsimile encoding. Teletex and mixed mode are not supported.

c) For the JBIG compression, the list contains:

- number of bit-planes	NP
- progressive data arrangement flag	BP
- initial layer to be transmitted	NP
- final layer to be transmitted	NP
- initial layer template flag	BP
- transmission order high to low flag	BP
- number of lines per stripe	NP
- number of stripes	NP
- maximum horizontal offset for Adaptive Template	NP
- maximum vertical offset for Adaptive Template	NP

e) For the MPEG-1 compression, the list contains:

- intraframe interval NP
- predicted frame interval NP
- picture rate NP

d) For the JPEG compression, applied to an image which consists of n bands, the list contains the following input parameters:

- hierarchical mode flag BP
- if hierarchical mode:
  - number of frames NP
- if hierarchical mode: for each frame:
  - differential frame BP
  - number of lines in frame NP
  - number of samples per line NP
  - list of components in frame n * NP
- for each frame (only one in non-hierarchical mode):
  - number of scans NP
  - for each scan:
    - list of components in scan n * NP
  - arithmetic entropy coding flag: BP
  - lossless mode flag BP
  - if lossless mode:
    - for each scan:
      - predictor NP
      - point transform value NP
  - if lossy mode:
    - extended mode flag BP
    - if extended mode:
      - progressive mode flag BP
    - for each component:
      - quantization table selector NP
    - for each scan:
      - for each component in scan:
        DC entropy table selector NP
        AC entropy table selector NP
      - if progressive mode:
        - start of spectral selection NP
        - end of spectral selection NP
        - successive approximation bit high NP
        - successive approximation bit low NP
  - for each scan optionally (if value has changed):
    - if lossy mode (repeat 0 to 4 times):
      - quantization table element precision NP
      - quantization table destination identifier NP
      - quantization table elements 64 * NP
    - if huffman entropy coding mode (repeat 0 to 4 times):
      - huffman table destination identifier NP
      - DC or lossless huffman table (16 + 16) * NP
    - if arithmetic entropy coding mode (repeat 0 to 4 times):
      - arithmetic coding table destination identifier NP

- DC or lossless conditioning table value NP
- if lossy mode and huffman entropy coding mode (repeat 0 to 4 times):
  - huffman table destination identifier NP
  - AC huffman table (16 + 256) * NP
- if lossy mode and arithmetic entropy coding mode (repeat 0 to 4 times):
  - arithmetic coding table destination identifier NP
  - AC conditioning table value NP
- restart interval segment length NP

Name:

compress_iif_syntax_entity

Function class:

COMPRESS/DECOMPRESS (6)

Parameters:

Uncompressed pixel field identifier	IP	in
Image structure entity identifier	IP	in
Compression algorithm	SP	in
List of compression parameters	IP	in
Compressed data unit entity identifier	IP	out

The allowed values of the compression algorithm parameter are given by the *data-representation* component within the *CompressedDataUnit* entity. Refer to 5.3.

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

817
Cause: Violation of data compression constraints
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 402 **COPY IIF SYNTAX ENTITY**

Description:

This function copies the data structure of an IIF syntax entity. The identifier to the new entity is returned. This affects all subentities.

This function may also be used to copy substructures of a data structure representing an IIF syntax entity. Therefore, the get_iif_syntax_entity_component function needs to be applied first, in order to get the identifier of the substructure.

Name:

copy_iif_syntax_entity

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity identifier	IP	in
New IIF syntax entity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 401 **CREATE IIF SYNTAX ENTITY**

Description:

This function creates a data structure that represents an IIF syntax entity. The number of the IIF syntax entity is given by the "IIF syntax entity number" parameter.

The function may be used to create syntax entities:

- elementary ASN.1 entities which are used within the IIF syntax (e.g., *INTEGER*, *GeneralizedTime*, etc.),
- elementary IIF syntax entities (e.g., *Identifier*, *Serialization*, etc.), and
- compound IIF syntax entities (e.g., *FundamentalImageStructure*, *Contents*, *FullDataFormat*, etc.).

The list of all valid IIF syntax entities and the respective numerical identification values is given in 5.3.

NOTE - This list also assigns numerical identifiers to the elementary ASN.1 entities (like *INTEGER*, *IA5String*, etc.) which are used within the IIF syntax.

Compound IIF syntax entities that have been created with the create_iif_syntax_entity function do not contain any subentities. Therefore, the attach_iif_syntax_entity function needs to be applied.

For a general description of the data structure access and manipulation functionality provided by the IPI-IIF Gateway, refer to 7.1.4 and 7.1.5.

Name:

create_iif_syntax_entity

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity number	SP	in
IIF syntax entity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 502 DECOMPRESS IIF SYNTAX ENTITY

Description:

This function decompresses a field of pixel values which is given by the "compressed data unit entity identifier" parameter. This parameter represents a *CompressedDataUnit* entity.

NOTE - All parameters relevant for the decompression process are included in the compressed data field. Thus, the only parameter necessary for the decompression function is the entity that describes the structure and data ordering of the image data after being decompressed.

The structure and data ordering of the image data is passed to the function by the "image structure entity identifier" parameter. It represents an *ImageStructure* entity.

NOTE - To avoid ambiguities, this *ImageStructure* entity may not comprise descriptions of other pixel fields not included in the given field of pixel values. Hence, it may not contain a subentity of type *CompoundImageArray*, *CompoundImageRecord*, *CompoundImageList*, *CompoundImageSet*, *BandRecord*, or *MetricArray* for which the *data-placement* component has been assigned to *distributed* (1).

The function returns the "uncompressed pixel field identifier" parameter. This parameter represents the uncompressed field of pixel values by one of the IIF syntax entities *BuiltinEncodedDataUnit*, *ExternallyDefinedDataUnit*, and *RegisteredDataUnit*. The order and types of data within these output entities conform to the *ImageStructure* entity.

Name:

decompress_iif_syntax_entity

Function class:

COMPRESS/DECOMPRESS (6)

Parameters:

Compressed data unit entity identifier	IP	in
Image structure entity identifier	IP	in
Uncompressed pixel field identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 408 DELETE IIF ATOMIC SEQUENCE ELEMENT

Description:

This function deletes an element from a component that is a SEQUENCE OF atomic types (such as INTEGER, etc.). The component is identified by the "component number" parameter. The "component index" parameter gives the index of the element to be deleted in the component. The sequence elements are numbered starting from zero. Calling this function for a component that is not a SEQUENCE OF an atomic type results in an error.

Name:

delete_iif_atomic_sequence_element

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

Entity	IP	in/out
Component number	SP	in
Component index	NP	in

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

824
Cause: Component not atomic type
Reaction: Abort operation and issue warning

826
Cause: IIF syntax entity is invalid
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 403 **DELETE IIF SYNTAX ENTITY**

Description:

This function deletes an IIF syntax entity and all its subentities. Entities which are attached to a parent entity need to be detached before deleting.

NOTE - During the manipulation of data structures using this function, the application programmer has to take care of the semantical correctness of the data. The consistency of reference labels, band-identifiers, component-identifiers etc. is not maintained automatically.

Name:

delete_iif_syntax_entity

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

822
Cause: IIF syntax entity still attached to another syntax entity
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 405 **DETACH IIF SYNTAX ENTITY**

Description:

This function allows one to decompose a compound IIF syntax entity. The syntax entity is identified by the first parameter. The "component number" parameter identifies the component that shall be detached from the IIF syntax entity. The latter parameter is ignored, if the parent entity consists of only one (unnamed) component.

The "component index" parameter is used in the case of components that are defined as *SEQUENCE OF* a certain syntax entity. It gives the index of the entity within the sequence that shall be detached. This parameter is ignored in all other cases. The entities are numbered starting from zero.

The function returns an identifier to the IIF syntax entity that represents the component which has been detached. This data structure is not copied from the input data structure but taken away from it.

EXAMPLE - A *CompoundImageRecord* entity can be decomposed into its record components by applying this function for every component of this entity (called *record-components*). Because this component is declared as a *SEQUENCE OF RecordComponent* entities, the function needs to be applied for each entity that is contained in this sequence by specifying its input value.

For a general description of the data structure access and manipulation functionality provided by the IPI-IIF Gateway, refer to 7.1.4 and 7.1.5.

NOTE - During the manipulation of data structures using this function, the application programmer has to assure the semantical correctness of the data. The consistency of reference labels, band-identifiers, component-identifiers etc. is not maintained automatically.

Name:

detach_iif_syntax_entity

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity identifier	IP	in/out
Component number	SP	in
Component index	NP	in
Detached IIF syntax entity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

815
Cause: Component name is ambiguous at this level of IIF syntax entity
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 006 **ERROR HANDLER**

Description:

This function performs the handling of errors that occurred during the execution of IPI-IIF Gateway functions. Upon entry, it sets the IIF Gateway state *IIF_GATEWAY_ERROR*. In this state, only the following IIF Gateway functions may be invoked:

- inquiry functions
- error_logger
- close_iif_gateway

The function then calls the error_logger to provide information about the error condition. The same parameter values input to the error_handler are passed through to the error_logger.

The function finally resets the state to *IIF_GATEWAY_OPEN* and performs a predeterminated action, which depends on the severity of the particular error condition.

As described in 7.1.1, an application may replace the error_handler function with its own special error mechanism. However, it must support the defined error handling concepts to maintain compliance with the standard.

Name:

error_handler

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

error number	SP	in
IIF syntax entity identifier	IP	in
error causing function	SP	in
error file identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 007 **ERROR LOGGER**

Description:

This function logs the appropriate error message on the designated error logging device, indicated by the error file identifier input parameter. The error-causing function is identified by its function number.

Note that this function is called by the IIF Gateway ERROR HANDLER function. An implementation may replace the ERROR LOGGER function with its own special error logger. However, it must support the defined error handling concepts to maintain compliance with this standard.

Name:

error_logger

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

error number	SP	in
IIF syntax entity identifier	IP	in
error causing function	SP	in
error file identifier	IP	in

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 008 **ERROR TEST**

Description:

This function tests for any occurrence of a detected error condition since the last ERROR TEST was executed. The output error flag is set *FALSE* if there is no error. It is set *TRUE* if there is an error.

Name:

error_test

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

error flag	BP	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 105 **EXPORT IIF DATA TO IPI-PIKS**

Description:

This function exports IIF data that are associated with a port to the IPI-PIKS. It needs to be called by the application program according to the interworking protocol specification, as described in 7.5.

Name:

export_iif_data_to_piks

Function class:

IMPORT/EXPORT (2)

Parameters:

Port identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

804
Cause: Identified port has not been opened for writing
Reaction: Abort operation and issue warning

806
Cause: Export to IPI-PIKS failed
Reaction: Abort operation and issue warning

807
Cause: I/O Error
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 202 GENERATE AND WRITE IIF IMAGE

Description:

This function generates an IIF-DF-conforming data stream (containing compressed or non-compressed image data) from an IIF gateway-internal data structure that represents a full IIF syntax tree, given by the *FullDataFormat* entity and all its subentities. The generation process includes the ordering of data according to the IIF-DF syntax, and the encoding of ASN.1 entities according to the guidelines described in ISO/IEC 12089.

The input parameters "IIF syntax entity identifier" and "port identifier" identify the image data structure and the port to which the data are being written. The "profile" parameter is used to determine the profile to which the generated IIF data stream has to conform. Refer to the *Profile* syntax entity in 5.3 for the allowed string values.

Note that the data structure manipulation functions create_iif_syntax_entity, delete_iif_syntax_entity, attach_iif_syntax_entity, and detach_iif_syntax_entity don't check the completeness of subentities and possible violations of semantical constraints. Hence, the generate_and_write_iif_image function needs to prove whether the input data structure represents a syntactical and semantical correct *FullDataFormat* entity.

Name:

generate_and_write_iif_image

Function class:

PARSE/GENERATE (3)

Parameters:

IIF syntax entity identifier	IP	in
Port identifier	IP	in
Profile	CS	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

804
Cause: Identified port has not been opened for writing
Reaction: Abort operation and issue warning

807
Cause: I/O Error
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

813
Cause: Violation of semantical constraints within the IIF syntax entity
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 308 **GET IIF SYNTAX ENTITY COMPONENT**

Description:

This function is used to traverse an IIF syntax entity by moving from its root to one of its components, represented as nodes in the tree-like structure. The input parameters are "IIF syntax entity identifier", "component number", and "component index". If the input syntax entity consists of only one (unnamed) component, the "component number" parameter will be ignored.

The list of valid component numbers depends on the IIF syntax entity. A complete list is given in Annex A for every IIF syntax entity. The function returns an identifier to the node that represents this component in the parameter "subentity identifier".

Some components are defined as *SEQUENCE OF* a certain IIF syntax entity. In this case the component encompasses an arbitrary number of these IIF syntax entities. Hence, the "component index" parameter is used as index of the IIF syntax entity that is to be reached. In all other cases, this parameter will be ignored. The entities are numbered starting from zero.

> EXAMPLE - In order to gain access to the *i*th band of a multiband image, represented as *BandRecord* entity, the get_iif_syntax_entity_component function needs to be applied using the following input parameters: the identifier of the *BandRecord* entity, the component name *record-components*, and the index value *i*.

Some components are marked with the keyword *OPTIONAL*. Others are one of multiple alternatives of *CHOICEs*. These components do not necessarily exist in the data structure tree. In these cases a NULL is returned and no error is generated.

In order to provide a faster access to the data within the tree, component numbers of subentities (and subsubentities, etc.) may also be used as input parameters. In the case of ambiguities due to multiple occurrences of these components within a subtree, a NULL is returned and no error is generated. The access to a component is considered to be "ambiguous" if there are two or more paths of minimal length in the current syntax entity from the root of this entity to the sub-entity in question. Whenever a matching component is found in one branch, no further search is done in that branch.

> EXAMPLE - Given an image that consists of one band, represented as *MetricArray* entity. It is possible to ask for the dimensionality of this image using the get_iif_syntax_entity_component function with the numeric identifier of the *number-of-dimensions* component because this component name occurs only once within the subentities of the *MetricArray* entity. For multiband images, one has to move to each individual component before inquiring about the dimensionality.

For further description of the data structure access and manipulation functions and the nature of the tree-like data structures that are processed by IPI-IIF Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

Name:

get_iif_syntax_entity_component

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Component number	SP	in
Component index	NP	in
Subentity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

815
Cause: Component name is ambiguous at this level of IIF syntax entity
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 309 **GET IIF SYNTAX ENTITY VALUE**

Description:

This function returns the value of an elementary IIF syntax entity or an elementary component of an IIF syntax entity. For the latter case, the "component number" parameters is used to specify the component number. In the case of a component that is declared as a *SEQUENCE OF* elementary entities, the "component index" parameter is used to specify the index value. This parameter is ignored in all other cases. The entities are numbered starting from zero.

The type of output parameter depends on the type of the elementary syntax entity. Refer to 7.4.

For a general description of the data structure access and manipulation functionality provided by the IPI-IIF Gateway, refer to 7.1.4 and 7.1.5. Refer also to the description of the put_iif_syntax_entity_value function and the get_iif_syntax_entity_component function.

Name:

get_iif_syntax_entity_value

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Component number	SP	in
Component index	NP	in
Value	{BP, SP, RP, CS, IP}	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

815
Cause: Component name is ambiguous at this level of IIF syntax entity
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 601 **GET IMAGE SUBRANGE**

Description:

This function creates an image which consists of a rectangular subset of the pixel data of another image. The latter image is identified by the input parameter "image entity identifier". The function returns the new image by the parameter "subimage entity identifier". Both parameters represent IIF syntax entities of type *Image*. The rectangle is specified by an interval in each of the array dimensions.

If the structure of the input image does not belong to the full PIKS profile, the function may fail returning an error message.

Name:

get_image_subrange

Function Class:

APPLICATION-ORIENTED (7)

Parameters:

Image entity identifier	IP	in
xmin	NP	in
xmax	NP	in
ymin	NP	in
ymax	NP	in
zmin	NP	in
zmax	NP	in
tmin	NP	in
tmax	NP	in
bmin	NP	in
bmax	NP	in
Subimage entity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

818
Cause: Invalid index subrange
Reaction: Abort operation and issue warning

819
Cause: Invalid size or starting coordinate
Reaction: Abort operation and issue warning

820
Cause: Input image structure too complex
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 106 **IMPORT IIF DATA FROM IPI-PIKS**

Description:

This function imports IIF data from the IPI-PIKS. It needs to be called by the application program according to the interworking protocol specification, as described in 7.5.

Note that this function does not assign an I/O port. Therefore, the open_port function needs to be called. This function only establishes the contact to the IPI-PIKS. For reading data, the read_and_parse_iif_image function needs to be called.

Name:

import_iif_data_from_piks

Function class:

IMPORT/EXPORT (2)

Parameters:

Port identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

803
Cause: Identified port has not been opened for reading.
Reaction: Abort operation and issue warning

805
Cause: Import from IPI-PIKS failed
Reaction: Abort operation and issue warning

807
Cause: I/O Error
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 004 **INQUIRE IIF GATEWAY STATUS**

Description:

This function allows one to inquire about the status of the IIF Gateway.

Name:

inquire_iif_gateway_status

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

Status of IIF Gateway	SP	*IIF_GATEWAY_OPEN*	(1)	out
		IIF_GATEWAY_ERROR	(2)	
		IIF_GATEWAY_CLOSED	(3)	

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR, IIF_GATEWAY_CLOSED

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 307 **INQUIRE IIF SYNTAX COMPONENT TYPE DEFINITION**

Description:

This function returns information about the structure of one component of a syntax entity type definition. The syntax entity type is specified by its number, as defined in clause 5.3. The component is specified by its position within the definition; positions are numbered from the top down starting with zero.

The function returns the following information about the component:

- *Component type*: the number associated with the component's data type definition.
- *Optional flag*: a Boolean flag indicating whether the component is required or OPTIONAL.
- *Sequence flag*: a Boolean flag indicating whether the component was declared as a SEQUENCE OF type
- *Implicit flag*: a Boolean flag indicating whether the component was declared as IMPLICIT
- *Choice flag*: a Boolean flag indicating whether the component is a member of a CHOICE
- *Choice identifier*: if the component is a member of a CHOICE, this represents the numeric identifier of the CHOICE
- *Component tag*: the component's tag value.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_component_type_definition

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity type	SP		in
Component position	NP		in
Component type	SP		out
Optional flag	BP		out
Sequence flag	BP		out
Implicit flag	BP		out
Choice flag	BP		out
Choice identifier	SP		out
Component tag	NP		out
Validity	EP	*{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 301 **INQUIRE IIF SYNTAX ENTITY CHOICE**

Description:

This function inquires which alternative is present within a CHOICE construct of a given IIF syntax entity. If the CHOICE is a sub-component of a syntax entity that is defined as a SEQUENCE, the "CHOICE number" parameter is used to identify the CHOICE construct. This parameter is ignored if the syntax entity only consists of the CHOICE. The function returns the component number of the alternative present in the CHOICE construct.

The numerical identification values for all IIF syntax entities are defined in 5.3. The list of valid component numbers depends on the IIF syntax entity. A complete list is given in Annex A for every IIF syntax entity.

For further description of the functions that are provided for data structure access and manipulation, and for a description of the nature of the tree-like data structures that are processed by IPI-IIF-Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_choice

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP		in
CHOICE number	NP		in
Component number	SP		out
Validity	EP	*{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 304 INQUIRE IIF SYNTAX ENTITY NUMBER

Description:

This function inquires the syntax number of an IIF syntax entity. The input parameter is an identifier to a data structure, as created by the read_and_parse_iif_image function or by the create_iif_syntax_entity function. The syntax numbers for all IIF syntax entities are defined in 5.3.

For further description of the functions that are provided for data structure access and manipulation, and for a description of the nature of the tree-like data structures that are processed by IPI-IIF Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_number

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
IIF syntax entity type	SP	out
Validity	EP *{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 302 INQUIRE IIF SYNTAX ENTITY OPTIONAL

Description:

This function inquires whether a certain component which is defined to be optional by the IIF syntax is present within a given IIF syntax entity. The function returns a Boolean flag.

The numerical identification values for all IIF syntax entities are defined in 5.3. The list of valid component numbers depends on the IIF syntax entity. A complete list is given in Annex A for every IIF syntax entity.

For further description of the functions that are provided for data structure access and manipulation, and for a description of the nature of the tree-like data structures that are processed by IPI-IIF Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_optional

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Component number	SP	in
IIF syntax entity optional	BP	out
Validity	EP *{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 303 INQUIRE IIF SYNTAX ENTITY SEQUENCE

Description:

This function counts the subentities of a certain component which is defined as SEQUENCE OF by the IIF syntax. The respective component is specified by the "component number" parameter. In case of syntax entities that consist of only one (unnamed) component, this parameter is ignored. The counted value is returned in the parameter "length".

The numerical identification values for all IIF syntax entities are defined in 5.3. The list of valid component numbers depends on the IIF syntax entity. A complete list is given in Annex A for every IIF syntax entity.

For further description of the functions that are provided for data structure access and manipulation, and for a description of the nature of the tree-like data structures that are processed by IPI-IIF Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_sequence

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Component number	SP	in
Length	NP	out
Validity	EP *{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 306 INQUIRE IIF SYNTAX ENTITY TYPE DEFINITION

Description:

This function returns information about the structure of a syntax entity type definition. The syntax entity type is specified by its number, as defined in clause 5.3. The function returns the class of the definition by the output parameter "IIF syntax entity class". The classes *SET* (4) and *SET OF* (5) are not used in the current syntax. They are introduced for future use. The class *equivalence* indicates that a syntax entity is defined as being "equivalent" to anoter entity, e.g.,

```
Identifier ::= IA5String.
```

The function also returns the number of components of the given IIF syntax entity.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_type_definition

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity type number	SP			in
IIF syntax entity class	SP	*SEQUENCE*	(1)	out
		CHOICE	(2)	
		SEQUENCE OF	(3)	
		SET	(4)	
		SET OF	(5)	
		equivalence	(6)	
Number of components	NP			out
Validity	EP	*{DATA_VALID, DATA_INVALID}*		out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 305 INQUIRE IIF SYNTAX ENTITY VALIDITY

Description:

This function inquires a given IIF syntax entity with respect to the IIF syntax and the set of constraints defined in clause 5. The "profile" parameter may be used to further constrain the parser to a (pre-defined or officially registered) application profile. Refer to the *Profile* syntax entity in 5.3 for the allowed string values. Note that this function checks one level of hierarchy only.

The function checks whether the given IIF syntax entity identifier is valid or not and returns the result in the output parameter "syntax entity identifier valid". The function checks whether one or more components are missing and returns the component numbers of the missing ones. In case of CHOICES, it chooses a member of the CHOICE at random and returns its component number. It also returns a Boolean parameter which is *TRUE* if the given syntax entity violates one or more constraints.

NOTE - During the manipulation of data structures using the data structure manipulation functionality, data structures may be created that do not represent syntactically complete and/or semantically consistent IIF syntax entities.

The numerical identification values for all IIF syntax entities are defined in 5.3. The list of valid component names depends on the IIF syntax entity. A complete list is given in Annex A for every IIF syntax entity. For further description of the functions that are provided for data structure access and manipulation, and for a description of the nature of the tree-like data structures that are processed by IPI-IIF Gateway, refer to 7.1.3, 7.1.4 and 7.1.5.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_iif_syntax_entity_validity

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Profile	CS	in
Syntax entity identifier valid	BP	out
Missing components	List of SP	out
Constraint error	BP	out
Validity	EP *{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 103 **INQUIRE PORT**

Description:

This function returns the respective contents of the entry in the GATEWAY STATUS TABLE. For a description of port attributes refer to 7.2.

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_port

Function class:

IMPORT/EXPORT (2)

Parameters:

Port identifier	IP			in
I/O direction	SP	*READ*	(1)	out
		WRITE	(2)	
		READ_AND_WRITE	(3)	
Access type	SP	*RANDOM*	(1)	out
		SEQUENTIAL	(2)	
Domain	SP	*PIKS*	(1)	out
		NON_PIKS	(2)	
I/O status	SP	*CLEAR*	(1)	out
		PENDING	(2)	
Validity	EP	*{DATA_VALID, DATA_INVALID}*		out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 005 **INQUIRE SUPPORTED FUNCTIONALITY**

Description:

This function allows one to inquire whether a certain function is supported by an IPI-IIF Gateway implementation.

The input parameter is the identification number of the function. The output parameter "function supported" is set *TRUE* if the function is supported and set *FALSE* otherwise. A list of all IPI-IIF Gateway functions and the respective identification numbers are given in 7.3

As all other inquiry functions, this function cannot cause an error. Thus the output parameter "validity" is used to indicate whether the data returned by the inquiry is valid or not.

Name:

inquire_supported_functionality

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

Function number	SP		in
Function supported	BP		out
Validity	EP	*{DATA_VALID, DATA_INVALID}*	out

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

None.

IPI-IIF GATEWAY FUNCTION No. 407 INSERT IIF ATOMIC SEQUENCE ELEMENT

Description:

This function inserts a new element into a component that is a SEQUENCE OF atomic types (such as INTEGER, etc.). The component is identified by the "component number" parameter. The "component index" parameter is used to give the index of the element to be associated in the component. The sequence elements are numbered starting from zero. Calling this function for a component that is not a SEQUENCE OF an atomic type results in an error.

EXAMPLE - Given an *IndexND* entity which has been "filled up" with two INTEGER entities, one for the *x* and the other for the *y* axis. In order to attach a third INTEGER entity, the insert_iif_atomic_sequence_element function needs to be called with three input parameters: the identifier of the *IndexND* entity, the number of the component (in this case: 31 301), and the index "2".

Subsequent calls of put_iif_syntax_entity_value may be used to assign the sequence elements' value.

Name:

insert_iif_atomic_sequence_element

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

Entity	IP	in/out
Component number	SP	in
Component index	NP	in

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

824
Cause: Component not atomic type
Reaction: Abort operation and issue warning

826
Cause: IIF syntax entity is invalid
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 001 **OPEN IIF GATEWAY**

Description:

This function opens the IIF gateway. The IIF GATEWAY STATUS TABLE is set to "IIF_gateway_open." The identifier of an error file is passed to the IIF gateway. This file is used by the IIF gateway to log warnings or error messages.

Name:

open_iif_gateway

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

error file	IP	in

Status:

IIF_GATEWAY_CLOSED

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 101 **OPEN PORT**

Description:

This function opens a port for reading in and/or writing out IIF image data. The input parameters are an external identifier, the desired I/O direction, the access type (e.g., random or sequential), and the domain. The "external identifier" parameter is used as low level exchange identifier, as described in 7.1.2. The other port attributes are further described in 7.2. The characteristics of the opened port will be added to the GATEWAY STATUS TABLE. The port identifier is returned.

Name:

open_port

Function class:

IMPORT/EXPORT (2)

Parameters:

External identifier	IP			in
I/O direction	SP	*READ*	(1)	in
		WRITE	(2)	
		READ_AND_WRITE	(3)	
Access type	SP	*RANDOM*	(1)	in
		SEQUENTIAL	(2)	
Domain	SP	*PIKS*	(1)	in
		NON_PIKS	(2)	
Port identifier	IP			out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

802
Cause: Identified port cannot be opened
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 310 PUT IIF SYNTAX ENTITY VALUE

Description:

This function sets the value of an elementary IIF syntax entity or an elementary component of an IIF syntax entity. For the latter case, the "component number" parameter is used to specify the component number. If the input syntax entity consists of only one (unnamed) component, this parameter is ignored.

In the case of a component that is declared as a *SEQUENCE OF* elementary entities, the "component index" parameter is used to specify the index value. This parameter will be ignored in all other cases. The entities are numbered starting from zero.

The type of the input value parameter depends on the type of the elementary syntax entity. Refer to 7.4

For a general description of the data structure access and manipulation functionality provided by the IPI-IIF Gateway, refer to 7.1.4 and 7.1.5. Refer also to the description of the get_iif_syntax_entity_value function and the get_iif_syntax_entity_component function.

NOTE - During the manipulation of data structures using this function, the application programmer has to take care of the semantical correctness of the data. The consistency of reference labels, band-identifiers, component-identifiers etc. is not maintained automatically.

Name:

put_iif_syntax_entity_value

Function class:

DATA STRUCTURE ACCESS (4)

Parameters:

IIF syntax entity identifier	IP	in
Value	{BP, SP, RP, CS, IP}	in
Component number	SP	in
Component index	NP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

811
Cause: IIF syntax entity has a wrong type
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

814
Cause: Component name does not exist for the given IIF syntax entity type
Reaction: Abort operation and issue warning

815
Cause: Component name is ambiguous at this level of IIF syntax entity
Reaction: Abort operation and issue warning

816
Cause: Specified component has no subentity at this index
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 602 **PUT IMAGE SUBRANGE**

Description:

This function replaces a rectangular subset of the pixel data of a given image with the pixel values from another image, called subimage. The images are identified by the parameters "image entity identifier" and "subimage entity identifier". Both parameters represent IIF syntax entities of type *Image*.

If the structures of the input images do not meet the constraints of the full PIKS profile, the function may fail and return an error message.

The location of the replaced pixels within the first image is specified by a starting coordinate; the size of the rectangular section is equal to the size of the second image. If the second image does not "fit" completely into the first image, given the starting coordinate, an error will be returned.

Name:

put_image_subrange

Function Class:

APPLICATION-ORIENTED (7)

Parameters:

Image entity identifier	IP	in/out
xstart	NP	in
ystart	NP	in
zstart	NP	in
tstart	NP	in
bstart	NP	in
Subimage entity identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

810
Cause: IIF syntax entity identifier is invalid
Reaction: Abort operation and issue warning

812
Cause: IIF syntax entity has missing subentities
Reaction: Abort operation and issue warning

819
Cause: Invalid size or starting coordinate
Reaction: Abort operation and issue warning

820
Cause: Input image structure too complex
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 201 **READ AND PARSE IIF IMAGE**

Description:

This function reads an IIF data stream from a port identified by the parameter "port identifier", parses the ASN.1-encoded data, checks the IIF syntax and the constraints, and creates a tree of IIF syntax entities whose root is returned in the parameter "IIF syntax entity identifier".

The root of the tree represents the *FullDataFormat* entity. The nodes represent arbitrary subentities (such as *FundamentalImageStructure* or *Dimensionality*) as specified by the IIF syntax rules. The leaves of the tree are the elementary entities, such as *INTEGER* and *REAL*.

Syntactical and semantical errors that have been detected during the parsing are added to the error logging file.

The input parameters are the "port identifier", four flags, and the "profile" parameter.

If the flags are set *TRUE*, the corresponding parts of the incoming data stream shall be ignored. The input parameter "image data skipping option" refers to the *image-data* component within *Image* entities. The input parameters "image-related data skipping option", "image attribute skipping option", and "image annotation skipping option" refer to the *ImageRelatedData*, *ImageAttribute*, and *ImageAnnotation* entities, which may occur within the *ContentsBody* or the *ImageStructure* entity. To gain access to skipped parts, the read_and_parse_iif_image function needs to be called again.

The "profile" parameter may be used to constrain the parser to a certain application profile. If the IIF data stream does not conform to this profile, a syntactical error is generated. Refer to the *Profile* syntax entity in 5.3 for the allowed string values.

The output parameter is the identifier of the root of the data structure tree, called "IIF syntax entity identifier".

The tree-like structure which is created by this function can be traversed for data access by other IPI-IIF Gateway functions:

- The inquire_iif_syntax_entity_type function allows one to inquire whether a given entity is elementary or compound. This function returns the numerical identification value of the syntax entity, as defined in 5.3.
- The get_iif_syntax_entity_component function provides access to a component of this IIF syntax entity. This component is also represented as IIF syntax entity.

Note that the parsing process does not decompress compressed data fields.

NOTE - Independent from the skipping options, this function may also be implemented in such a way that it initially reads and parses only a limited portion of the image data and sets pointers to the skipped parts. During the execution of traverse and access functions, the skipped parts need to be read and parsed "on the fly." This depends on the characteristics of the port.

Name:

read_and_parse_iif_image

Function class:

PARSE/GENERATE (3)

Parameters:

Port identifier	IP	in
Image data skipping option	BP	in
Image-related data skipping option	BP	in
Image attribute skipping option	BP	in
Image annotation skipping option	BP	in
Profile	CS	in
IIF syntax entity identifier	IP	out

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

803
Cause: Identified port has not been opened for reading
Reaction: Abort operation and issue warning

807
Cause: I/O Error
Reaction: Abort operation and issue warning

808
Cause: Syntactical error within IIF data stream
Reaction: Abort operation and issue warning

809
Cause: Semantical error within IIF data stream
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 406 RELEASE IIF SYNTAX ENTITY IDENTIFIER

Description:

This function invalidates the identifier of a non-root IIF syntax entity without affecting the representation of the entity it references. The entity referenced by the identifier must be attached to a parent. The identifier of a root entity must not be released. Releasing the identifier allows a gateway implementation to deallocate any memory associated with the identifier without affecting the syntax entity it references. Attempting to reference a syntax entity through a previously released identifier will produce an error.

Name:

release_iif_syntax_entity_identifier

Function class:

DATA STRUCTURE MANIPULATION (5)

Parameters:

IIF syntax entity identifier IP in

Status:

IIF_GATEWAY_OPEN, IIF_GATEWAY_ERROR

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

823
Cause: IIF syntax entity not attached to a parent
Reaction: Abort operation and issue warning

826
Cause: IIF syntax entity is invalid
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 003 **RESET IIF GATEWAY**

Description:

This function resets the IIF Gateway. All operations are aborted. The entries in all tables are deleted. Image structures are deleted. All ports are closed. The IIF Gateway remains in the "IIF_gateway_open" status.

Name:

reset_iif_gateway

Function class:

IIF GATEWAY CONTROL (1)

Parameters:

None

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

IPI-IIF GATEWAY FUNCTION No. 104 **RESET PORT**

Description:

This function clears any pending I/O operation on the port. The respective entry in the GATEWAY STATUS TABLE is set.

Name:

reset_port

Function class:

IMPORT/EXPORT (2)

Parameters:

Port identifier	IP	in

Status:

IIF_GATEWAY_OPEN

Error Codes:

801
Cause: IIF Gateway is not in the proper status
Reaction: Abort operation and issue warning

803
Cause: Identified port has not been opened for reading
Reaction: Abort operation and issue warning

804
Cause: Identified port has not been opened for writing
Reaction: Abort operation and issue warning

7.5 PIKS-IIF interworking protocol

The PIKS-IIF interworking protocol consists of function call sequences which refer to both the IPI-PIKS and the IPI-IIF Gateway. It distinguishes between two cases. The first case deals with the transfer of image data from the IPI-IIF Gateway to the IPI-PIKS. The following nomenclature is used for the protocols:

a) Parameters that are used as object identifiers: *inPortID* and *outPortID* are IIF port identifiers, *imageID* and *newImageID* are IIF syntax entity identifiers, and *piksID* is an IPI-PIKS object identifier.

b) The *InquireAndManipulateIIFEntity* function represents a sequence of function calls (e.g., *GetIIFSyntaxEntityComponent*, *GetIIFSyntaxEntityValue*, etc.) which are applied to the *imageID* in order to create a new entity which conforms to the full PIKS profile.

c) The *CreatePIKSImageObject* function covers any PIKS-internal image creation process.

d) Functions marked with a"*" need to be synchronized between the IPI-IIF Gateway and the IPI-PIKS.

1.	**IIF Gateway function call:**	**IPI-PIKS function call:**
	OpenIIFGateway()	
	inPortID := OpenPort()	
	imageID := ReadAndParseIIFImage(inPortID)	
	newImageID := InquireAndManipulateIIFImageStructure(imageID)	
	outPortID := OpenPort()	
	GenerateAndWriteIIFImage(newImageID,outPortID)	
	ExportIIFDataToPIKS(outPortID)*	
		Input Object(outPortID)*

2.	**IPI-PIKS function call:**	**IIF Gateway function call:**
	piksID := CreatePIKSImageObject()	
		OpenIIFGateway()
		inPort := OpenPort()
	OutputObject(piksID,inPortID)*	
		ImportIIFDataFromPIKS(inPortID)*
		imageID := ReadAndParseIIFImage(inPortID)
		InquireAndManipulateIIFImageStructure()

NOTE - Although the two examples show the interchange of an image between IPI-PIKS and IPI-IIF, any PIKS object can be interchanged, using the same mechanism.

Annex A
(normative)
List of IIF-DF syntax entities and component names

This Annex provides a list of the names and identification numbers of all components that are defined for IIF syntax entities. The identification numbers are used as input parameters for the following IPI-IIF Gateway functions:

- get_iif_syntax_entity_component
- get_iif_syntax_entity_value
- put_iif_syntax_entity_value
- attach_iif_syntax_entity
- detach_iif_syntax_entity
- insert_iif_atomic_sequence_element
- delete_iif_atomic_sequence_element

Note that according to the IIF-DF syntax, described in 5.3, some of the component names are optional or represent one of multiple alternatives within a *CHOICE* construct. Hence, these components do not necessarily exist within a data structure that represents an IIF syntax entity.

The first column of Table 7 gives the IIF syntax entity numbers. The second column provides the respective syntax entity names. The third column of the table provides the component names, the fourth column the respective numbers. These numbers are constructed by appending the three digit syntax entity numbers with a two digit component count.

Note that the same component name may occur within multiple syntax entities, e.g., the component name *number-of-bands*. The numbering scheme assigns multiple numbers to the same name. However, within the IIF Gateway, all different numbers which have been assigned to one component name may be used synonymously as input parameters.

In the fifth column, elementary components are marked with an *"E"* and constructed components are marked with a *"C."* Components represented by *ANY* syntax entity are marked with an *"A."* As described in clause 7, the access and manipulation functions allow one to address subcomponents directly by the component names of subentities (and subsubentities, etc.). For these names, refer to the syntactical description of the IIF-DF, given in 5.3.

In the sixth column of Table 7 it is indicated whether a certain component is represented as *SEQUENCE OF* an IIF syntax entity or as a *CHOICE* of a list of IIF syntax entities. Both cases have implications for the access and inquiry functions, as described in 7.1.4:

- SEQUENCE OF:
  The access and inquiry functions require an additional indexing parameter.
- CHOICE:
  These component numbers may not be used for the function attach_iif_syntax_entity.

IIF syntax entity		Component names	No.	Type	
001	FullDataFormat	format-descriptor	00 100	C	
		contents-header	00 101	C	
		contents	00 102	C	
002	FormatDescriptor	self-identification	00 200	E	
		version	00 201	C	
		profile	00 202	C	
003	Version	standard	00 300	E	
		publication-date	00 301	E	
004	Profile	--	--	E	
005	ContentsHeader	title	00 500	C	
		owner	00 501	C	
		date-and-time	00 502	E	
		message	00 503	C	
		application-data	00 504	A	
006	CharacterString	standard-characters	00 600	E	
		special-characters	00 601	C	
007	SpecialCharacterString	character-set-escape	00 700	E	
		characters	00 701	E	
008	Contents	--	--	C	SEQ OF
009	ContentsElement	prolog	00 900	A	
		body	00 901	C	SEQ OF
		epilog	00 902	A	
010	ContentsBody	image	01 000	C	
		image-related-data	01 001	C	
		image-attribute	01 002	C	
		image-annotation	01 003	C	
		basic-data-object	01 004	C	

Table 7 - List of IIF-DF syntax entities and components