

Implementation of
ISO/IEC 12087-2:1994

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

Part 2: Programmer's imaging kernel
system application program interface

ICS 35.140.00

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

BSI
Documents

AVA

Document Sharing Hub

Committees responsible for this British Standard

The preparation of this British Standard was entrusted by the Information Systems Technology Standards Policy Committee (IST/-) to Technical Committee IST/31, upon which the following bodies were represented:

- Department of Trade and Industry [IT Standards Unit (It'd6a)]
- Department of Trade and Industry (National Engineering Laboratory)
- Information Systems Committee of the Universities Funding Council
- Rutherford Appleton Laboratory

This British Standard, having been prepared under the direction of the Information Systems Technology Standards Policy Committee, was published under the authority of the Standards Board and comes into effect on 15 April 1995

© BSI 05-2000

The following BSI references relate to the work on this standard:
Committee reference IST/31
Draft for comment 92/68727 DC

ISBN 0 580 23710 9

Amendments issued since publication

Amd. No.	Date	Comments

Contents

	Page
Committees responsible	Inside front cover
National foreword	ii
Foreword	v
Text of ISO/IEC 12087-2	1

National foreword

This British Standard reproduces verbatim ISO/IEC 12087-2:1994 and implements it as the UK national standard.

This British Standard is published under the direction of the Information Systems Technology Standards Policy Committee whose Technical Committee IST/31 has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international committee any enquiries on interpretation, or proposals for change, and keep UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

NOTE International and European Standards, as well as overseas standards, are available from Customer Services, BSI, 389 Chiswick High Road, London, W4 4AL.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i and ii, the ISO/IEC title page, pages ii to vi, pages 1 to 646 and a back cover.

This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

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

Part 2:

Programmer's imaging kernel system
application program interface

*Technologies de l'information — Infographie et traitement de l'image —
Traitement de l'image et échange (IPI) — Spécification fonctionnelle —
Partie 2: Interface de programme d'application PIKS*



Reference number
ISO/IEC 12087-2:1994(E)

Contents

	Page
Foreword	v
1 Scope	1
2 Normative references	2
3 Symbols and abbreviations	2
4 Programmer's Imaging Kernel System specification	6
4.1 PIKS imaging model	6
4.1.1 Image data objects	7
4.1.2 Non-image data objects	8
4.1.3 Data object creation	9
4.2 PIKS operators, tools, data object repository utilities, and system mechanisms	9
4.2.1 Operators	9
4.2.2 Tools	10
4.2.3 Data object repository	10
4.2.3.1 Impulse response function arrays	10
4.2.3.2 Dither arrays	11
4.2.3.3 Colour conversion matrices	11
4.2.4 Utilities	11
4.2.5 System mechanisms	11
4.3 PIKS operator model	19
4.3.1 Non-image to non-image operators	19
4.3.2 Image to non-image operators	19
4.3.3 Image to image operators	19
4.3.4 Neighbourhood operators	20
4.3.5 Operator index assignment	20
4.4 PIKS system mechanisms	22
4.4.1 Data object allocation	22
4.4.2 Match point control	23
4.4.3 ROI control	24
4.4.4 ROI data object creation and manipulation	26
4.4.5 Asynchronous control	27
4.4.6 Element chaining	29
4.4.6.1 Chain construction	29
4.4.6.2 Chain execution	30
4.4.7 Virtual register control	32
4.4.7.1 Virtual registers for storage of PIKS temporary variables	33
4.4.7.2 Virtual registers for asynchronous control	33
4.4.7.3 Virtual registers for chain iteration and conditional execution	34
4.4.7.4 Virtual registers for auditing asynchronous state	34
4.4.8 Global element control	34
4.4.9 Composite image management	35
4.4.9.1 Composite image identifier arrays	35
4.4.9.2 Composite image identifier lists	36
4.4.9.3 Composite image identifier records	36
4.4.10 PIKS error handling	37
4.4.11 PIKS operational states	38
4.5 PIKS utilities	40
4.5.1 Inquiry	40
4.5.2 Import and export	40
4.5.2.1 Data object import and export utilities	41

	Page
4.5.2.2 PIKS to application data type conversion	41
5 PIKS conformance and extension methods	43
5.1 Foundation profile	43
5.2 Application profiles	43
5.3 Full profile	43
5.4 IIF capability profiles	43
5.5 PIKS profile definitions	43
5.6 Extension methods	53
6 PIKS element specification template	53
7 PIKS element specifications	59
Annex A (normative) Definitions of mathematical functions	539
A.1 Conventional mathematical symbols	539
A.2 Operational symbols	539
A.3 Mathematical functions	540
Annex B (informative) PIKS element support — source image structure	544
Annex C (informative) PIKS element support — destination image structure	554
Annex D (informative) PIKS element support — source and destination image data type	490
Annex E (informative) PIKS operator support — operator order	500
Annex F (informative) PIKS element functionality	505
Annex G (normative) PIKS data object repository	514
G.1 Impulse response function arrays	514
G.2 Dither arrays	525
G.3 Colour conversion matrices	526
Annex H (normative) PIKS image resampling	538
Annex J (normative) PIKS error codes	544
Annex K (informative) Bibliography of image processing books	552
Annex L (informative) Alphabetical listings of PIKS elements by profile	553
L.1 Alphabetical listing of PIKS elements in the Foundation profile	553
L.2 Alphabetical listing of PIKS elements in the Technical profile	555
L.3 Alphabetical listing of PIKS elements in the Scientific profile	559
L.4 Alphabetical listing of PIKS elements in the Full profile	566
Figure 1 — PIKS imaging model	6
Figure 2 — PIKS operator model: non-image to non-image operators	19
Figure 3 — PIKS operator model: image to non-image operators	20
Figure 4 — PIKS operator model: image to image operators	21
Figure 5 — Operator index assignment	21
Figure 6 — Example of match point translation for image subtraction	24
Figure 7 — Examples of ROI operation	25
Figure 8 — Example of the relationship between a ROI and an image	27
Figure 9 — Examples of PIKS element chains	31
Figure 10 — PIKS operational state diagram	39
Figure 11 — PIKS interface to the IIF gateway and an application	40
Figure 12 — PIKS to application interface	42
Table 1 — PIKS image objects	7
Table 2 — PIKS operators listed by functional class	11
Table 3 — PIKS tools listed by functional class	15

	Page
Table 4 — PIKS utilities listed by functional class	16
Table 5 — PIKS system mechanisms listed by functional class	17
Table 6 — PIKS data type codes	42
Table 7 — External physical data types supported by PIKS	43
Table 8 — Data types of PIKS non-image data objects supported in the Foundation profile	44
Table 9 — PIKS conformance profiles	45
Table 10 — Number of operators, tools, utilities, mechanisms, and total elements in each profile	46
Table 11 — PIKS elements in each profile	46

Descriptors: Data processing, information interchange, graphic data processing, image processing, communication procedure, specifications, operating requirements.

Foreword

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

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

International Standard ISO/IEC 12087-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 24, *Computer graphics and image processing*.

ISO/IEC 12087 consists of the following 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*.

Annex A, Annex G, Annex H and Annex J form an integral part of this part of ISO/IEC 12087.

Annex B, Annex C, Annex D, Annex E, Annex F, Annex K and Annex L are for information only.

1 Scope

This part of ISO/IEC 12087 establishes the specification of the application program interface (API), called the Programmer's Imaging Kernel System (PIKS). ISO/IEC 12087-1 establishes the conceptual and architectural definitions of the Common Architecture for Imaging (CAI). ISO/IEC 12087-3 establishes the specification of the Image Interchange Facility (IIF).

PIKS is intended to provide a rich set of both low-level and high-level services on image and image-derived data objects. These services can be used as building blocks for a broad range of common imaging applications.

A conscious effort has been made by the developers of PIKS to create a standard that does not favor any particular computing system. Implementations of PIKS should be possible on computing systems ranging in architecture from general purpose computers to specialised hardware accelerators, ranging in size from personal computers to mainframe supercomputers, and ranging in connectivity from stand-alone machines to distributed computing networks.

Where applicable, PIKS relies on other APIs and data format standards to provide capabilities that are not unique to imaging. The following lists contain a summary of technological capabilities provided by PIKS and not provided by PIKS. However, it should be noted that PIKS functionality may be useful as a pre-processor or co-processor for many of the technologies in the "Not provided by PIKS" list.

Provided by PIKS

- image analysis
- image classification (basic)
- image enhancement
- image interchange between PIKS and an application
- image interchange between PIKS and the IIF
- image manipulation primitives
- image processing data object generation tools (e.g., image filter functions)
- image restoration
- image visualization (basic)
- standard colour models

Not provided by PIKS

- audio
- computer graphics
- device control
- image acquisition
- image communication
- image compression and decompression
- image display
- image transport between applications
- image understanding
- multimedia
- pattern recognition
- specific implementations
- video
- window systems

NOTE The Image Interchange Facility of ISO/IEC 12087-3 specifies image compression and decompression functionality and image transport between applications and between an application and PIKS.

2 Normative references

The following standards contain provisions which, through references in this text, constitute provisions of this part of the 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 standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 12087-1:—, *Information technology — Computer graphics and image processing — Image Processing and Interchange (IPI) — Functional specification — Part 1: Common Architecture for Imaging*¹⁾.

ISO/IEC 12087-3:—, *Information technology — Computer graphics and image processing — Image Processing and Interchange (IPI) — Functional specification — Part 3: Image Interchange Facility*¹⁾.

3 Symbols and abbreviations

The following are symbols and abbreviations utilized in this standard. The mathematical functions are defined in Annex A.

0D	zero-dimensional
1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
4D	four-dimensional
5D	five-dimensional
ACOS	arc cosine function
AND	logical AND operator
API	Application Program Interface
ASIN	arc sine function
ASSIGN	index assignment function between image and operator input indices
ASN.1	abstract syntax notation one
ATAN	arc tangent function
ATAN2	arc tangent ratio function
b	spectral band index of an image
B	image spectral band size
BD	Boolean data type (PIKS internal)
BER	basic encoding rules
BI	external Boolean pixel data type
BITEXT	bit extraction function
BITINS	bit insertion function
BP	Boolean data type (PIKS parameter)
CAI	Common Architecture for Imaging
CAS	Hartley cas function
CCIR	Comite Consultatif International des Radiocommunications
CD	complex arithmetic data type (PIKS internal)
CF	external complex floating point pixel data type
CHAIN	chain data object
CHOICE	choice function
CIE	Commission Internationale de l'Eclairage
COLR	colour image
CONJ	complex conjugate function

¹⁾ to be published.

COS	cosine function
CP	complex arithmetic data type (PIKS parameter)
CS	character string data type
D	single operator output image
DET	determinant of matrix argument
DOB	destination non-image object
D _q	q-th operator output image, $1 \leq q \leq Q$
DST	single destination image
DST _q	q-th destination image $1 \leq q \leq Q$
EBU	European Broadcasting Union
EP	enumerated parameter
ERF	Gaussian error function
EXP	exponential function
EXT	external data type
GEN	generic image
HIST	histogram data object
i	square root of minus one, $i = \sqrt{-1}$
ID	data object identifier (PIKS internal)
ID_ARRAY	composite image identifier array
ID_LIST	composite image identifier list
ID_RECORD	composite image identifier record
IEC	International Electrotechnical Commission
IEEE	Institute Electrical and Electronic Engineers
IIF	Image Interchange Facility
IMAG	imaginary part of complex number argument function
in	input parameter
IP	data object identifier (PIKS parameter)
IPI	Image Processing and Interchange
ISO	International Organisation for Standardisation
j	first operator index
J	first operator index size
k	second operator index
K	second operator index size
l	third operator index
L	third operator index size
LIV	lowest integer value of argument function
LOG	base e natural logarithm function
LOOK	lookup table function
LUT	lookup table data object
m	fourth operator index
M	fourth operator index size
MATRIX	matrix data object
MAX	maximum of argument sequence function
MIN	minimum of argument sequence function
MOD	modulus function
MON	monochrome image

n	fifth operator index
N	fifth operator index size
NA	not applicable
NAND	logical NAND operator
NBHOOD_ARRAY	neighbourhood array data object
ND	non-negative integer data type (PIKS internal)
NI	external non-negative integer pixel data type
NIV	nearest integer value of argument function
NOR	logical NOR operator
NOT	logical NOT operator
NP	non-negative integer data type (PIKS parameter)
NTSC	National Television Systems Committee
NULL	null data type
OR	logical OR operator
out	output parameter
PIKS	Programmer's Imaging Kernel System
PIXEL_RECORD	pixel record data object
POW	power law function of two arguments
RD	real arithmetic data type (PIKS internal)
REAL	real part of complex number argument function
REASSIGN	index reassignment function between operator output and image indices
RES1	one-dimensional resampling function
RES2	two-dimensional resampling function
RES3	three-dimensional resampling function
RES4	four-dimensional resampling function
RES5	five-dimensional resampling function
RF	external real floating point pixel data type
ROI	Region-of-Interest
ROI_ARRAY	array Region-of-Interest data object
ROI_COORD	coordinate collection Region-of-Interest data object
ROI_ELLIP	elliptical Region-of-Interest data object
ROI_GEN	generic Region-of-Interest data object
ROI_POLY	polygon Region-of-Interest data object
ROI_RECT	rectangular Region-of-Interest data object
ROID	Region-of-interest of destination image
ROIDq	Region-of-interest of q-th destination image, $1 \leq q \leq Q$
ROIS	Region-of-interest of source image
ROISp	Region-of-interest of p-th source image, $1 \leq p \leq P$
RP	real arithmetic data type (PIKS parameter)
S	single operator input image
SD	signed integer data type (PIKS internal)
SI	signed integer number
SIN	sine function
SMPTE	Society of Motion Picture and Television Engineers
SOB	source non-image object
SORT	sort function

Sp	p-th operator input image, $1 \leq p \leq P$
SP	signed integer data type (PIKS parameter)
SPE	spectral image
SRC	single source image
SRCp	p-th source image, $1 \leq p \leq P$
STATIC_ARRAY	static array data object
t	temporal index of an image
T	image temporal size
TAN	tangent function
TEM	temporal image
TC	temporal-image
TI	two's complement fixed point integer number
TS	temporal-spectral image
TUPLE	tuple data object
VALUE_BOUNDS	value bounds collection data object
VOL	volume image
VC	volume-colour image
VREG	virtual register data object
VS	volume-spectral image
VT	volume-temporal image
VTC	volume-temporal-colour image
VTs	volume-temporal-spectral image
x	horizontal index (along rows) of an image
X	image horizontal size (image width)
XOR	exclusive OR operator
y	vertical index (along columns) of an image
Y	image vertical size (image height)
z	depth index of an image
Z	image depth size
$\cap$	boolean intersection symbol
$\cup$	boolean union symbol
$\bar{[]}$	overbar indicates complement of boolean argument
$ [] $	vertical lines indicate magnitude of arithmetic argument
$[\cdot]^T$	superscript T indicates matrix transpose
+	plus or positive symbol
−	minus or negative symbol
×	multiply symbol
/	arithmetic divide symbol
÷	integer divide symbol
$\oplus$	logical AND symbol
$\ominus$	logical OR symbol
$\otimes$	logical XOR symbol
$\circledast$	convolution symbol
$\boxplus$	morphological dilation symbol
$\boxminus$	morphological erosion symbol
Σ	sequence summation symbol

Π	sequence product symbol
M	sequence maxima symbol
N	sequence minima symbol
$\cup$	sequence union symbol
$\cap$	sequence intersection symbol

4 Programmer’s Imaging Kernel System specification

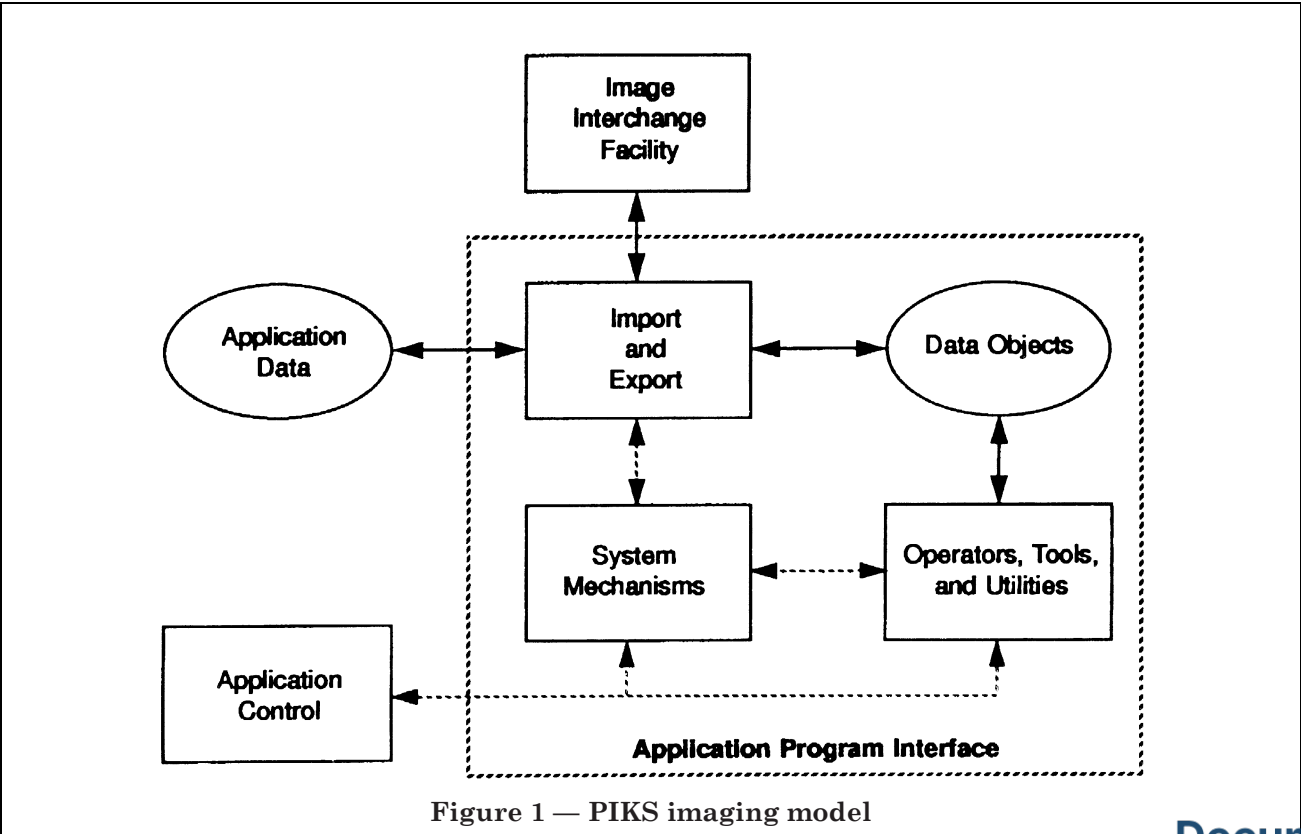
This clause establishes the specification of the Programmer’s Imaging Kernel System (PIKS) application program interface.

4.1 PIKS imaging model

Figure 1 describes the PIKS imaging model. The solid lines indicate data flow and the dashed lines indicate control flow. The PIKS application program interface consists of four major parts:

- data objects
- operators, tools, and utilities
- system mechanisms
- import and export

The PIKS data objects are specified in ISO/IEC 12087-1. They include both image and non-image data objects. The operators, tools, and utilities are functional elements that are used to process images or data objects extracted from images. The system mechanisms manage and control the processing. PIKS receives information from the application to invoke its system mechanisms, operators, tools, and utilities, and returns certain status and error information to the application. The import and export facility provides the means of accepting images and image-related data objects from an application, and for returning processed images and image-related data objects to the application. PIKS can transmit its internal data objects to an external facility through the Image Interchange Facility (IIF) specified in ISO/IEC 12087-3. Also, PIKS can receive data objects in its internal format, which have been supplied by the IIF. PIKS can operate independently of the IIF.



4.1.1 Image data objects

ISO/IEC 12087-1 describes the image data objects supported by PIKS. This subclause summarizes this information and establishes notation used in this part of ISO/IEC 12087.

As specified in ISO/IEC 12087-1, PIKS supports a general five-dimensional image object with ordered indices

$$0 \leq x \leq X - 1$$

$$0 \leq y \leq Y - 1$$

$$0 \leq z \leq Z - 1$$

$$0 \leq t \leq T - 1$$

$$0 \leq b \leq B - 1$$

where x is the horizontal index, y is the vertical index, z is the depth index, t is the time index, and b is the band index. The characters X , Y , Z , T , B represent the maximum sizes of the image coordinates x , y , z , t , b , respectively.

NOTE Some of the image dimensions may be unpopulated, e.g., for a monochrome image, $Z = 1$, $T = 1$ and $B = 1$.

A two-dimensional pixel array involving only two indices is called a pixel plane. A pixel plane involving the x - y indices is called a x - y pixel plane. Conceptually, a five-dimensional image object can be considered to be organized as a set of depth, time, and band x - y pixel planes.

NOTE PIKS operators are able to access and process arbitrary pixel planes.

In this part of ISO/IEC 12087, a single source image is expressed notationally as a five-dimensional array

$$\text{SRC}(x, y, z, t, b)$$

and the p -th source image in a set of P images is expressed notationally as

$$\text{SRC}_p(x, y, z, t, b)$$

where $1 \leq p \leq P$.

NOTE In general, source images can be of different sizes.

In this part of ISO/IEC 12087, a single destination image is expressed notationally as a five-dimensional array

$$\text{DST}(x, y, z, t, b)$$

and the q -th destination image in a set of Q images is expressed notationally as

$$\text{DST}_q(x, y, z, t, b)$$

where $1 \leq q \leq Q$.

NOTE In general, source and destination images can be of different sizes. However, some PIKS operators require that source and destination images be of the same size, e.g., the Fourier transform operator.

PIKS gives semantic meaning to certain dimensional subsets of a general five-dimensional image object. These are listed in Table 1. Definitions of these image objects are presented in ISO/IEC 12087-1.

Table 1 — PIKS image objects

Semantic description	image indices	code
monochrome	$x, y, 0, 0, 0$	MON
volume	$x, y, z, 0, 0$	VOL
temporal	$x, y, 0, t, 0$	TEM
colour	$x, y, 0, 0, b$	COLR
spectral	$x, y, 0, 0, b$	SPE
volume-temporal	$x, y, z, t, 0$	VT
volume-colour	$x, y, z, 0, b$	VC
volume-spectral	$x, y, z, 0, b$	VS
temporal-colour	$x, y, 0, t, b$	TC
temporal-spectral	$x, y, 0, t, b$	TS
volume-temporal-colour	x, y, z, t, b	VTC
volume-temporal-spectral	x, y, z, t, b	VTs
generic	x, y, z, t, b	GEN

NOTE The generic category represents images whose indices do not have any semantic meaning. For notational consistency, the generic image uses the same index notation as the semantic categories.

As specified in ISO/IEC 12087-1, PIKS supports the following abstract data types for the internal representation of image pixels.

<u>Code</u>	<u>Data type</u>
BD	Boolean
ND	non-negative integer
SD	signed integer
RD	real arithmetic
CD	complex arithmetic

All image pixels accessed by the x, y, z, t indices shall be of the same pixel data type. The pixel data type may differ between bands. An image whose pixels are of different data types across bands is called a heterogeneous band image.

A PIKS implementation shall be free to provide any form of physical representation of the image pixel data types provided that the physical representation satisfies the abstract data type definitions, as specified in ISO/IEC 12087-1. The minimum levels of storage precision for the ND, SD, RD, and CD pixel types shall be

ND	8 bits per pixel
SD	16 bits per pixel
RD	32 bits per pixel
CD	32 bits per real component and per imaginary component per pixel

4.1.2 Non-image data objects

ISO/IEC 12087-1 describes the non-image data objects supported by PIKS. This subclause summarizes this information and establishes notation used in ISO/IEC 12087-2.

PIKS supports the following internal, non-image data objects, which are defined in ISO/IEC 12087-1:

<u>Code</u>	<u>Name</u>
CHAIN	chain
HIST	histogram
ID_ARRAY	composite identifier array
ID_LIST	composite identifier list
ID_RECORD	composite identifier record
LUT	lookup table
MATRIX	matrix
NBHOOD_ARRAY	neighbourhood array
PIXEL_RECORD	pixel record
ROI_ARRAY	region-of-interest, array
ROI_COORD	region-of-interest, coordinate collection
ROI_ELLIP	region-of-interest, elliptical
ROI_GEN	region-of-interest, generic
ROI_POLY	region-of-interest, polygon
ROI_RECT	region-of-interest, rectangular
STATIC_ARRAY	static array
TUPLE	tuple
VALUE_BOUNDS	value bounds collection
VREG	virtual register

Non-image data objects that are created by PIKS elements and returned directly to an application are not regarded as PIKS non-image data objects. The data structures of directly-returned non-image data objects are defined by the elements providing the return. Collectively, such data objects are called external data objects, and are referenced by the code EXT.

In this part of ISO/IEC 12087, a single source non-image data object is expressed notationally as SOB, and the p -th source non-image data object in a set of P objects is expressed as SOB $_p$ where $1 \leq p \leq P$. In this standard, single destination non-image data object is expressed notationally as DOB, and the q -th destination non-image data object in a set of Q objects is expressed as DOB $_q$ where $1 \leq q \leq Q$.

4.1.3 Data object creation

PIKS source and destination data objects that are referenced by an operator, tool, or utility shall be allocated by a system management allocation mechanism, as described in 4.4, prior to the invocation of an operator, tool, or utility. There are three classes of allocated data objects:

Class A — identified with object attributes and data;

Class B — identified with object attributes only;

Class C — identified only

When an image is allocated, an identifier is assigned to the image data object by the PIKS implementation. The image allocation mechanism allows an application to specify certain image attributes, such as size, structure, and colour space, upon allocation, or to leave the attributes unspecified. When a class C data object is specified as a destination, many of the PIKS operators, tools, and utilities will supply the “missing” data object attributes.

NOTE A PIKS implementation may reserve data storage for a Class B data object at the time that the data object is allocated, or the implementation may defer data storage reservation until the time that the data for a data object is first available.

4.2 PIKS operators, tools, data object repository, utilities, and system mechanisms

This subclause establishes the general specification of the PIKS operators, tools, data object repository, utilities, and system mechanisms.

4.2.1 Operators

PIKS operators are functional elements that perform manipulations of images or of data objects extracted from images in order to enhance images, restore images, or to assist in the extraction of information from images. Operators can be classified as being primitive or high-level operators. Primitive operators are a small set of low-level operators, fundamental to image processing, which can be used to implement high-level operators. Examples are image convolution and image histogram generation. High-level operators are operators of greater complexity. Examples are adaptive histogram equalization and Hough transform.

NOTE There is no requirement in the PIKS specification that high-level operators shall be constructed from primitive operators.

Another form of operator classification is by functionality, e.g., operators that perform geometric manipulations such as rotate or operators that detect image edges. Table 2 contains a list of the PIKS operators arranged by functionality class. These operators are specified in clause 7.

Still another form of operator classification, as listed below, is by the types of source and destination objects supported by an operator.

<u>Source</u>	<u>Destination</u>
non-image	non-image
image	non-image
image	image

NOTE PIKS does not provide any non-image to image operators. PIKS does provide several image generation tools.

Table 2 indicates the type of source and destination objects supported by each functional class of operators. PIKS operators that have an image as a source data object can also be classified by their neighbourhood processing dimensionality as defined below.

0D A zero-dimensional operator is one that operates on each pixel of a source image independently of its neighbouring pixels. An example is taking the logarithm of a spectral image.

- 1D A one-dimensional operator is one that operates on a group of pixels along one dimension of an image. An example is a one-dimensional moving window average along the rows of a monochrome image. Another example is a one-dimensional Fourier transform along the columns of a monochrome image.
- 2D A two-dimensional operator is one that operates on a group of pixels along two dimensions of an image. An example is erosion of a monochrome image with a two-dimensional morphological structuring element.
- 3D A three-dimensional operator is one that operates on a group of pixels along three dimensions of an image. An example is convolution of a volume image with a three-dimensional impulse response function array.
- 4D A four-dimensional operator is one that operates on a group of pixels along four dimensions of an image. An example is computation of the four-dimensional Fourier transform of a volume-temporal image.
- 5D A five-dimensional operator is one that operates on a group of pixels along five dimensions of an image. An example is computation of the extrema pixels of a five-dimensional image.

NOTE 1 By this form of operator classification, the neighbourhood need not be compact. Also, the neighbourhood may contain all of the pixels along an image dimension.

NOTE 2 In subsequent clauses, an “rD” operator is called an operator of order r. The words “dimensional” and “dimension” are reserved, whenever possible, for image descriptors.

Annex B to Annex F contain alphabetical listings of PIKS elements with indications of the supported image structure, image data type, operator order, and element functionality.

4.2.2 Tools

PIKS tools are elements that create data objects to be used by PIKS operators. Examples are the generation of test images, the creation of impulse response function arrays, and the generation of Region-of-Interest objects.

Table 3 contains a list of PIKS tools. The PIKS tools are specified in clause 7.

4.2.3 Data object repository

Many data objects are routinely used by operators. An example is the set of Sobel edge detection horizontal and vertical edge gradient impulse response function arrays. PIKS provides a repository of commonly used data objects. Annex G contains a specification of the PIKS data object repository.

The data object repository contains three types of data:

- impulse response function arrays
- dither arrays
- colour conversion matrices

4.2.3.1 Impulse response function arrays

Impulse response function arrays in the data object repository are commonly-used instantiations of two-dimensional neighbourhood array data objects. In the repository, each array is specified according to the following general form:

$$\frac{1}{S} \begin{bmatrix} H(0,0) & H(1,0) & \dots H(C-1,0) \\ \vdots & \vdots & \vdots \\ H(0,r) & H(1,r) & \dots H(C-1,r) \\ \vdots & \vdots & \vdots \\ H(0,R-1) & H(1,R-1) & \dots H(C-1,R-1) \end{bmatrix}$$

where S is a scale factor.

NOTE 1 The impulse response function arrays are specified with signed integer scale factors and signed integer array terms. The physical storage data format is implementation dependent.

NOTE 2 The impulse response array indexing is the same as the indexing of a two-dimensional image array.

4.2.3.2 Dither arrays

Dither arrays in the data object repository are commonly-used instantiations of two-dimensional neighbourhood array data objects. In the repository, each dither array is specified according to the same general form as an impulse response function array.

4.2.3.3 Colour conversion matrices

Colour conversion matrices in the data object repository are commonly-used instantiations of matrix data objects. In the repository, each 3x3 matrix is specified according to the following general form:

$$\begin{bmatrix} T(1,1) & T(1,2) & T(1,3) \\ T(2,1) & T(2,2) & T(2,3) \\ T(3,1) & T(3,2) & T(3,3) \end{bmatrix}$$

All colour conversion matrices are specified for the transformation of an input column vector to an output column vector.

NOTE 1 ISO/IEC 12087-1 defines the colour spaces and associated chromaticity coordinates and white points used to generate each colour conversion matrix.

NOTE 2 In order to minimize quantization error in colour component conversion, the colour conversion matrices are specified in terms of six decimal digits of precision, even though the original definitions specify fewer digits of precision.

NOTE 3 The colour conversion matrix names specify the proper input and output colour spaces for colour component conversion.

4.2.4 Utilities

PIKS utilities are elements that perform basic mechanical image manipulation tasks such as extracting a pixel from an image and composing a volumetric image from a set of monochromatic images. Table 4 contains a list of PIKS utilities. These utilities are specified in clause 7.

4.2.5 System mechanisms

PIKS system mechanisms are elements that perform control and management tasks such as allocating data storage references for PIKS data objects, opening and closing PIKS sessions, error handling, and element chaining. Table 5 contains a list of PIKS mechanisms.

Table 2 — PIKS operators listed by functional class

ANALYSIS OPERATORS

Class of image to non-image operators that extract numerical information from an image.

- accumulator
- difference measures
- extrema
- histogram, one-dimensional
- histogram, two-dimensional
- Hough transform
- line profile
- moments
- value bounds

CLASSIFICATION OPERATORS

Class of image to image operators that classifies each pixel of a multispectral image into one of a specified number of classes based upon the amplitudes of pixels across image bands.

- classifier, Bayes
- classifier, nearest neighbour

Table 2 — PIKS operators listed by functional class**COLOUR OPERATORS**

Class of image to image operators that convert a colour image from one colour space to another.

- colour conversion, linear
- colour conversion, nonlinear
- colour conversion, subtractive
- colour lookup, interpolated
- luminance generation

COMPLEX IMAGE OPERATORS

Class of image to image operators that perform basic manipulations of images in real and imaginary or magnitude and phase form.

- complex composition
- complex conjugate
- complex decomposition
- complex magnitude

CORRELATION OPERATORS

Class of image to non-image operators that compute a correlation array of a pair of images.

- cross-correlation
- template match

EDGE DETECTION OPERATORS

Class of image to image operators that detect the edge boundary of objects within an image

- edge detection, orthogonal gradient
- edge detection, second derivative
- edge detection, template gradient

ENHANCEMENT OPERATORS

Class of image to image operators that improve the visual appearance of an image or that convert an image to a form better suited for analysis by a human or a machine.

- adaptive histogram equalization
- false colour
- histogram modification
- outlier removal
- pseudocolour
- unsharp mask
- Wallis statistical differencing

ENSEMBLE OPERATORS

Class of image to image operators that perform arithmetic (add, subtract, multiply), extremal (maximum, minimum), and logical (and, or) combination of a pair of images.

- alpha blend, constant
- alpha blend, variable
- dyadic, arithmetic
- dyadic, complex
- dyadic, logical
- dyadic, predicate
- split image
- Z merge

Table 2 — PIKS operators listed by functional class**FEATURE EXTRACTION OPERATORS**

Class of image to image operators that compute a set of image features at each pixel of an image.

- label objects
- Laws texture features
- window statistics

FILTERING OPERATORS

Class of image to image operators that perform neighbourhood combinations of pixels directly or by Fourier transform domain processing.

- convolve, five-dimensional
- convolve, two-dimensional
- filtering, homomorphic
- filtering, linear
- filtering, median
- filtering, pseudomedian
- filtering, rank order

GEOMETRIC OPERATORS

Class of image to image operators that perform geometric manipulations such as minification, magnification, rotation, and rubber sheet warping.

- cartesian to polar
- flip, spin, transpose
- polar to cartesian
- rescale
- resize
- rotate
- subsample
- translate
- warp, control point
- warp, lookup table
- warp, polynomial
- zoom

HISTOGRAM SHAPE OPERATORS

Class of non-image to non-image operators that generate shape measurements of a pixel amplitude histogram of an image.

- histogram shape, one-dimensional
- histogram shape, two-dimensional

MORPHOLOGICAL OPERATORS

Class of image to image operators that perform morphological operations such as erosion, dilation, and skeletonization on boolean and grey scale images.

- erosion or dilation, Boolean
- erosion or dilation, grey
- fill region
- hit or miss transformation
- morphic processor
- morphology
- neighbour count
- open and close

Table 2 — PIKS operators listed by functional class

PIXEL MODIFICATION OPERATORS

Class of image to image operators that modify an image by pixel drawing or painting.

- draw pixels
- paint pixels

POINT OPERATORS

Class of image to image operators that perform point manipulation such as magnitude, logarithm, and threshold on a pixel-by-pixel basis.

- bit shift
- complement
- error function scaling
- gamma correction
- histogram scaling
- level slice
- lookup
- lookup, interpolated
- monadic, arithmetic
- monadic, complex
- monadic, logical
- noise combination
- power law scaling
- rubber band scaling
- threshold
- unary, integer
- unary, real
- window-level

PRESENTATION OPERATORS

Class of image to image operators that prepare an image for display.

- diffuse
- dither

SHAPE OPERATORS

Class of image to non-image operators that label objects and perform measurements of the shape of objects within an image.

- perimeter code generator
- shape metrics
- spatial moments, invariant
- spatial moments, scale

UNITARY TRANSFORM OPERATORS

Class of image to image operators that perform multi-dimensional forward and inverse unitary transforms of an image.

- transform, cosine
- transform, Fourier
- transform, Hadamard
- transform, Hartley

Table 2 — PIKS operators listed by functional class**3D SPECIFIC OPERATORS**

Class of image to image operators that perform manipulations of three-dimensional image data.

- sequence average
- sequence Karhunen-Loeve transform
- sequence running measures
- 3D slice

Table 3 — PIKS tools listed by functional class**IMAGE GENERATION**

Class of tools that create test images.

- image, bar chart
- image, constant
- image, Gaussian image
- image, grey scale image
- image, random number image

IMPULSE RESPONSE FUNCTION ARRAY GENERATION

Class of tools that create impulse response function neighbourhood array data objects.

- impulse, boxcar
- impulse, derivative of Gaussian
- impulse, difference of Gaussians
- impulse, elliptical
- impulse, Gaussian
- impulse, Laplacian of Gaussian
- impulse, pyramid
- impulse, rectangular
- impulse, sinc function

LOOKUP TABLE GENERATION

Class of tools that create entries of a lookup table data object.

- array to LUT

MATRIX GENERATION

Class of tools that create matrix data objects.

- colour conversion matrix

REGION-OF-INTEREST GENERATION

Class of tools that create region-of-interest data objects from a mathematical description of the region-of-interest.

- ROI, coordinate
- ROI, elliptical
- ROI, polygon
- ROI, rectangular

Table 2 — PIKS tools listed by functional class

STATIC ARRAY GENERATION

Class of tools that create filter transfer function, power spectrum, and windowing function static array data objects.

- filter, Butterworth
- filter, Gaussian
- filter, inverse
- filter, matched
- filter, Wiener
- filter, zonal
- Markov process power spectrum
- windowing function

Table 4 — PIKS utilities listed by functional class

EXPORT FROM PIKS

Class of utilities that export image and non-image data objects from PIKS to an application or to the IIF.

- export histogram
- export image
- export LUT
- export matrix
- export neighbourhood array
- export ROI array
- export static array
- export tuple
- export value bounds
- get colour pixel
- get pixel
- get pixel array
- get pixel record
- output object

IMPORT TO PIKS

Class of utilities that import image and non-image data objects to PIKS from an application or from the IIF.

- import histogram
- import image
- import LUT
- import matrix
- import neighbourhood array
- import ROI array
- import static array
- import tuple
- import value bounds
- input object
- put colour pixel
- put pixel
- put pixel array
- put pixel record

Table 4 — PIKS utilities listed by functional class**INQUIRY**

Class of utilities that return information to the application regarding PIKS data objects, status, and implementation.

- inquire chain environment
- inquire chain status
- inquire elements
- inquire image
- inquire index assignment
- inquire non-image object
- inquire PIKS implementation
- inquire PIKS status
- inquire repository
- inquire resampling

INTERNAL

Class of utilities that perform manipulation and conversion of PIKS internal image and non-image data objects.

- constant predicate
- convert array to image
- convert image data type
- convert image to array
- convert image to ROI
- convert ROI to image
- copy window
- create tuple
- equal predicate
- extract pixel plane
- insert pixel plane

Table 5 — PIKS system mechanisms listed by functional class**CHAINING**

Class of system mechanisms that manage execution of PIKS elements inserted in chains.

- chain abort
- chain begin
- chain delete
- chain end
- chain execute
- chain reload

COMPOSITE IDENTIFIER MANAGEMENT

Class of system mechanisms that perform manipulation of image identifiers inserted in arrays, lists, and records.

- composite identifier array equal
- composite identifier array get
- composite identifier array put
- composite identifier list empty
- composite identifier list equal
- composite identifier list get
- composite identifier list insert
- composite identifier list remove
- composite identifier record equal
- composite identifier record get
- composite identifier record put

Table 5 — PIKS system mechanisms listed by functional class

CONTROL

Class of system mechanisms that control the basic operational functionality of PIKS.

- abort asynchronous execution
- close PIKS
- close PIKS, emergency
- open PIKS
- synchronize

ERROR

Class of system mechanisms that provide means of reporting operational errors.

- error handler
- error logger
- error test

SYSTEM MANAGEMENT

Class of system mechanisms that allocate, deallocate, bind, and set attributes of data objects and set global variables.

- allocate chain
- allocate composite identifier array
- allocate composite identifier list
- allocate composite identifier record
- allocate histogram
- allocate image
- allocate lookup table
- allocate matrix
- allocate neighbourhood array
- allocate pixel record
- allocate ROI
- allocate static array
- allocate tuple
- allocate value bounds collection
- allocate virtual register
- bind match point
- bind ROI
- deallocate data object
- define sub image
- return repository identifier
- set globals
- set image attributes
- set index assignment

VIRTUAL REGISTER

Class of system mechanisms that manage the use of virtual registers.

- vreg alter
- vreg clear
- vreg conditional
- vreg copy
- vreg create
- vreg delete
- vreg get
- vreg set
- vreg wait

4.3 PIKS operator model

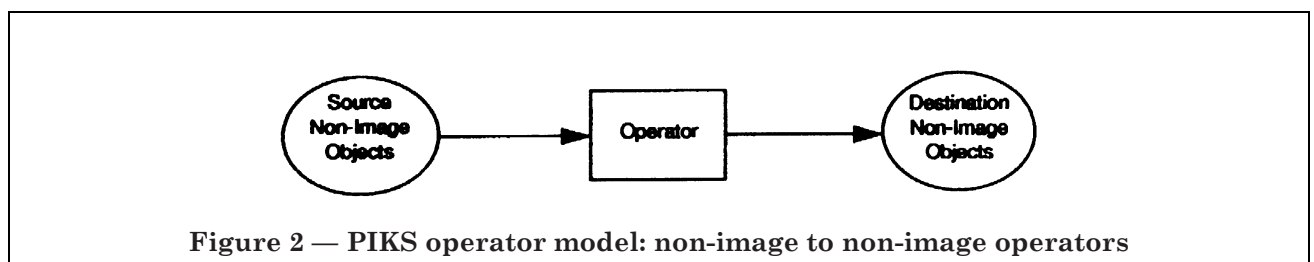
The PIKS operator model defines three possible transformations of PIKS data objects by a PIKS operator:

- non-image to non-image
- image to non-image
- image to image

Class A source data objects (identified with object attributes and data) shall exist prior to the invocation of an operator. Destination data objects may be Class A or Class B (identified with object attributes only). In the case of a Class A destination data object, the previously stored data will be overwritten unless there is inhibition invoked by a PIKS system mechanism, as described in 4.4.1. Some operators support Class C destination data objects (identified only) or Class B destination data objects with partial object attributes. Such operators shall supply the necessary object attributes. If the attributes of a source data object are mismatched to the attributes expected by an operator, or if the attributes of a destination data object are mismatched to the attributes expected by an operator, that operator shall generate an error indication at the time of execution.

4.3.1 Non-image to non-image operators

Figure 2 shows the PIKS operator model for the transformation of non-image data objects by an operator to produce other non-image data objects. An example of such a transformation is the generation of shape features from an image histogram. In this operator model, there is no explicit operator processing control; the transformation is completely defined by the specification of each operator, as presented in clause 7.



4.3.2 Image to non-image operators

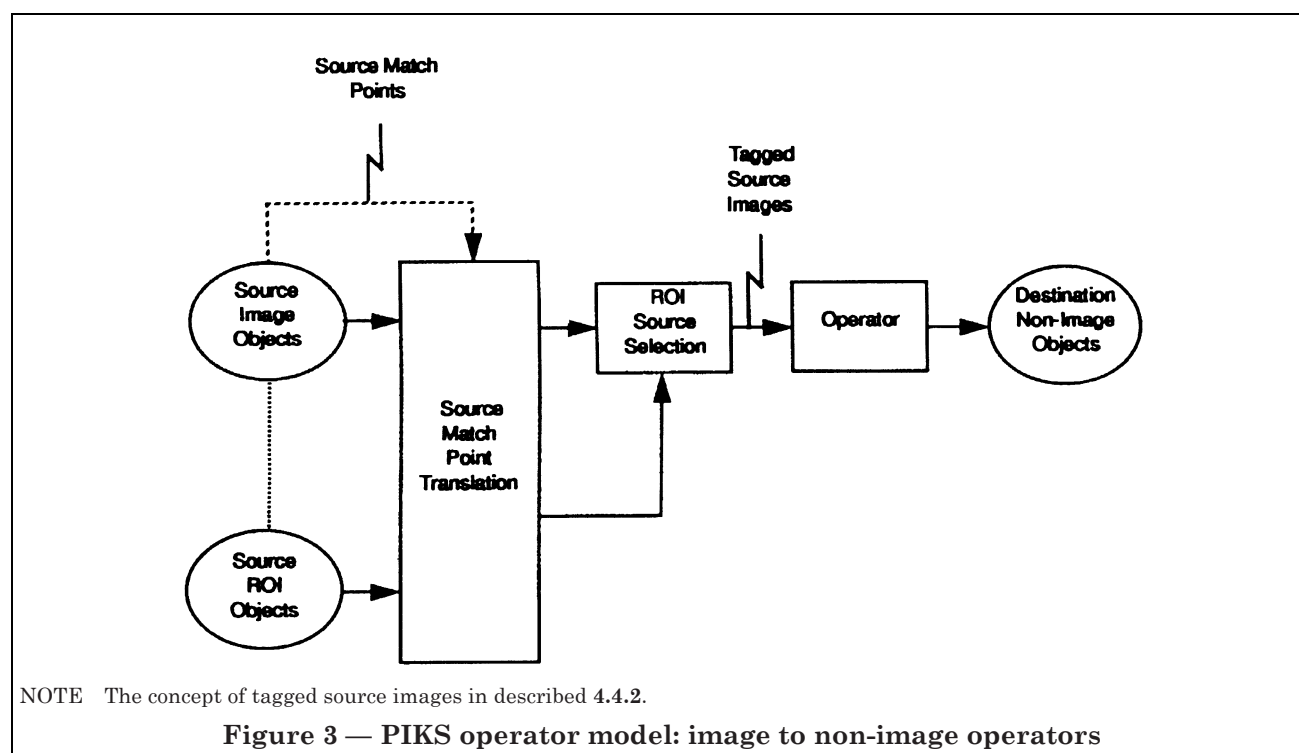
Figure 3 shows the PIKS operator model for the transformation of image data objects by an operator to produce non-image data objects. An example of such a transformation is the computation of the sum-squared difference between a pair of images. In this operator model, processing is subject to two control mechanisms: source match point translation and Region-of-Interest (ROI) source selection. As shown in Figure 3, source match point translation occurs first, followed by ROI source selection. This is followed by index assignment within the operator, as defined in Figure 5. These control mechanisms are defined in 4.4.2 and 4.4.3, respectively. The dashed line in Figure 3 indicates the transfer of control information. The dotted line indicates the binding of source ROI objects to source image objects.

4.3.3 Image to image operators

Figure 4 shows the PIKS operator model for the transformation of image data objects by an operator to produce other image data objects. An example of such a transformation is the unsharp masking operator that enhances detail within an image. In this operator model, processing is subject to four control mechanisms: source match point translation, destination match point translation, ROI source selection, and ROI destination selection. As shown in Figure 4, source match point translation occurs first, followed by ROI source selection. This is followed by index assignment within the operator, as defined in Figure 5. The destination match point translation and ROI destination selection control mechanisms are defined in 4.4.2 and 4.4.3, respectively.

NOTE Some PIKS operators are restricted to certain semantic types of image objects. For example, the colour processing operators are applicable only to colour images.

Certain PIKS operators are capable of accepting source images with pixels of non-negative, or signed integer data type, and automatically converting the source images to the real arithmetic data type for processing. The source promotion conversion process is implementation-dependent. The Element Functionality list associated with each PIKS operator indicates by “yes” or “no” if the operator supports source promotion.



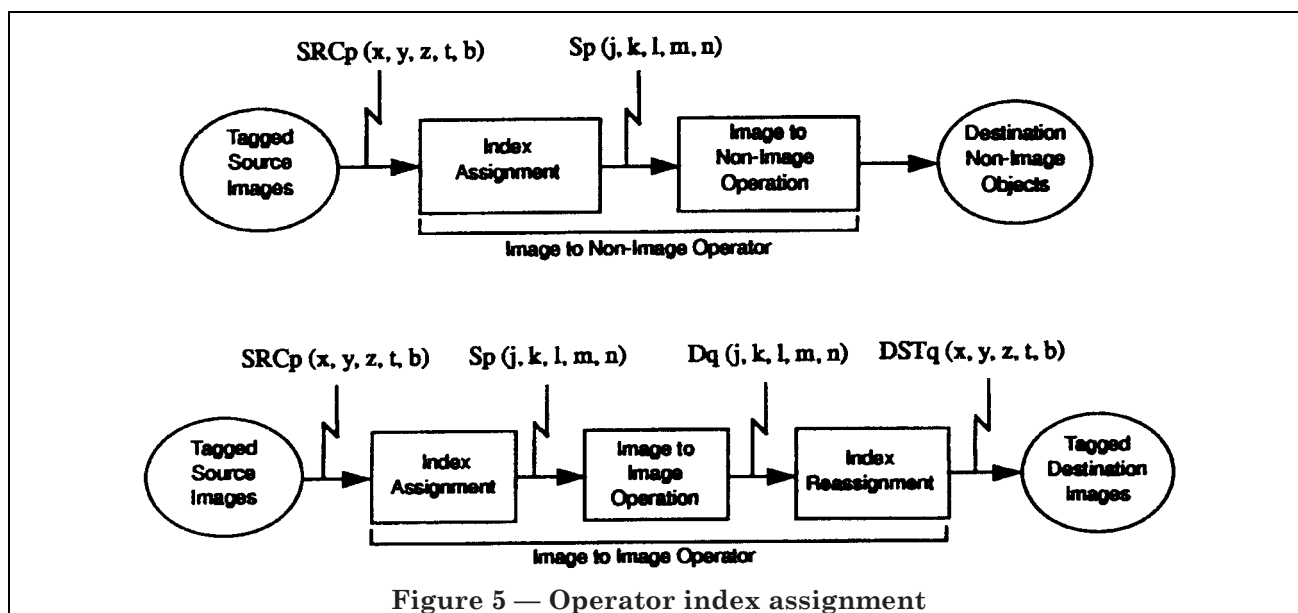
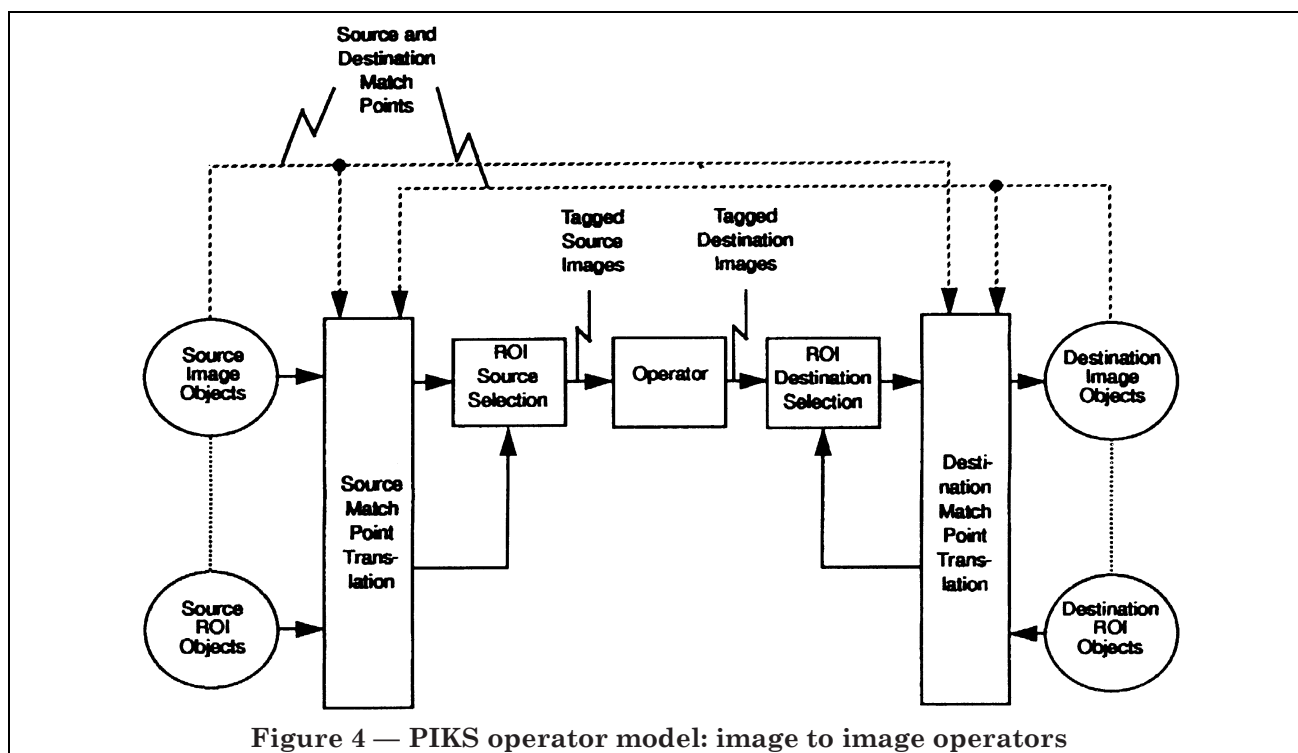
4.3.4 Neighbourhood operators

Some PIKS operators, e.g., two-dimensional convolution and dilation, perform linear or nonlinear combinations of source image pixels within a neighbourhood of a reference source pixel to produce a single destination image pixel. For such operators, the neighbourhood is specified by a neighbourhood array non-image data object as defined in ISO/IEC 12087-1. Each neighbourhood array has an associated key pixel, which is, in general, a five-dimensional coordinate offset measured with respect to the origin of the neighbourhood array. In operation, the key pixel is sequentially translated over all source image pixels, the linear or nonlinear neighbourhood combination is computed for each reference source pixel, and the computed result is recorded at the destination pixel corresponding to the reference source pixel that lies “under” the key pixel. In general, if the neighbourhood extends beyond the boundary of the source image, the “missing” source pixel values are assumed to be of zero value for arithmetic combinations or of FALSE value for Boolean combinations. This is the default condition. Some operators override the default condition by specifying the boundary conditions differently. For example, the *convolve_2d* operator has an option for which the “missing” neighbourhood pixels are replaced by pixels obtained by reflections about the boundaries of pixels within the source image.

4.3.5 Operator index assignment

Some PIKS image to non-image and image to image operators shall have the capability of assigning operator indices to image indices. This capability permits operators that are inherently r -th order, where $r > 0$, to be applied to five-dimensional images in a flexible manner.

Figure 5 illustrates the PIKS image to non-image and image to image operator index assignment and reassignment process. There is a global index assignment 5-tuple, which relates the ordering of the five image indices (x, y, z, t, b) to five generic operator indices (j, k, l, m, n).



Prior to the execution of an operator, each source image $SRCp(x, y, z, t, b)$ is logically converted to an operator input image $Sp(j, k, l, m, n)$ according to a global operator assignment specification table, generated by the *index_assignment* system management mechanisms, as shown below.

Operator Index	Image Index
j	x or y or z or t or b
k	x or y or z or t or b
l	x or y or z or t or b
m	x or y or z or t or b
n	x or y or z or t or b

The character assignment [x, y, z, t, b] must be unique, i.e., if x is assigned to the j operator index, it cannot be assigned to any other operator index. Operator index assignment is not permitted for an rD operator if the band index is to be reassigned to one of the first r indices, and the source image does not have bands of the same data type.

The assignment process defined above is expressed functionally as

$$Sp(j, k, l, m, n) = SRCp(ASSIGN\{x, y, z, t, b\})$$

The destination image index reassignment process from an operator output image Dq(j, k, l, m, n) to a destination image DSTq(x, y, z, t, b) is a direct inverse of the assignment process. The reassignment process is expressed functionally as

$$DSTq(x, y, z, t, b) = Dq(REASSIGN\{j, k, l, m, n\})$$

NOTE The index assignment and index reassignment processes are described in the *index_assignment* system management mechanism described in clause 7.

As an example, consider a temporal source image SRC(x, y, 0, t, 0), which is input to a 3D convolution operator to perform image smoothing in x-y-t image space. The following index assignment table could be formed.

<u>Operator Index</u>	<u>Image Index</u>
j	x
k	y
l	t
m	z
n	b

For an rD operator, the assignment sequence of the (r + 1)-st and higher indices is immaterial. In the example, above, b could have been assigned to m, and z could have been assigned to n.

For most operators, if the operator is of order less than the image dimension, then the operator is executed independently over the pixel planes not directly involved in the operator. As an example, consider a temporal-colour source image SRC(x, y, 0, t, b), which is input to a 2D convolution operator to perform image smoothing in each x-y pixel plane. The following index assignment table could be formed.

<u>Operator Index</u>	<u>Image Index</u>
j	x
k	y
l	z
m	t
n	b

The 2D convolution operation will be executed for all T temporal planes and all B bands. A 2D operator need not be restricted to operate on the x-y pixel planes of a source image. It could, for example, operate on the t-x pixel planes of a temporal image by assignment of t to j and x to k.

NOTE 1 A PIKS implementation may choose to physically reorder the indices of an image or choose to pass to an operator the index assignment information

NOTE 2 The index assignment technique applies to some utilities and to some image generation and ROI generation tools.

4.4 PIKS system mechanisms

This subclause describes the PIKS system mechanisms. These mechanisms are specified in clause 7.

4.4.1 Data object allocation

PIKS shall provide automatic memory management that permits an application to specify image and non-image data objects by object identifiers, and not be concerned with physical storage mechanisms. Image data object identifiers are created by the *allocate_image* system management mechanism. Non-image data object identifiers are created by an *allocate* system management mechanism corresponding to the non-image data object name. Image and non-image attribute information is accessed via the object identifier by inquiry mechanisms.

Any temporary images that may be required for storage of intermediate results in an implementation of a PIKS operator shall be automatically generated. A PIKS application does not have access to temporary images through PIKS.

PIKS does not specify the internal physical storage format of an implementation. PIKS implementations may choose to store images as conventional pixel planes, tiled planes, interleaved pixels, or any combination of storage means. The physical storage mechanism is hidden from the PIKS application.

Each PIKS implementation shall establish image size and capacity limits, and shall make this information available to the PIKS system inquire utility.

4.4.2 Match point control

Each PIKS image object has an associated match point coordinate set (x, y, z, t, b) , which some PIKS operators utilize to control multi-dimensional translations of images prior to processing by an operator. Notes associated with each operator indicate if the operator is or is not subject to match point control. In order to simplify the mathematical definitions of the operators, they are expressed as if there were no match point control.

NOTE Match points need not be integer valued.

The generic effect of match point control for an operator that creates a single destination image from a single source image is to independently translate each dimension of the source image such that its match point is aligned with corresponding match point of the destination image prior to processing. This generic operation is mathematically defined as follows. Let

$SRC(x, y, z, t, b) = X_s x Y_s y Z_s z T_s t B_s b$ source image

$DST(x, y, z, t, b) = X_D x Y_D y Z_D z T_D t B_D b$ destination image

$(x_{sm}, y_{sm}, z_{sm}, t_{sm}, b_{sm}) =$ source match point

$(x_{dm}, y_{dm}, z_{dm}, t_{dm}, b_{dm}) =$ destination match point

$O\{\cdot\} =$ general operator

Then

$DST(x, y, z, t, b) = O\{RESr\{SRC, x - x_{dm} + x_{sm}, y - y_{dm} + y_{sm}, z - z_{dm} + z_{sm}, t - t_{dm} + t_{sm}, b - b_{dm} + b_{sm}\}\}$

for the destination image pixels x, y, z, t, b lying in the range

$MAX\{0, x_{dm} - x_{sm}\} \leq x \leq MIN\{X_D - 1, X_S - 1 + x_{dm} - x_{sm}\}$

$MAX\{0, y_{dm} - y_{sm}\} \leq y \leq MIN\{Y_D - 1, Y_S - 1 + y_{dm} - y_{sm}\}$

$MAX\{0, z_{dm} - z_{sm}\} \leq z \leq MIN\{Z_D - 1, Z_S - 1 + z_{dm} - z_{sm}\}$

$MAX\{0, t_{dm} - t_{sm}\} \leq t \leq MIN\{T_D - 1, T_S - 1 + t_{dm} - t_{sm}\}$

$MAX\{0, b_{dm} - b_{sm}\} \leq b \leq MIN\{B_D - 1, B_S - 1 + b_{dm} - b_{sm}\}$

where $RESr\{\cdot\}$, for $1 \leq r \leq 5$, denotes a resampling function, as defined in Annex H, whereby destination image pixels are generated from interpolated source image pixels.

The destination image $DST(x, y, z, t, b)$ is only modified by operators within the bounds specified above. All other portions of the destination image remain unchanged if it is an existent Class A image. If the destination image is a Class B or Class C image (void of image data) prior to the operation, pixels in all other portions of the destination image are set to zero value if the destination image is an arithmetic data type or set to the FALSE state if the destination image is a boolean data type. As indicated by the generic definition, the source and destination images are aligned by their match points prior to an image operation. Match point alignment occurs prior to ROI control and image to operator index assignment.

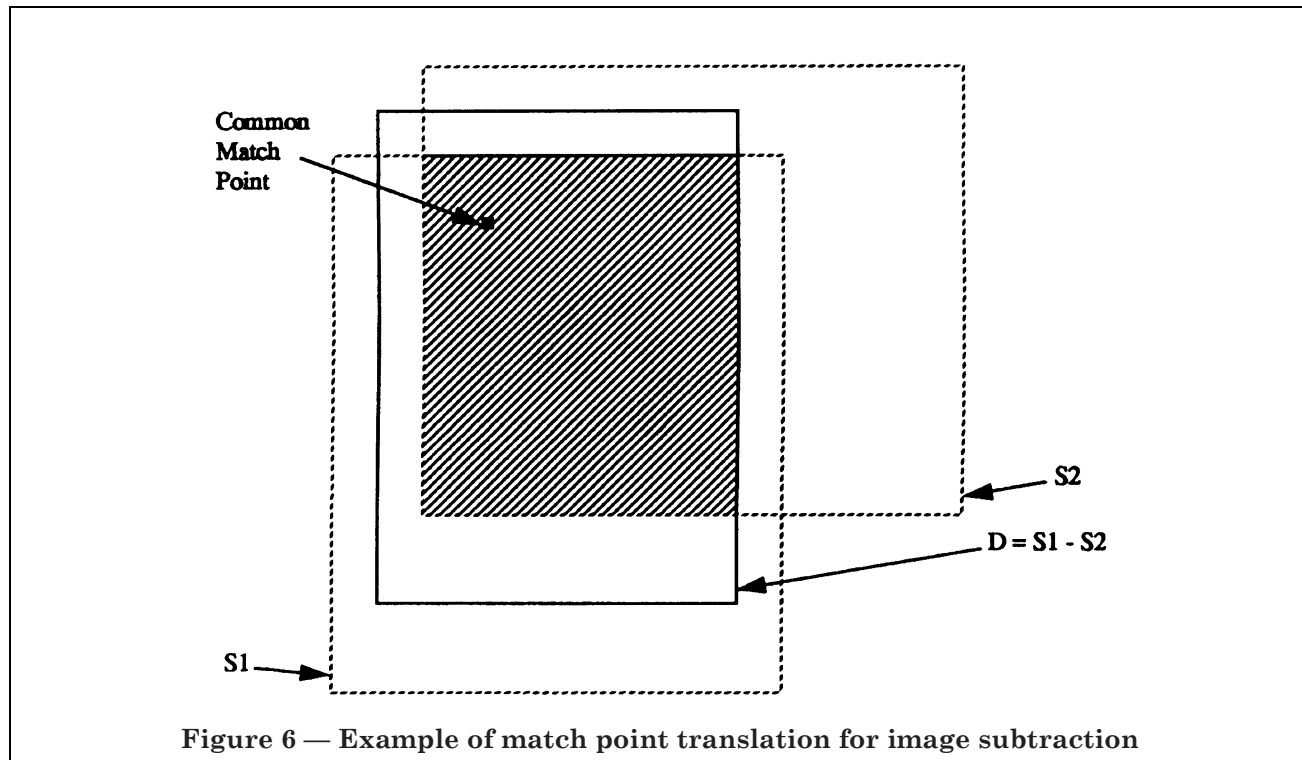
The generic effect of match point control for an operator that creates multiple destination images from multiple source images is to translate each source image and each destination image, other than the first destination image, such that the match points of these images are aligned with the match point of the first destination image prior to processing. Figure 6 provides an example of match point translation for the subtraction of a pair of monochrome images. In the example, the difference region is shown cross-hatched. A ROI that is bound to an image is spatially translated in register with the image to which it is bound. Each translated image and bound ROI control object is resampled according to the global resampling options specified in 4.4.8.

NOTE The match point control mechanism can be used to extract a lower dimensional subset of an image for processing. For example, by properly choosing the source and destination match points, and by properly selecting the spatial size of the destination image, it is possible to process only a spatial window of each pixel plane of a source image.

4.4.3 ROI control

The PIKS region-of-interest (ROI) control mechanism shall adhere to the following definition.

NOTE This ROI definition conceptually describes the semantics of the PIKS ROI functionality. There is no requirement that the functionality shall be implemented in the conceptual manner described.



Region-of-interest is a mechanism for controlling which pixels within an image will be processed by an operator, for specifying how they will be processed (for some, but not all operators), and for specifying how the processed pixels will be recorded in a destination image. Conceptually, a ROI control object is a Boolean-valued, five-dimensional array that is associated with each source and destination image. It is of the same size as its associated image, and is in spatial register with respect to the image origin. The ROI control object array assumes one of two logical states at each pixel. If a ROI pixel is in the logical TRUE state, the corresponding image pixel is said to be inside the ROI. Conversely, an image pixel is outside the ROI if its ROI pixel is in the logical FALSE state.

A ROI control object is created, conceptually, by a ROI binding mechanism that binds a ROI data object to an image. The ROI binding mechanism creates a ROI data object identifier reference in the image attribute list associated with each image. ROI data object creation and binding is described in 4.4.4.

If an image attribute list makes no reference to a ROI, a default ROI control object is assumed to exist and to be logically TRUE for each pixel of the image.

NOTE The ROI control object defined in this clause is used as a means of explaining the operation of ROI control in the PIKS operator model. The ROI control object is not a standardized PIKS data object.

In the PIKS operator model, the source ROI control object is a selector that chooses which source pixels are to be processed. The destination ROI control object is a spatial selector that chooses which processed source pixels are to be recorded in a destination image. The selection operations are defined only at integer pixel coordinates, i.e., PIKS does not support subpixel ROI selection.

The image to operator index assignment operation specified in 4.3.4 applies to a ROI control object bound to an image as well as to the image itself. For example, if the y index of an image is associated with the first operator input index, then the y index of the associated ROI control object is also associated with the first operator input index.

Figure 4 contains a diagram that conceptually describes the ROI operation in the PIKS operator model for image to image operators. In the special case of the processing of multiple source images by a point processing operator to produce a single destination image, let

$\text{SRCp}(x, y, z, t, b) = p\text{-th source image for } 1 \leq p \leq P$

$\text{DST}(x, y, z, t, b) = \text{destination image}$

$\text{P}(x, y, z, t, b) = \text{operator output image without ROI control}$

$\text{O}(x, y, z, t, b) = \text{operator output image when ROI is invoked}$

$\text{ROISp}(x, y, z, t, b) = p\text{-th source ROI control object}$

$\text{ROIS}(x, y, z, t, b) = \text{composite source ROI control object}$

$\text{ROID}(x, y, z, t, b) = \text{destination ROI control object}$

A composite source ROI control object, $\text{ROIS}(x, y, z, t, b)$, is conceptually formed by taking the logical intersection of all of the source ROI control objects. Thus

$\text{ROIS}(x, y, z, t, b) = \text{ROIS1}(x, y, z, t, b) \cap \text{ROIS2}(x, y, z, t, b) \cap \dots \cap \text{ROISP}(x, y, z, t, b)$

Conceptually, the composite source ROI control object tags each pixel in the source image as being inside or outside the ROI, and conceptually toggles a source control switch within the operator such that

$\text{O}(x, y, z, t, b) = \text{P}(x, y, z, t, b)$

if $\text{ROIS}(x, y, z, t, b)$ is TRUE. There is no operator output if $\text{ROIS}(x, y, z, t, b)$ is FALSE.

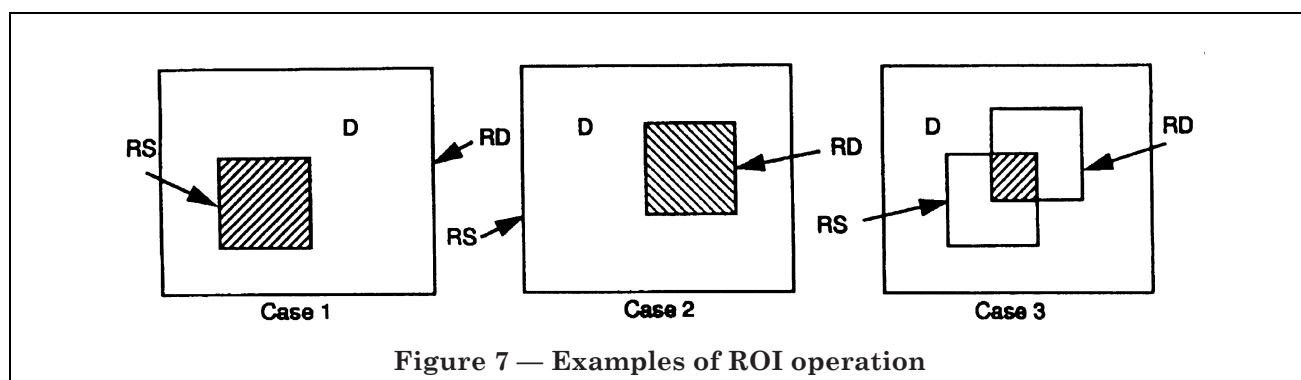
Conceptually, the destination ROI control object toggles a destination control switch such that

$\text{DST}(x, y, z, t, b) = \text{O}(x, y, z, t, b)$

if $\text{ROIS}(x, y, z, t, b)$ is TRUE and if $\text{ROID}(x, y, z, t, b)$ is TRUE. The destination image remains unchanged if $\text{ROID}(x, y, z, t, b)$ is FALSE or if $\text{ROIS}(x, y, z, t, b)$ is FALSE and $\text{ROID}(x, y, z, t, b)$ is TRUE.

Figure 7 illustrates three cases of ROI functionality for point processing of a monochrome image. In Case 1, the destination ROI is logically TRUE over the full image extent, and the source ROI is TRUE over a rectangular region smaller than the full image. In this case, the destination image consists of the existing destination image with an insert of processed source pixels. For Case 2, the source ROI is of full extent, and the destination ROI is of a small rectangular extent. The resultant destination image consists of processed pixels inserted into the existing destination image. Functionally, the result is the same as for Case 1. The third case shows the destination image when the source and destination ROIs are overlapping rectangles smaller than the image extent. In this case, the processed pixels are recorded only in the overlap area of the source and destination ROIs.

The ROI concept extends to multiple destination images. Each destination image has a separately bound ROI control object, which independently controls recording of pixels in the corresponding destination image.



The ROI concept applies to neighbourhood operators as well as point operators. Each neighbourhood processing object, such as an impulse response array or a morphological structuring element, has a pre-defined key pixel. If the key pixel lies within a source control ROI, the output pixel is formed by the neighbourhood operator even if any or all neighbourhood elements lie outside the ROI.

Some PIKS operators are under match point control. For such operators, the source and destination images and their registered ROI control objects are translated such that their match points are in alignment prior to ROI processing. Image processing operations are valid only in the intersection region of the translated source and destination images and their corresponding ROIs.

When a ROI control object is translated under match point control along with the image to which it is bound, the ROI resampling option invoked is applied to the ROI control object. For resampling, the ROI control object is conceptually treated as a bi-level image in which the TRUE state pixels are represented as value 1.0 and FALSE state pixels are represented as value 0.0. After the resampling interpolation operation is completed, the interpolated ROI control object is thresholded at a level of 0.5 for which the equal or above threshold value is set TRUE and the below threshold value is set FALSE. ROI resampling is defined mathematically in Annex H.

Some PIKS operators, such as the unitary transform operators, are not subject to ROI control because they are only defined over rectangular extent images. It is noted in the specification of an operator if it is or is not subject to ROI control.

In order to simplify the mathematical definition of the operators in clause 7, they are expressed as if there were no ROI control.

Some PIKS operators, e.g., rotate, can perform the same processing operation on a source ROI control object as on a source image. For such operators, the existing destination ROI control object can be replaced by the processed source ROI control object. The Element Functionality list associated with each PIKS element indicates by “yes” or “no” if the element supports ROI processing. The processing of a source ROI control object and subsequent replacement of the existing destination ROI control object can be suppressed by the *set_globals* system management mechanism described in 4.4.8.

NOTE In PIKS, ROI control is only defined conceptually. Some implementations may choose to implement ROIs as physical bitmaps. Other implementations may choose to implement ROIs as physical coordinate lists or as run length code generators.

4.4.4 ROI data object creation and manipulation

PIKS provides tools for generating ROI data objects from higher level specifications. Such supported specifications include:

Name	Code
array	ROI_ARRAY
coordinate collection	ROI_COORD
elliptical shape	ROI_ELLIP
polygonal shape	ROI_POLY
rectangular shape	ROI_RECT

These tools, together with the ROI binding mechanism, provide the capability to conceptually generate five-dimensional ROI control objects from lower dimensional descriptions by pixel plane extensions. For example, with the elliptical ROI generation tool, it is possible to generate a circular disk ROI in a spatial pixel plane, and the cause the disk to be conceptually replicated over the other pixel planes of a volume image to obtain a cylinder-shaped ROI. The specific functionality of the ROI tools is defined in clause 7.

Conceptually, a ROI data object is a Boolean-valued, five-dimensional array. The ROI data object array assumes one of two logical states (TRUE or FALSE) at each point in the array. Figure 8 presents an example of the spatial cross-section of an elliptical ROI data object. The TRUE state region is shown cross-hatched.

In ISO/IEC 12087-1, a ROI data object is expressed, notationally, as a five-dimensional array:

ROI(x, y, z, t, b)

The PIKS ROI data objects are specified in ISO/IEC 12087-1. Associated with each image is a key point 5-tuple (x_k, y_k, z_k, t_k, b_k) consisting of five signed integers, which defines an offset of ROI origin with respect to the image origin. When a ROI data object is bound to an image, conceptually, the ROI data object is translated in each orthogonal direction such that the origin of the ROI data object is aligned in five-dimensional space with the key point in the image to which the ROI data object is bound. The directional sense (left/right, up/down, etc.) is the same as an image translated by match point alignment. The ROI control object is conceptually formed by clipping the ROI data object to the extent of the image. If the ROI data object is not in the TRUE state at the geometric centre of a particular pixel of the image, the ROI control object is set FALSE at that pixel position. Figure 8 shows an example of the spatial relationship between an ROI data object and an image to which it is bound.

NOTE The key point may lie outside the extent of the ROI data object and the image.

PIKS provides utilities for converting a Boolean image to a ROI array data object and for converting a ROI array data object to a Boolean image.

PIKS provides operators for the processing of ROI data objects. For example, a rectangular ROI data object can be resized to produce another rectangular-shaped ROI data object. Such operators can accept as a source ROI any of the ROI data object structures, e.g., ROI_ARRAY, ROI_RECT, etc., but the operator output is specified only as being a generic ROI structure. The ROI_GEN structure is implementation-dependent.

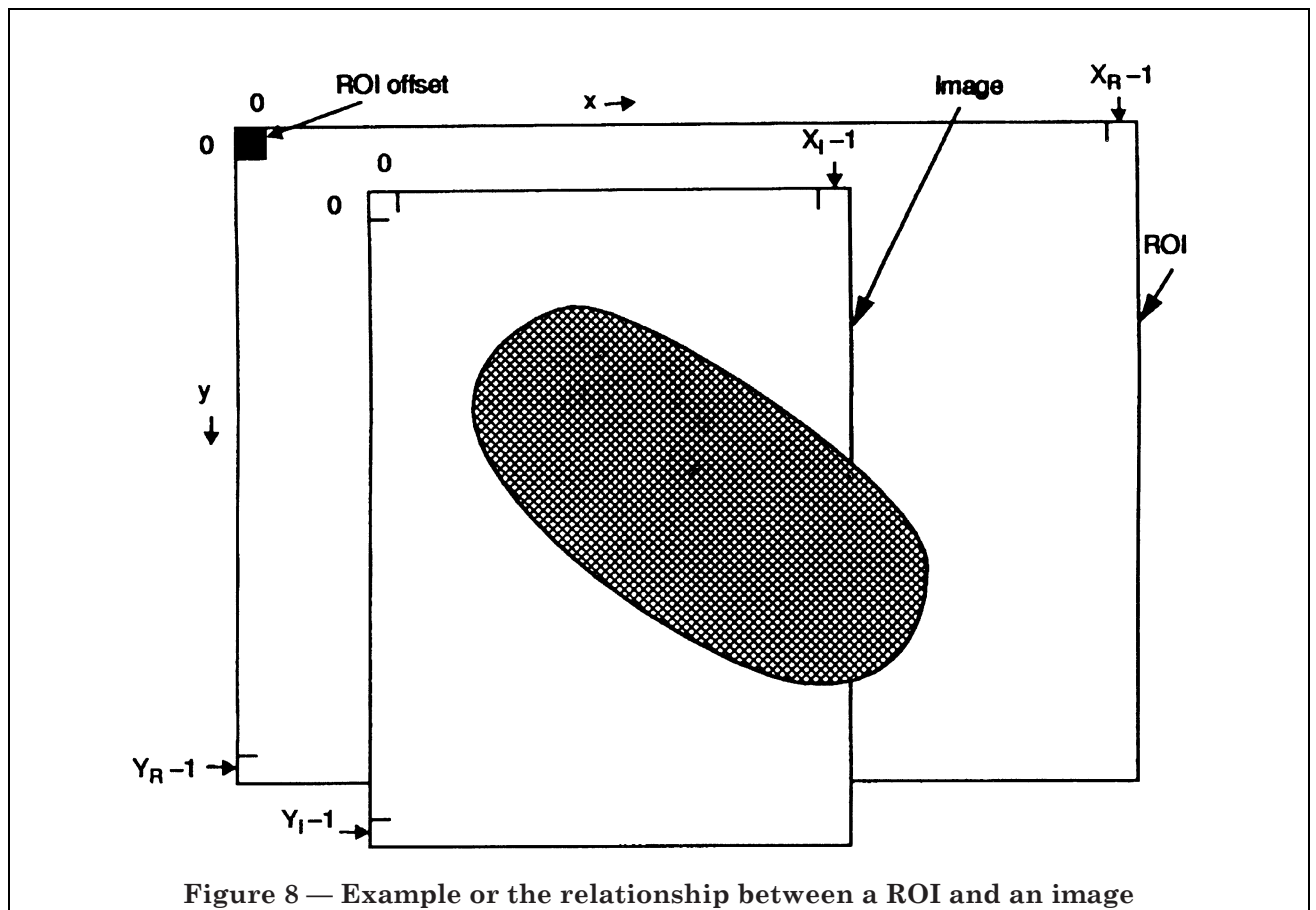


Figure 8 — Example of the relationship between a ROI and an image

4.4.5 Asynchronous control

PIKS provides two global modes in which certain of its elements (operators, tools, utilities, or mechanisms) may be executed: synchronous or asynchronous. The global PIKS mode is initially set to Synchronous by the *open_piks* mechanism and remains in that state until changed to Asynchronous by invocation of the *set_globals* mechanism. The Element Functionality list associated with each element indicates by “yes” or “no” if the element supports asynchronous execution.

The following control mechanisms relating to asynchronous execution are available:

set_asynchronous (via the *set_globals* mechanism)

set_synchronous (via the *set_globals* mechanism)

synchronize

abort_asynchronous_execution

When PIKS elements are executed synchronously, all processing and error reporting is completed before the implementation returns control to the calling application. This is the least sophisticated mode of operation, and shall be supported by all implementations.

Asynchronous execution permits implementations to use multiple processors or external accelerator devices to take advantage of possible parallel or distributed execution resources. It also allows for some optimization of the execution of sequences of PIKS elements because execution need not begin prior to returning to the application. The implementation may defer execution of a sequence of PIKS elements in order to combine consecutive PIKS elements into single macro operations or to execute a sequence of operations in a way that reduces loop control overhead, lessens the temporary storage space needed or better allocates available processing facilities to the deferred set of operations. The methods provided to enhance asynchronous operations are strictly internal to the implementation, outside the scope of this standard, and need not be visible to the application.

Asynchronous execution also permits the application to perform other activities, such as numerically intensive processing, the reading and writing of files, input/output operations, or other time consuming processing, while a PIKS implementation is performing the enqueued imaging operations.

The degree of optimization supplied in the asynchronous mode is strictly implementation-dependent. In fact, all conformance profiles allow an implementation to support the asynchronous mode with synchronous execution, without violating conformance to the standard. In this case, all operations in the asynchronous mode would actually be performed in the synchronous mode. An application may perform in *inquire_piks_implementation* to verify whether asynchronous operations are supported by a particular PIKS profile.

In the asynchronous mode, each PIKS element is added to the end of a queue of all pending asynchronous PIKS elements. This mechanism is referred to as the Asynchronous Execution Queue. The parameters associated with each enqueued PIKS element are also fetched and saved to prevent subsequent changes by the application from modifying the function to be performed. The queue keeps the elements in the same sequence that they were called by the application. Control may be immediately returned to the application, prior to initiation of execution, without any error checking or reporting. Although the timing of the error detection in parameters of asynchronous calls is implementation-dependent, the resulting error detection and error handling produce the standard-specified results via the PIKS error mechanisms.

A synchronization control mechanism, called *synchronize*, is provided to allow an application to request a wait until all pending asynchronous PIKS elements have been completed. While the *synchronize* function causes the application to wait, it does not affect the execution of asynchronous elements.

Some PIKS elements cause an implicit synchronization. For example, PIKS elements that must return a value of a non-PIKS data type to the application must wait until processing is completed before they can return a result, and then return control to the calling program. Most PIKS operations do not cause implicit synchronization. Because PIKS data types are referenced by identifiers — not values — their values are automatically updated by an asynchronous element, and are available to asynchronous elements invoked later. An invocation of the *set_globals* mechanism, when the synchronicity mode is set to SYNCHRONOUS, causes an implicit synchronization.

The *close_piks* mechanism does an implicit synchronization to assure that all operations have been completed prior to the closing actions.

Execution of the enqueued asynchronous operations is controlled by the specific scheduling mechanism provided by the implementation. It must begin and complete execution of all enqueued elements prior to giving a return to any explicit or implicit *synchronize* call. In most cases, PIKS elements will not be executed as fast as they can be added to the queue. In the event that the asynchronous execution control does complete all enqueued elements, it simply waits until additional elements are entered to the queue to resume asynchronous processing.

All PIKS elements which do not cause a synchronization prior to their execution are placed on the Asynchronous Execution Queue. This includes operations that simply change a global state, such as the resampling mode. This will ensure that all operations are executed in the order they were invoked, and that earlier operations properly affect later operations avoiding the side-effects that could be caused by changes in the global state during the execution of asynchronous elements invoked by prior calls.

The only exceptions to the sequential execution rule are the inquiry operations, which always return their values immediately and which do not directly affect the execution of any element. An application is free to invoke an explicit synchronization prior to performing an inquiry element if it wishes to inquire state values or to affect virtual register values after all asynchronous elements have been executed.

The *abort_asynchronous_execution* mechanism is provided for the application to terminate all on-going asynchronous operations and to clear the Asynchronous Execution Queue. This procedure can be used to recover from serious error conditions or when the program or user changes the flow of processing.

While in “synchronous” mode, invocation of the *synchronize* or *abort_asynchronous_execution* control elements is treated as a no-operation mechanism.

If Asynchronous is “yes” and the Asynchronous Mode is in effect, the element will not be immediately executed if another element is presently being executed. It will instead be put onto the Asynchronous Execution Queue and executed when it reaches the head of the queue.

NOTE For implementations which do not support the Asynchronous Mode, operations are always executed asynchronously, as if the Synchronous Mode was in effect.

If Asynchronous is “yes” and the Asynchronous Mode is not in effect, the element will not be put onto the Asynchronous Execution Queue, but will be executed immediately.

If Asynchronous is “no” the new element will not be put onto the Asynchronous Execution Queue even if Asynchronous Mode is in effect. Instead, the implicit synchronize mechanism is invoked, that is, execution of the new element will be deferred until all presently queued elements have been executed asynchronously; then the new element is executed. Exceptions to this operation are the *abort_asynchronous_execution* and *close_piks_emergency* mechanisms which are executed immediately upon being invoked.

4.4.6 Element chaining

PIKS provides a mechanism, known as chaining, for the assembly and controlled execution of a sequence of PIKS elements. Using this mechanism, an implementation can provide for the efficient execution of element sequences.

The primary objective of chaining is to facilitate the optimization of element sequences. This includes minimizing data storage requirements, eliminating unnecessary data flow, maximizing the use of hardware facilities, and condensing sequences of PIKS elements into single operations. Because entire sequences of PIKS elements can be analyzed as a whole, individual elements may be restructured (but not resequenced in any manner that changes the result of the computation) to suit the facilities available for their execution. The choice and extent of the optimization provided rests solely with each PIKS implementation; such optimizations shall not change the result of a computation.

Chains are constructed and stored within an implementation during the CHAIN_BUILD state. The stored chains may then be executed, and re-executed as many times as the application desires. When or if any optimization occurs is implementation-dependent.

4.4.6.1 Chain construction

Once a chain identifier has been allocated by invoking the *allocate_chain* mechanism, a chain can be constructed, executed and manipulated using the following control mechanisms:

chain_abort
chain_begin
chain_delete
chain_end
chain_execute
chain_reload

The application starts a new chain definition by performing *chain_begin*, which sets PIKS to the CHAIN_BUILD state.

In the `CHAIN_BUILD` state, all subsequent PIKS calls for elements which support chaining will simply have their functions added to the chain, instead of being executed. The chain definition is completed when a `chain_end` is performed, returning the PIKS implementation to its previous state. While PIKS is in the `CHAIN_BUILD` state, new operators are not placed on the asynchronous execution queue, but operations already on the queue continue to be executed. There is an additional requirement that the building of chains must be performed outside of chaining. Although chains may execute chains, they cannot build chains. Not allowing an executing chain to build a chain avoids confusion about alterations of program synchronization.

NOTE Image and non-image data objects are not overtly copied as a result of the invocation of any chaining mechanism.

Only one chain may be in the `CHAIN_BUILD` state at any time. Each chain must be completed (or closed) by invoking `chain_end` prior to starting another chain. Closing a chain is required prior to calling a non-chainable PIKS functional element, otherwise an error will be issued and the function will be ignored. Once closed, a chain may not be reopened, added to or modified, except as allowed by the `chain_reload` function, which is described below.

Not all PIKS elements can be included in a chain. The Element Functionality list associated with each element indicates by “yes” or “no” if an element is chainable without virtual registers. Virtual registers are defined in 4.4.7. Some PIKS elements can be included in chains if and only if their outputs are directed to virtual registers. Elements that require the use of virtual registers during chaining are denoted as “vreg” under the chainable designation.

Only PIKS elements can be included in a chain and not other functions supplied by a programming language. An implementation need not report invalid attempts to do so, though such attempts will probably result in invalid results.

Figure 9 illustrates a few exemplary chains supportable by PIKS. The linear chain consists of a simple sequence of PIKS elements. The multi-thread chain consists of several interrelated sequences, each containing PIKS elements. Each of the multi-thread chains may be executed in parallel to produce a common result. The use of virtual registers as chained elements allows synchronization. Single threaded chaining implementations produce the same result, by blocking on mutually required resources held in virtual registers.

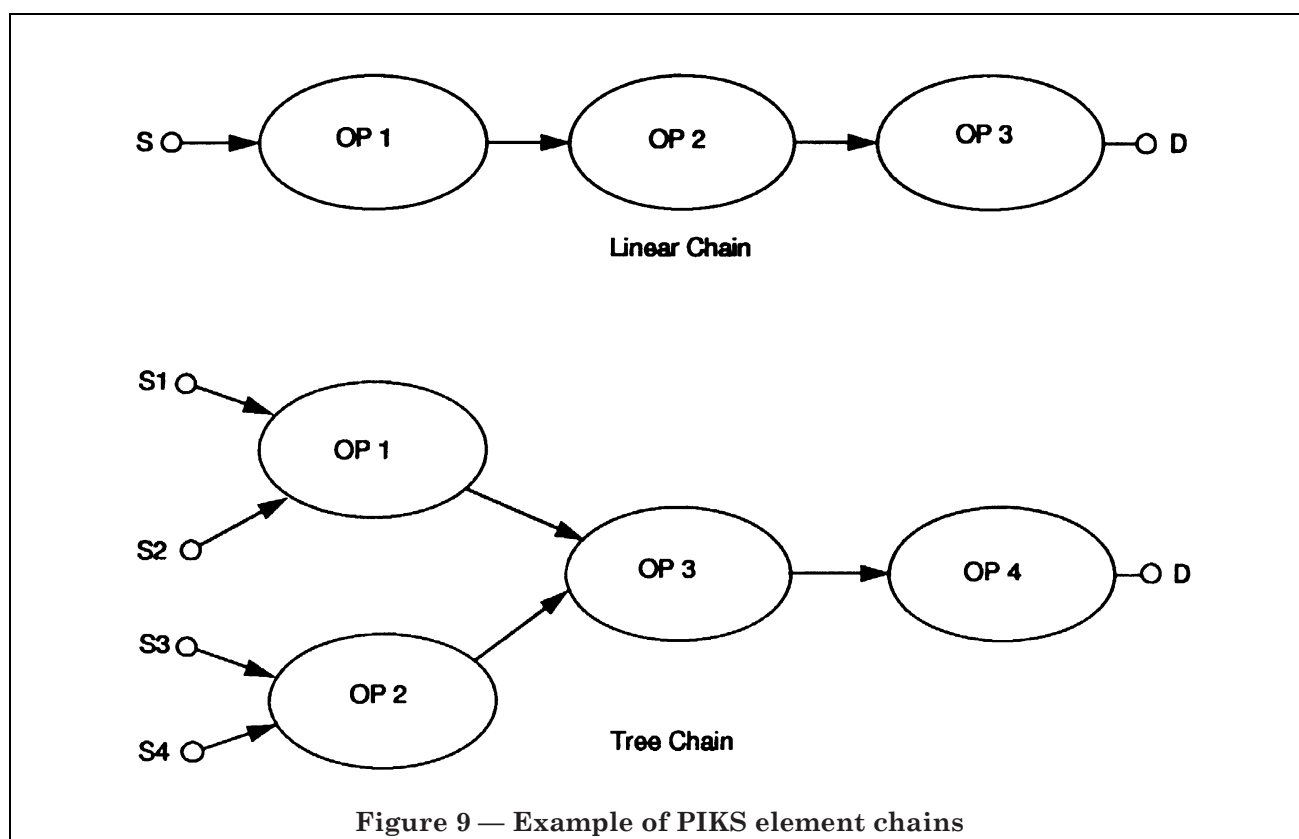
Chain reconstruction continues until a `chain_end` is performed. PIKS then reverts to its previous state. Invoking `chain_end` causes the chain to be stored and optimization may occur within the implementation. A chain is executed by invoking `chain_execute`, with the chain identifier of a completed and closed chain. In asynchronous mode, the `chain_execute` element is put onto the Asynchronous Execution Queue and is treated in the same way as any other PIKS element in that queue.

4.4.6.2 Chain execution

A `chain_execute` element may form part of a chain. The chain identifier of the embedded `chain_execute` may reference an undefined chain identifier (one which has been allocated but not yet defined by a `chain_begin/chain_end` sequence) at the time it is entered into a chain. However, by the time a chain executes, any embedded chains must be defined; otherwise, an error will result. In synchronous mode, chain elements are executed in the order they are placed in the chain; embedded chains act as subroutines. In asynchronous mode, an implementation is free to reorder or execute PIKS elements in parallel as long as the result seen by the application is the same as if the chain were executed in synchronous mode.

The number of chains which may be nested or initiated from within other chains is not limited. However, the number of chains, and the total number of chain elements stored may be limited by an implementation. Likewise the number of simultaneous `chain_execute` threads supported by an implementation may be limited. Information on the implementation limitations for chaining are provided by the `inquire_piks_implementation` element. Regardless of the implementation of chaining, the results provided by a one-thread implementation shall be the same as that provided by an implementation having an unbounded number of threads, except for the time required to complete the execution of all elements.

More than one instance of a particular chain may be in execution at a time. Each will carry its own state. However, this opens up the possibility of multiple instances of the same chain simultaneously operating on the same image resources. It is the application's responsibility to assure that no particular chain is being executed more than once if that is not intended for purposes of recursion. It is likewise the application's responsibility to assure that the set of chains executed do not form an infinite loop, such as would occur if the set contains a circular definition, such as would be the case if chain A executes chain B and chain B executes chain A.



Chain execution may be either synchronous or asynchronous according to the global mode. In asynchronous mode, *chain_execute* may return to the application immediately. PIKS provides a mechanism for an application to determine the state of execution of a particular chain, using the *inquire_chain_status* mechanism.

During chain execution, PIKS elements use the parameters and data referenced at the time the chain was built. PIKS provides a means for any PIKS element embedded in a chain to reference new parameters or data after the chain build has been completed, and even after the chain has been executed. One method of changing data and/or parameter values is accomplished by using the *chain_reload* element prior to execution or re-execution of a completed chain. Another method is provided by the use of PIKS virtual registers as element parameters. This latter method is explained in detail in subsequent sections.

PIKS system mechanisms which allocate and deallocate PIKS data objects, such as images, may be chained. A PIKS data object which is allocated and deallocated within a chain can be considered a temporary data object whose value is not made available to the application. This provides a means of producing and controlling temporary data objects during chain execution.

The *chain_reload* mechanism allows chain elements to be modified. The mechanism *chain_reload* can access any element built into a chain by specifying the chain identifier and the position of the element in the chain. The first element in the chain is identified as element 1 and the n-th element as element n. To prevent excessive complications and possible behaviour problems, *chain_reload* mandates that the replacing element be exactly the same type (function name) as the replaced element. Only the parameters and data associated with the two elements may be different.

Additional chain elements may be reloaded by accessing them one at a time. The new values established by *chain_reload* will be used in all subsequent executions of the chain, or until the *chain_delete* mechanism destroys the entire stored chain. If extensive modifications are to be made to a chain, it may be easier for the application to invoke *chain_delete* to destroy the stored chain. Then a replacement chain may be built in the same fashion as originally performed, using the new variables.

Chain execution terminates when a fatal error occurs during the execution of any PIKS element in a chain. The PIKS Error Handler and Error Logger are invoked when an error is detected, just as is the case for non-chained elements. The *inquire_chain_status*, and other inquiry functions, can be used to query the various state parameters to determine the location and type of error that has occurred.

As described in 4.4.7, additional functionality is provided by the use of PIKS virtual registers to hold temporary results or to signal chain events. PIKS provides a method for specifying that any output parameter may be a temporary object, to be used only during the execution of a particular chain, and not to be returned directly to the application. These temporary objects are defined as the content of virtual registers, (VREGs). VREG values may also be used to hold parameters and data for chain-embedded PIKS elements. Because VREGs can contain any valid PIKS data type, these registers can also be used to communicate chain element outputs to other chains or to the application. Iteration, “test and branch,” and “wait on completion” programming constructs are also provided. In addition to carrying either data or parameters, the VREGs allow the application to initiate a chain execution sequence to another chain execution or to signal the application. This two way communication may be used to signal progress and status or to alter the sequence of chain execution. One obvious use is to inform the application that data produced at some intermediate point in the chain is now valid. Details on the use of VREGs within any asynchronous or chained operation are provided in the next clause.

The *chain_abort* mechanism allows an application to prematurely terminate execution of a particular chain. This is useful for error recovery, in response to unsatisfactory partial results obtained from virtual registers, or when the user signals that a change of process is desired. This mechanism only affects the chain to be aborted. Other dependent chains could end up in a blocked state as a result. It is possible to circumvent this hazard by specifying a *chain_abort* for all dependent chains; these aborts may even be contained in a chain set up explicitly for that purpose. Note also that the PIKS global *abort_asynchronous_execution* mechanism (rather than *chain_abort*) will kill any and all chains as well as any asynchronous operations, either on-going or waiting in the Asynchronous Execution Queue.

4.4.7 Virtual register control

PIKS provides a set of virtual registers, or VREGs, to support application control for chaining and asynchronous processing. The VREGs are specific for each of the valid PIKS data types and are bound to a particular data type when they are created. The use of virtual registers is primarily in the context of chaining control, but does not preclude other uses, such as in asynchronous communication and control within PIKS. Virtual registers may also be used in sequential, non-chained PIKS programs.

PIKS virtual registers are manipulated with the following mechanisms:

- vreg_alter*
- vreg_clear*
- vreg_conditional*
- vreg_copy*
- vreg_create*
- vreg_get*
- vreg_set*
- vreg_wait*

Virtual registers are provided for all PIKS data types and act as virtual storage locations within PIKS. There is a complete set of virtual registers for each of the PIKS data types. Appropriate virtual registers can be used as parameters for any PIKS functional element. This allows an application to specify operational results to be held in the context of the PIKS environment, alleviating the need for PIKS to return results to an application, and forcing a synchronization. Virtual registers can also be used to control program flow within chained operations by auditing the progress of asynchronous operations, and through multi-threaded synchronization and resource monitoring. Because of the special control nature of VREGs, an “immediate-mode” parameter is associated with each of them, allowing for instant or delayed execution under the asynchronous PIKS mode.

Virtual registers in PIKS have three associated fields: status flag, data type, and data value. The first field, a status flag, indicates whether the register has been “set,” VREG_SET, or “cleared,” VREG_CLEARED. The “value” contains a data value appropriate for the data type of register. Allocation of virtual registers is performed by the *allocate_vreg* mechanism, which supplies PIKS with the appropriate data type to associate with the VREG. A VREG identifier is returned to the application. If the status flag has not been set, the value field is not meaningful. VREGs are initially in the VREG_CLEARED state when they are created with an *allocate_vreg* mechanism. Only VREG operations which set a VREG can change the state to VREG_SET. A *vreg_clear* mechanism allows the VREG to be re-initialized to the VREG_CLEARED state. All associated data is considered meaningless as a result, but the virtual register itself is not destroyed, as it would be with a *vreg_delete*.

The use of VREGs depends on the application requirements to manipulate these symbolic variables. The flexibility achieved by symbolic, or token, specification of PIKS functional element parameters is especially useful in the context of element chaining. There are three main uses for VREGs in the context of chaining operations: communication, synchronization, and process control. Although the VREGs can be used outside of chaining, the remainder of this clause will primarily exemplify their use in chaining.

4.4.7.1 Virtual registers for storage of PIKS temporary variables

Virtual registers may be used to store the temporary results of chained elements. Temporary results can be input to any chained element in one of two ways. First, by substituting an invocation of the *vreg_get* mechanism in the parameter list of the chained element at the position of the desired parameter, if and only if the VREG contains an internal identifier. Second, by substituting the VREG identifier itself in the parameter list of the chained element at the position of the desired parameter.

The VREG mechanisms allow an application to use internal PIKS variables as either input or output parameters. The output of a PIKS element can be returned in a virtual register and then used as an input parameter to another PIKS element. An important distinction between VREG variables and application variables is that VREGs are local to PIKS, and not returned to the application unless explicitly requested through a *vreg_get* mechanism. This means that the virtual register variables are insulated from inadvertent alteration by an application. A second difference is that virtual register variables allow for synchronization, unlike program variables. This feature, which can be very valuable for multi-chained asynchronous operations, is discussed in the next clause.

Virtual registers provide an alternative to the *chain_reload* mechanism because they allow for the alteration of chained element parameters, and therefore can be automatically altered prior to or during execution of a chain. Virtual registers allow chained elements to access previously computed results. It can be of significant advantage to use VREGs as an autonomous mechanism for chains to pass results without supervision and synchronization required from an application.

In addition to PIKS functional elements setting virtual registers as output parameters, an application or a chain can also set virtual registers with a *vreg_set* mechanism. The use of the *vreg_clear* mechanism allows either a chain or an application to re-initialize the virtual registers. The *vreg_copy* mechanism allows the contents of a virtual register to be replicated to other virtual registers. This can be useful for multiple chains sharing a partial result independently.

Persistence of virtual registers is for the duration of the PIKS_OPEN state, unless explicitly deleted using the *vreg_delete* mechanism.

4.4.7.2 Virtual registers for asynchronous control

One use of virtual registers is to coordinate synchronization on asynchronous PIKS operations. The completion of a PIKS event can be signaled by setting a virtual register to a value. This can be either explicit, with a call to *vreg_set*, or implicit, through use of a VREG as an output parameter of a PIKS chainable or asynchronous functional element. Synchronization can be achieved by having the application, or another chain, wait on the VREG_SET event using the *vreg_wait* mechanism. This call will block execution of the caller until the associated VREG is set to the value indicated in the *vreg_wait* call. The value waited on can be either specific to the VREG data type, or general to any value being set in the VREG. When a VREG is an input parameter to a PIKS element in the asynchronous of chaining mode, an implicit “wait until VREG_SET” occurs. The *vreg_clear* mechanism resets the state of the virtual register to VREG_CLEARED, for further synchronization usage. The use of *vreg_set*, *vreg_wait*, and *vreg_clear* allows an application to control the synchronization of chains, which depend on other chains for input of computed variables or data objects.

4.4.7.3 Virtual registers for chain iteration and conditional execution

Additional control over chain operations is supplied by the *vreg_conditional* mechanism, which provides a “test and branch” chaining control mechanism. The *vreg_conditional* mechanism allows program specification of a condition to be met, which will cause a branch to another executable chain. The test is for the value of a virtual register to be within a specified value range; the branch can be taken on either a TRUE or a FALSE result. Additional specification of process control is provided by *vreg_conditional* and indicates whether the chain that executes the branch is to be returned to after completion of the branch. Multiple occurrences of the *vreg_conditional* mechanism within a chain, with different value ranges and conditions, can duplicate “if-else” and “case” program constructs. Iteration can be specified with the use of *vreg_alter* to change (by increment or decrement) the value of a virtual register, and then the *vreg_conditional* mechanism can be used to test the result and branch back to the beginning of the chain.

4.4.7.4 Virtual registers for auditing asynchronous state

The *vreg_get* mechanism allows the application to query the state of a virtual register. Thus, state information on the progress of asynchronous or chained processing can be obtained. Another use for *vreg_get* is to return temporary values at any stage within a chain. This feature can be used for a number of purposes. One purpose would be to use VREGs to indicate results of a particular stage in a chain selectively, especially for debugging purposes. Setting of virtual registers as outputs either within a chain or for asynchronous operations can also be used as an audit trail to indicate progress on operations. The *vreg_get* mechanism allows an application to inquire the data type, value, and status (VREG_SET or VREG_CLEARED) of a particular virtual register.

NOTE 1 The use of chaining and virtual registers together can provide algorithmic specifications within PIKS. Such programming extensions, however, must be constructed out of existing PIKS functional elements. Thus, chaining, as an extension mechanism, is limited to macro capabilities on top of PIKS functional capabilities.

NOTE 2 PIKS does not specify process priority or interrupt mechanisms to allow program control over asynchronous execution. Therefore, when using chaining and virtual registers, it is important to specify inter-process dependencies in unambiguous ways, which do not assume anything beyond the implicit sequence of operations within a single chain. Race conditions, non-terminal iteration, conflicting processing states, and other programming defects may result from improper usage of these chaining mechanisms. The *chain_abort* mechanism can insulate an application from some of these malignancies, but ultimately it is the responsibility of the application to insure programming integrity.

4.4.8 Global element control

PIKS shall provide mechanisms for global operator control within the operator model. The operator control functions are:

ENGAGE ROI CONTROL — This is a switch mechanism that engages the standard ROI control functionality of the operator model when the switch is ON. When the switch is OFF, the ROI control functionality is disabled.

ENGAGE ROI PROCESSING — This is a switch mechanism that engages the processing of ROI control objects by certain operators when the switch is ON. When the switch is OFF, a source ROI control object is not modified by an operator.

ENGAGE MATCH POINT — This is a switch mechanism that engages the match point functionality of the operator model when the switch is ON. When the switch is OFF, the match point functionality is disabled.

ENGAGE INDEX ASSIGNMENT — This is a switch mechanism that engages the index assignment and reassignment functionality of the operator model when the switch is ON. When the switch is OFF, the index assignment and reassignment functionality is disabled.

ENGAGE ASYNCHRONOUS — This is a switch mechanism that engages asynchronous operation when the switch is ON. When the switch is OFF, operation becomes synchronous.

IMAGE RESAMPLE SELECT — This is an option selector that specifies which resampling function will be used in image resampling when match point translation or relevant operator processing is performed in the operator model.

ROI RESAMPLE SELECT — This is an option selector that specifies which resampling function will be used in ROI resampling when match point translation or relevant operator processing is performed in the operator model.

The image and ROI global resampling options are:

- 1 support 1 constant (nearest neighbour)
- 2 support 2 linear
- 3 support 2 lookup table
- 4 support 4 truncated sinc
- 5 support 4 cubic B-spline
- 6 support 4 cubic convolution
- 7 support 4 Lagrangian
- 8 support 4 lookup table
- 9 support 8 truncated sinc
- 10 support 8 lookup table

Additions to the global resampling options are possible through the extension method described in 5.6.

The ENGAGE ROI CONTROL switch overrides any image attribute references to bound ROIs. When the switch is OFF, the operator model acts in the same manner as it would if all bound ROIs were TRUE at all pixels.

The ENGAGE MATCH POINT switch overrides any image attribute references to match points. When the switch is OFF, the operator model acts in the same manner as it would if all match points were set to the origin.

The ENGAGE INDEX ASSIGNMENT switch overrides any image attribute references to an index assignment table. When the switch is OFF, the operator model acts as if: x was assigned to j; y was assigned to k; z was assigned to 1; t was assigned to m; and b was assigned to n.

Operators that provide resample options shall override the global resampling option.

Clause 7 contains specifications of the global operator control mechanisms. Annex H defines the global resampling options.

4.4.9 Composite image management

This clause specifies aggregation and deaggregation image object composition mechanisms. They are higher-level adjuncts to basic PIKS image objects. The purpose of these mechanisms is to provide an application with a means of handling ensembles of related images. These mechanisms permit the composition and decomposition of aggregates of images, such as lists, arrays, and records of images, whose pixels are BD, ND, SD, RD, CD data types, as defined in ISO/IEC 12087-1. These image aggregates are directly specialized from the compound image data types defined in ISO/IEC 12087-1 to comply with the five-dimensional memory model underlying the PIKS architecture. They provide for storage and retrieval of image identifiers.

NOTE PIKS does not provide operators that perform operations directly on composite images. However, PIKS does provide operators that perform operations on PIKS image objects that are referenced by identifiers in composite image aggregates.

There are three types of composite image identifier objects:

- composite image identifier arrays
- composite image identifier lists
- composite image identifier records

The following clauses describe each of these types of composite image objects.

4.4.9.1 Composite image identifier arrays

Composite image arrays are one-dimensional, fixed-length arrays of image identifiers, which provide references to PIKS image objects that have been created using PIKS system management mechanisms. The individual composite image identifier array elements can be accessed by a non-negative integer index. Prior to any access, composite image identifier arrays are created by invoking the mechanism.

allocate_composite_identifier_array

which creates an array of image identifiers, all of which are initialised to NULL value upon creation. For storage, image identifiers can be arbitrarily accessed by index by using the mechanism

composite_identifier_array_put

For retrieval, the corresponding mechanism is

composite_identifier_array_get

The equality of two composite image arrays may be tested by a composite image mechanism

composite_identifier_array_equal

which determines if both composite image identifier arrays store identical sequences of image identifiers.

If no longer needed, composite image identifier arrays can be explicitly deleted by invoking the general system management mechanism

deallocate_data_object

Clause 7 specifies the composite image identifier array mechanisms.

The code for a composite image identifier array data object is ID_ARRAY.

4.4.9.2 Composite image identifier lists

Composite image identifier lists are sequences of image identifiers, which provide references to PIKS image objects. The ordering of a list is imposed by the order in which list elements have been appended to the tail of the list. Prior to any access, composite image identifier lists are created by invoking the mechanism

allocate_composite_identifier_list

Upon creation, the list is empty. Image identifiers can be inserted in a composite image identifier list by the mechanism

composite_identifier_list_insert

Image identifiers can be removed from the tail of a composite image list by the mechanism

composite_identifier_list_remove

Individual images referenced by their identifiers in a composite image identifier list can be accessed by moving an internal list pointer through relations between the list elements. These relations are defined by the order in which images have been inserted in the composite image identifier list. The image identifiers can be retrieved by the mechanism.

composite_identifier_list_get

The following mechanism determines if a composite image identifier list is empty.

composite_identifier_list_empty

The equality of two composite image identifier lists may be tested by a composite image mechanism

composite_identifier_list_equal

which determines if both lists contain identical sequences of image identifiers.

If no longer needed, composite image identifier lists can be explicitly deleted by invoking the general mechanism

deallocate_data_object

Clause 7 specifies the composite image identifier list mechanisms.

The code for a composite image identifier list data object is ID_LIST.

4.4.9.3 Composite image identifier records

Composite image records are records of image identifiers, which provide references to PIKS image objects. A composite image identifier record is an aggregate of named attributes, which provides access by name to the PIKS image identifiers it stores. Prior to using a composite image identifier record, it is created by invoking the mechanism

allocate_composite_identifier_record

which specifies character string attribute names of the components of the record. All access to image identifiers stored in the composite image identifier record shall be by these attribute names.

An image identifier can be stored in an image identifier record by invoking the mechanism

composite_identifier_record_put

It can be retrieved by the mechanism

composite_identifier_record_get

The equality of two composite image identifier records may be tested by a composite image mechanism

composite_identifier_record_equal

which determines if both records contain identical names

If no longer needed, composite image identifier records can be explicitly deleted by invoking the general mechanism

deallocate_data_object

Clause 7 specifies the composite image identifier record mechanisms.

The code for a composite image identifier record data object is ID_RECORD.

NOTE 1 Composite image identifier arrays, lists, and records provide an efficient means of manipulating large ensembles of related physical images. Physical images can be “virtually stored” between several image identifier arrays, lists, and records, without duplicating bulk pixel data, by copying the respective image identifiers.

NOTE 2 Many of the composite identifier array, list, and record mechanisms can be used to store non-image object identifiers. It is the responsibility of the application to insure consistency between these mechanisms and other PIKS elements.

4.4.10 PIKS error handling

For each PIKS element, error situations are specified, any of which will cause the PIKS error handling function to be called. Every PIKS implementation shall support this form of error checking. The error handling function provides an interface between PIKS and an application. The error handling function, if provided by an application, may interpret the information about the error and may store data in a data area for subsequent interpretation by the application after returning from the PIKS element that caused the error.

NOTE Inquiry utilities do not generate error indications.

The PIKS error handling strategy is derived from the following error classification:

- I. errors resulting in a precisely defined reaction
- II. errors resulting in an attempt to save the results of previous operations
- III. errors that cause unpredictable results, including the loss of information

PIKS recognizes three situations in which errors are detected:

- A. error detected in PIKS functions
- B. error detected in functions called from PIKS (driver functions, operating system functions)
- C. error detected in other areas of the application program

If errors are detected outside PIKS (situation C), either the application may regain control over the execution, or program execution will be terminated abnormally. In the latter case, results are unpredictable (class III), and, in the worst situation, all processed image information produced thus far may be lost. However, if the application obtains control, it may attempt to close PIKS properly, or at least attempt an emergency closure by calling the *close_piks_emergency* mechanism. Similarly, if an error occurs in the element called by PIKS, and control is not returned properly to PIKS, the effects will be unpredictable.

Finally, all errors that are listed explicitly as part of the definition of PIKS elements belong to class I. Either these errors are detected with PIKS itself (situation A), or an element called from PIKS has returned control to the corresponding PIKS element with appropriate error information (situation B). In all of the these class I cases, PIKS calls the *error_handler* mechanism. If a PIKS element is invoked with more than one error condition applicable, at least the first error is reported.

The application may either provide its own error handling function, or may use the one provided as part of PIKS. Any error handling function accepts the following information from PIKS.

- identification of the error condition
- identification of the PIKS element that called the error handling function
- error file

The error handling function provided by PIKS calls for an *error_logger* mechanism using the same set of parameters. The latter performs the following actions:

- writes an error message and PIKS element identification on the error file
- returns to the calling element

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 is an example of an error handling function supplied by an application:

```
ERROR HANDLER (error number, identification of PIKS element, indication of
    asynchronous or synchronous state, identification of chain if one exists, error file);
    Interpret PIKS element and error identification in order to select the following cases:
CASE "special treatment":
    Interpret error parameters as passed from PIKS;
    Store information for the application in application-supplied data area;
    Return to calling PIKS element
CASE "standard treatment":
    Call error logging function with all of the above parameters;
    Return to calling PIKS element
END
```

A PIKS application may suppress transfer of control to the *error_handler* mechanism by setting the "error reporting mode" in the PIKS error state list to OFF. The default value for the "error" reporting mode is ON.

All PIKS elements takes these steps after detecting an error condition:

```
set error state to ON
call error_handler function with appropriate parameters
set error state to OFF
do built-in error reaction
```

At least one error detected is reported.

An error handling function, supplied by an application, has access to the set of PIKS state variables. However, no modification of PIKS state is possible during error handling. Only the *error_logger*, *close_piks_emergency*, and PIKS inquiry mechanisms may be called by the error handling function, which is supplied by the application. This is achieved by setting the error state to ON prior to calling the error handling function from PIKS, and by setting the error state to OFF afterwards.

Error numbers for error originating in PIKS elements can be found in clause 7 under each PIKS element. Annex J contains a compilation of PIKS error codes. The following errors numbers are reserved:

- Negative error numbers are reserved for implementation errors.
- Unused error number less than 2000 are reserved for future standardization.
- Error numbers 2000 to 3999 are reserved for language bindings.
- Error numbers greater than or equal to 4000 are reserved for registration.

Error numbers are registered in the ISO International Register of Graphical Items, which is maintained by the Registration Authority. When an error has been approved by ISO, the error number will be assigned by the Registration Authority.

When multiple errors are possible for a PIKS element, the order in which they are listed in clause 7 is the order in which an implementation should test for and report any error conditions.

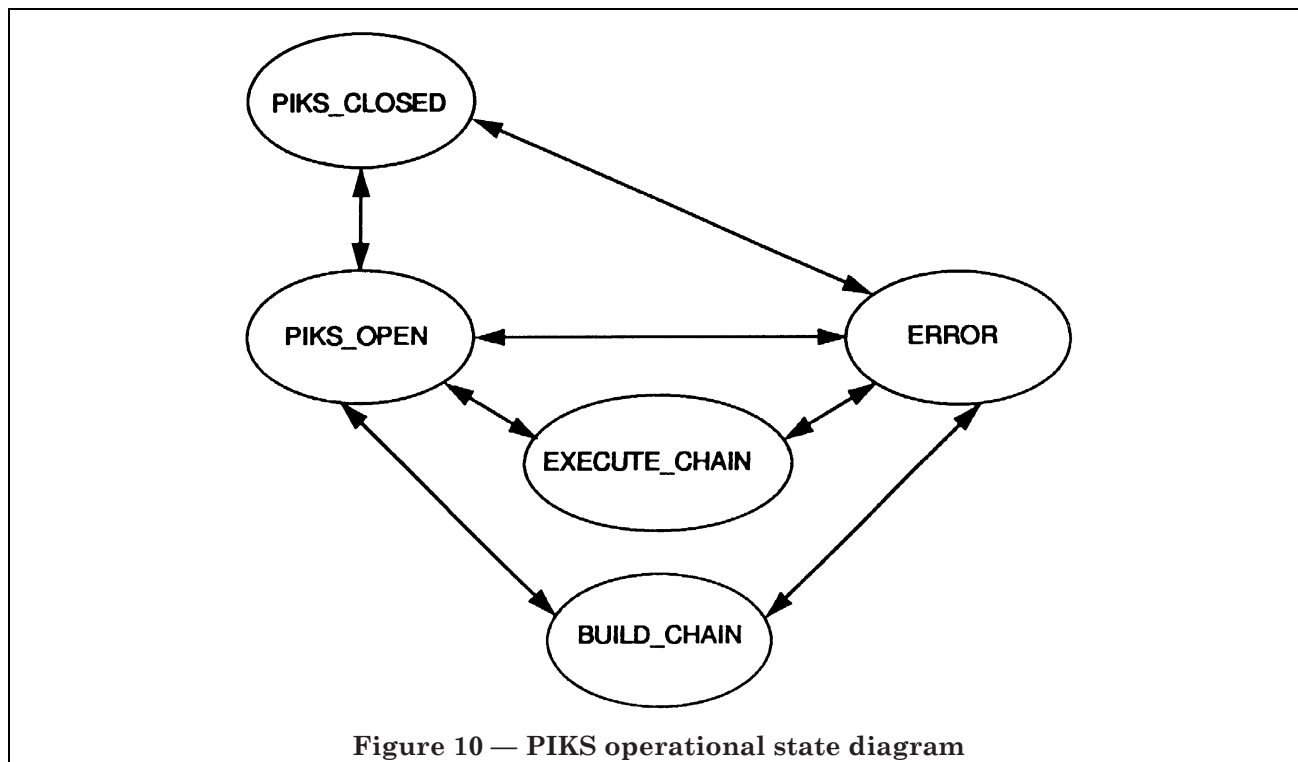
4.4.11 PIKS operational states

PIKS functions in a number of operational states. Each state supports certain allowable capability as well as certain restrictions. These states are designed to provide integrity and protection for both PIKS and applications using PIKS.

Figure 10 shows the relationship of the PIKS operational states.

In some instances, PIKS will change back and forth among states, as in the case of ERROR, EXECUTE_CHAIN, and BUILD_CHAIN, which interleave with PIKS_OPEN.

Individual PIKS elements declare the PIKS state or states that allow their execution. If an element is invoked when PIKS is in any state other than an allowed state, an error will result.



A description of the various PIKS states follows.

State: ERROR

This state is entered when an error occurs during the execution of a PIKS element.

State: PIKS_CLOSED

This state is in effect prior to successful execution of the *open_piks* mechanism and following a successful invocation of the *close_piks* or *close_piks_emergency* mechanisms. In this state, it is permissible that PIKS global storage (including state tables) have not yet been allocated, and that no default values or other values has been established. Therefore, execution of any inquiry system management mechanism will result in an “inquiry data not available” response. Execution of any PIKS element, other than an inquiry or *open_piks* mechanism, will result in an error.

State: PIKS_OPEN

This state is established after successful execution of the *open_piks* mechanism after all internal tables have been established and have been given either default values or other values. The only elements that cannot be executed in this state are the *open_piks* and *chain_end* mechanisms. A successful execution of the *close_piks* mechanism puts PIKS back into the PIKS_CLOSED state.

State: BUILD_CHAIN

In this state, the operation code and parameters of all elements invoked are put onto a specified chain prior to returning control to the application. The parameters of the elements of the chain may not have been fully examined for validity, but they may have undergone a preliminary analysis for correctness.

The BUILD_CHAIN state is entered from the PIKS_OPEN state by execution of the *chain_begin* mechanism. The BUILD_CHAIN state is terminated, returning back to the PIKS_OPEN state, by execution of the *chain_end* mechanism. Once completed, the chain may be executed by invoking the *chain_execute* mechanism.

The specification for each PIKS element indicates whether it can be chained. Any PIKS element, which may be invoked in the PIKS_OPEN state, may be invoked in the BUILD_CHAIN state provided that it is identified as chainable.

If an error is detected during the building of a chain, the ERROR HANDLER will be invoked. PIKS will be put into the ERROR state, and the condition will be handled prior to the normal return to the application.

State: EXECUTE_CHAIN

This PIKS state is in effect when the PIKS elements, which are part of a chain, are being interpreted and executed. The EXECUTE_CHAIN state is entered when the *chain_execute* mechanism is invoked. When the *chain_execute* mechanism returns control to the application, the state returns to PIKS_OPEN.

Normally, PIKS elements are invoked by an application in the PIKS_OPEN state. When these elements are executed as part of a chain, PIKS is placed into the EXECUTE_CHAIN state in order to make more information available if an error occurs.

When an error occurs, the current PIKS state is reported as part of the error information. If the error state is EXECUTE_CHAIN, the application can then inquire about additional information, such as which chain and which chain element caused the error.

PIKS allows PIKS elements and chains to be executed asynchronously. In addition, implementations may provide multiple execution threads to provide high performance PIKS environments. There are no PIKS facilities for multi-thread process handling, and the presence of such threads is apparent only in improved performance. However, if an error occurs during the execution of a PIKS element by an asynchronous control thread, some PIKS condition needs to be reported. Thus, if an error occurs during the asynchronous execution of a PIKS element, then one of two conditions is reported as the error state. These two conditions are: ASYNCHRONOUS_EXECUTION and ASYNCHRONOUS_EXECUTE_CHAIN. The error condition is ASYNCHRONOUS_EXECUTE_CHAIN if the PIKS element that caused the error is part of a chain; if the PIKS element was placed on the Asynchronous Execution Queue individually and not as part of a chain, then the condition reported as the error state is ASYNCHRONOUS_EXECUTION.

4.5 PIKS utilities

This clause describes the general properties of PIKS utilities. These utilities are specified in clause 7.

4.5.1 Inquiry

PIKS provides a class of inquiry functions whose purpose is to supply information to the application about the operational status of PIKS internal functionality. Inquiry utilities do not return errors. If an inquiry utility is unable to ascertain the requested information, a data validity flag is set to indicate “data unavailable.”

4.5.2 Import and export

PIKS provides capabilities to import and export image and non-image data objects between PIKS and an application and between PIKS and the Image Interchange Facility. Figure 11 describes the flow of data objects between PIKS and the application and between PIKS and the IIF.

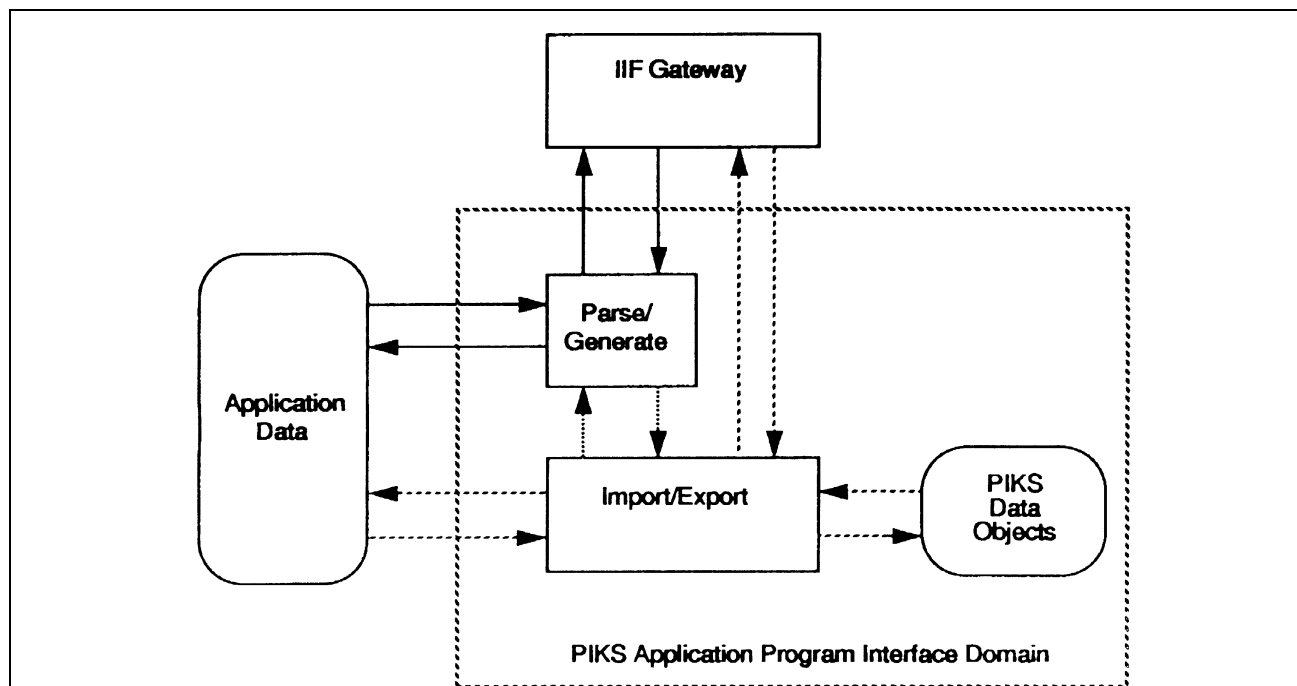


Figure 11 — PIKS interface to the IIF gateway and an application