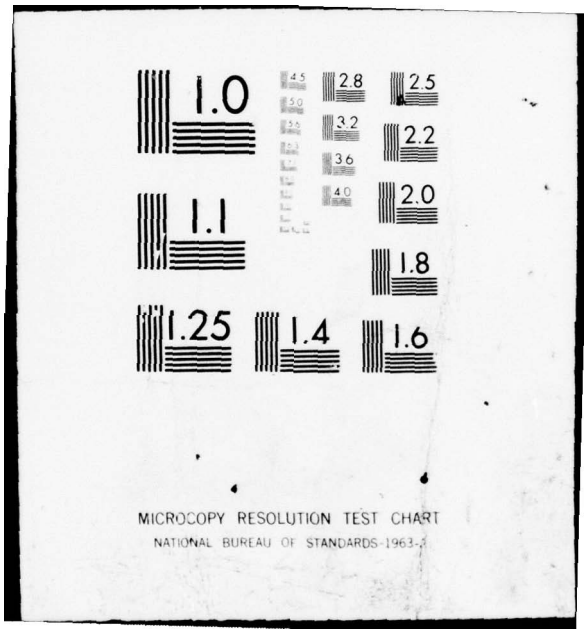
AD-A069 823 ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE F/G 5/8
A FLEXIBLE DATA STRUCTURE FOR ACCESSORY IMAGE INFORMATION.(U)
MAY 79 P G SELFRIDGE N00014-78-C-0164
UNCLASSIFIED TR-45 NL

| OF |
AD
A069823

END
DATE
FILMED
7-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

電腦科學

LEVEL IV

(4)

Department of Computer Science
University of Rochester
Rochester, New York 14627

A FLEXIBLE DATA STRUCTURE FOR ACCESSORY IMAGE INFORMATION.

Peter G. Selfridge
Department of Computer Science
University of Rochester
Rochester, NY 14627

TR-45
May 1979
31 p.
Technical rept.

Abstract

A simple and powerful system for keeping track of derived images and accessory information about images has been designed and implemented. It takes advantage of extra room in image file headers to store information in the form of NAME-TYPE-VALUE (NTV) triples. The NAME is a string name for the triple, the TYPE an integer designating the type, and thus the internal format, of the VALUE. This report describes the format and use of NTV information, including design decisions, current uses, and user software available. In addition, future uses are described, including using NTV triples to specify relations between images, and images and other kinds of information in a general and useful way.

The research reported in this paper was supported in part by the Defense Advanced Research Projects Agency, monitored by the ONR under Contract No. N00014-78-C-0164.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR45	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: A Flexible Data Structure for Accessory Image Information	5. TYPE OF REPORT & PERIOD COVERED: technical report
	6. PERFORMING ORG. REPORT NUMBER: 45
7. AUTHOR(s): P.G. Selfridge	8. CONTRACT OR GRANT NUMBER(s): N00014-78-C-0164
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Rochester, Computer Science Department, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: May 1979
	13. NUMBER OF PAGES: 29
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. *(of this report)*: unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

image processing
associative data structures
image file format

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A simple and powerful system for keeping track of derived images and accessory information about images has been designed & implemented. It takes advantage of extra room in image file headers to store information in the form of NAME-TYPE-VALUE triples. The NAME is a string name for the triple, the TYPE an integer designating the type, and thus the internal format, of the VALUE. This report describes the format and use of NTV information, including design decisions, current uses, and user software available. Future uses are also described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

I. Introduction:

Research in computer vision and image processing necessitates storing and accessing digitized images. In the course of research, many derived images can be produced. In addition, many operations are performed on images which are computationally expensive. Keeping track of image information, including image statistics, relations between images and between images and other data structures is a problem in a large image data base.

A stored image is essentially a list of pixels, along with information about how to use those pixels to re-create the image. The latter information is often stored in an image header. We have extended the header to include additional information in the form of NAME-TYPE-VALUE (NTV) triples or slots. The NAME is a string name for the triple, the TYPE an integer designating the type, and thus the internal format, of the VALUE. Various types can be used, including the usual INTEGER (in 16 and 36 bit lengths), REAL (again in two forms), BOOLEAN, STRING, and lists of these types.

NTV triples can be used for storing comments, including image history, how and from where a particular image was derived, and image statistics such as the mean, minimum and maximum pixel values. In addition, NTV triples can be used as file pointers to other derived images.

This report describes the NTV system, including design decisions, relation to other research efforts in our department, current uses, and the variety of software available for adding, editing and listing NTV information in image headers. In addition, the use of NTV triples to store image information addresses several current problems in image management, including some of the problems of structuring image files in a useful way.

II. Problems in Image Management:

As research in image processing and computer vision progresses, problems in managing images and image information can be expected to proliferate. Image managment problems include keeping track of various kinds of image information, producing and perpetrating standard image file structure formats for increased portability, and the relatively un-explored issue of structuring images and image information to reflect useful relations between images. Putting intelligent structure into image data-bases could have far reaching effects in image accessability, both for people and programs. In addition, there is a great need for organization that would make it easier for programs to be run on and produce sets of images.

Image information can usefully be divided into three catagories. The first is information necessary to re-create the image from the list of pixels, such as the dimensions of

the image and packing information. Such "format descriptors" must allow one to access the [i,j]th pixel value, which may be a single number or a vector of numbers, in the case of multi-spectral images.

The second catagory is annotational information. This might include a descriptive title, creation date, conditions when taken, and various useful image statistics like the minimum, maximum and mean pixel value. What kinds of annotational information is useful depends on the particular image data base and how it is being used.

The third is a different kind of information: relational information, relating images to other images, or to other, perhaps symbolic, representations of the real world. Storing relational information involves, in some way, structuring the image data base. This can be done in a system seperate from the actual images, like a seperate relational data base, or with the images themselves with the use of file pointers between images and to other data structures.

The first two catagories of information have been addressed in reports of individual image processing laboratories (for example, see [Chang], [Haralick], [Hayes], [Lein], [Mckeown], [Tamura], and [Zobrist]) and have arisen in recent workshops ([WorkshopIEEE], [WorkshopNBS]). By and large, solutions have centered on fixed header formats, where each unit of information is allocated a space for a

number or code encoding the information. A program for accessing the information does so by absolute location in the header, and the information is of fixed length. This solution, while satisfactory for format descriptors, is inflexible, especially to changing user needs in the kinds of annotational information that is useful to associate with images. Fixed length strings, limiting the length of image titles, for example, are especially confining.

A related problem in image management is portability. In research in image processing and computer vision, there is a great need for the ability to apply algorithms to many images. Currently, there is an abundance of image processing techniques, especially "low-level" (non-semantic) algorithms: line finders, edge operators, region growers and other techniques abound. A demonstration of the robustness of the techniques, and some real comparison and analysis is lacking. This is partly due to the paucity of images, and the difficulty of tranfering images amoung different computers and different installations. The difficulties have persisted despite efforts to promote standards of various kinds ([WorkshopNBS]). (A clearing house approach is also being advocated ([Schmidt]); some of the same problems remain.)

The problems in transfering images are related to the problems of keeping track of image information mentioned above. Preserving image information of various kinds amoung

different computers and languages is a hard problem in general. Clearly, solutions to storing information should increase ease of portability by being general, straightforward, position-independent and self-describing, if possible.

The problem of structuring image files has been addressed primarily by proposing relational data-base schemes for images ([Chang], [Kunii], [Lein], [Mckeown], [Tamura]). The goal here is to be able to specify images by their characteristics, and to be able to search a data-base of images with certain characteristics using a relational query language. Usually, relational information is kept in data structures separate from the actual images. This clearly complicates the image portability issue. Relational data-bases represent an approach to image management at a higher level than that of the NTV system, although the two are not mutually exclusive.

The NTV system addresses these problems in several ways. Information of most kinds falls into this structure, and its generality allows the user to pick the level of information and primitives desired. The NTV concept is general enough to ease the problem of image portability; we currently share images with associated image information between a mainframe and our mini-computer network. The NTV system is both simple and extendable, and it easily allows links to other, more complicated data structures. While

clearly not as powerful or general as a data base management system, it retains significant flexibility, and could be easily linked to such a system. It seems to be a good compromise solution to some of the problems of intelligently structuring image files.

III. The NTV Package:

The NTV (Name-Type-Value) Package is a system that allows the user to store and access general image information in a flexible way. The unit of information is the NTV slot or triple, to be described shortly. Slots are stored in the image file header, and thus are always associated with images. The basic structure of an image file is shown in figure 1, and expanded in detail in Appendix I.

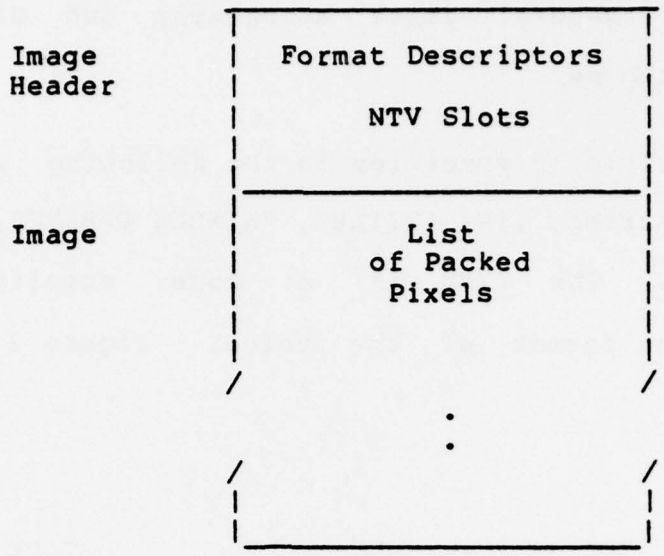


Figure 1: Rochester Image File Format.

Slots, as units of information, are accessed by a string NAME, not by absolute or even relative position in the header. The sub-unit of the NTV slot is an 8 bit byte, allowing some promise of generality to different machines.

The underlying concept of NTV information is that it describes a relation. For example, the TITLE of an image might be the string "TUMOR WINDOW FROM LUNG57". Part of its heritage comes from the LEAP concepts developed for the language SAIL ([Feldman]). In addition, it shares several concepts with the PLITS message based system designed at Rochester, including the actual internal format for the triples ([Low]) (Appendix I describes the internal format of NTV triples). Such overlap should prove more useful as our

research in general image management and distributed computing continues.

An NTV triple is specified in the following way. The NAME is a string, like "TITLE", "WINDOW PARAMS", or "MEAN PIXEL VALUE". The TYPE is a code specifying the representation format of the value. Figure 2 lists the current types:

TYPE	CODE
16 bit integer	INTEGER16
36 bit integer	INTEGER36
boolean	BOOLEAN!
PDP 10 real	REALPDP
ALTO real	REALALTO
string	STRING!
list of INTEGER16's	INTEGER16!LIST
list of INTEGER36's	INTEGR36!LIST
list of BOOLEAN's	BOOLEAN!LIST
list of PDP real's	REALPDP!LIST
list of ALTO real's	REALALTO!LIST

Figure 2: The types and type codes for NTV triples.

Two kinds of REAL's are used, one for our large computer (PDP-10) and the other for our mini-computers, which have a different word size and real number format. Type STRING indicates a character string of any length between 1 and 255. LIST TYPE's indicate that the value is a concatenation of a number of the primitive TYPE's. They can have any number of elements. A complete description of the

internal format of NTV slots is specified in Appendix I.

Several examples of NTV triples are shown in figure 3:

NAME	TYPE	VALUE
"WINDOW PARAMS"	INTEGER16!LIST	X1,Y1,X2,Y2
"TITLE"	STRING!	"LUNG WINDOW WITH TUMOR"
"MEAN"	REALPDP	70.2
"CONVOLUTION FILE 1"	STRING!	"CNVL1.RV"

Figure 3: Several example NTV Triples or Slots.

The following NTV primitives are available:

HDRIN - read in the header of an image file

HDROUT - write out the header

PUTNTV - put an NTV slot

GETNTV - get the slot TYPE and VALUE given its NAME

FETCHNTV - fetch the NAME, TYPE, VALUE of the nth slot

DELETE - delete a slot, given its name

INITNTV - initialize the NTV part, clear all slots

GETROOM - get the number of unused words in the header

Note that these primitives allow both random access by NAME (GETRV), and sequential access by slot order (FETCHNTV). The latter is convenient for listing the information contained in NTV slots. All primitives return a

status, which describes either success on the call or a variety of error conditions. A full characterization of all the primitives available is in Appendix II. Appendix III shows, for further clarification, a complete SAIL program utilizing the NTV Package to write TITLE slots.

The NTV Package includes two other programs that manipulate NTV information in image headers. NTVEDT is a comprehensive NTV editor. It allows one to read in an image header, access all NTV slots using the above primitives, and write the header out again. RVLIST is a program for listing the NTV slots given a file specification. The listing can go to the terminal or to a file. This allows listing and cataloging of image files given their slots information, and removes the ambiguitiy of the six-character file names demanded by the TOPS-10 operating system. Appendix IV shows a sample listing from RVLIST.

IV. Current Uses

Current uses of NTV information in image headers includes:

- general annotation, including automatic logging of image history
- memo functions
- hierachical region representation
- pointers to convolution files

- direct annotation of digitized images

General annotations include a short descriptive "TITLE" slot which is used in directory listing of image files, an "ORIGINAL" slot for recording the original image the current image was derived from, and annotations for processed or derived images. Again, the NTV structure allows users to choose their own annotations.

Memo functions are functions which automatically annotate the header with computed image information such as image statistics. Our STATS function is used to write the minimum, maximum, mean and standard deviation of the pixel values using agreed upon name conventions. Thus, our GET!STATS procedure first examines the header for the required slots, if the information is there, it is used. (This raises an important issue: the issue of quaranteeing that information is current. In general, this is very hard; all image management systems have this problem). Otherwise, STATS is called and the header is updated for next time. As a simple computation will show, this can result in very significant time savings (several orders of magnitude) when considering a large image.

NTV slots are used to represent regions. In this case, a region is represented in a region slot as a list of integers representing the seed coordinates and a threshold for a standard pixel-merging region grower ([Selfridge]). A

region slot represents objects in the image such as "ROAD5", or "OIL TANK 3". Another slot with the name "ROAD!REGIONS" contains string pointers to the region slots, as illustrated in figure 4. Appendix IV lists the slots of a file which are organized in this manner.

NAME	TYPE	VALUE
OIL!TANKS	STRING	"OILT1 OILT2"
ROAD!REGIONS	STRING	"ROAD1 ROAD2 ROAD3"
.	.	.
.	.	.
OILT1	INTEGER16!LIST	95 39 23
OILT2	INTEGER16!LIST	117 125 31
.	.	.
ROAD1	INTEGER16!LIST	23 142 59
ROAD2	INTEGER16!LIST	114 52 65
ROAD3	INTEGER16!LIST	101 9 48

Figure 4: 2-level region representation in NTV slots. The meanings of the INTEGER!16 VALUE's are: X and Y for the seed, followed by the merging threshold.

This organization allows access to the region information in a reasonable and structured way. Certain regions or catagories of regions can be easily generated by accessing their region-growing parameter slots and re-growing using those parameters. While region-growing is computationally expensive, this representation is concise. Notethat other region information could be stored in this manner as well, such as the mass, centrex, and enclosing window. This use bears a resemblance to scene descriptions in "Image Description Files" (IDF's) in the MIDAS system at

CMU ([Mckeown]). The difference here is one of scope: the MIDAS system implements a complete relational data base. The scope of the NTV system is more modest, yet retains considerable power.

Another current use of NTV triples is to store string pointers to derived images that are expensive to compute, such as the convolution of an image with a certain kernel. Thus, when the convolution of an image is required, the header can be checked to see if it exists already; if not, it can be computed, stored in a disk file, and linked to the original with an NTV slot (again, this can result in very significant time savings).

We also have used NTV slots to directly annotate images as they are digitized. This has provided a very convenient method of keeping track of images from the start: they are guaranteed to be labelled with a title and date, as well as the settings of the digitizer at the time.

FUTURE USES:

Some straightforward extensions of current uses are also being planned, including ARRAY slot types, universal slot names for memo functions, and file pointers to histogram files. In addition, we are beginning to explore the implications of using slots as file pointers for intelligent structuring of our image files, one of the

current problems in image management.

Part of this exploration has been to ask "how do we want to access and use images, both interactively and from a program?" Currently image files are accessed by file name only, but it seems more reasonable that images should be accessed by their characteristics, part of which may be relations between images. For example, a person or a program may want an image that is a 64 by 64 window of an aerial photograph of an industrial site, that has been FFT'ed, and contains round structures. Other details, such as the actual file name or even where it resides on a distributed system, may be unimportant.

We are exploring using NTV slots as file pointers to allow this kind of access. One possible concept is that of an "image family", a linked list of images related in a certain way. For example, all original lung images, convolutions of a certain image using different window sizes, and a single image at different resolutions (the latter is a structure that we are exploring for display and transmission purposes, see ([Sloan])); are all candidates for image families. The general result of these file pointers in the image headers is to structure the images into a tree or graph, illustrated in figure 5.

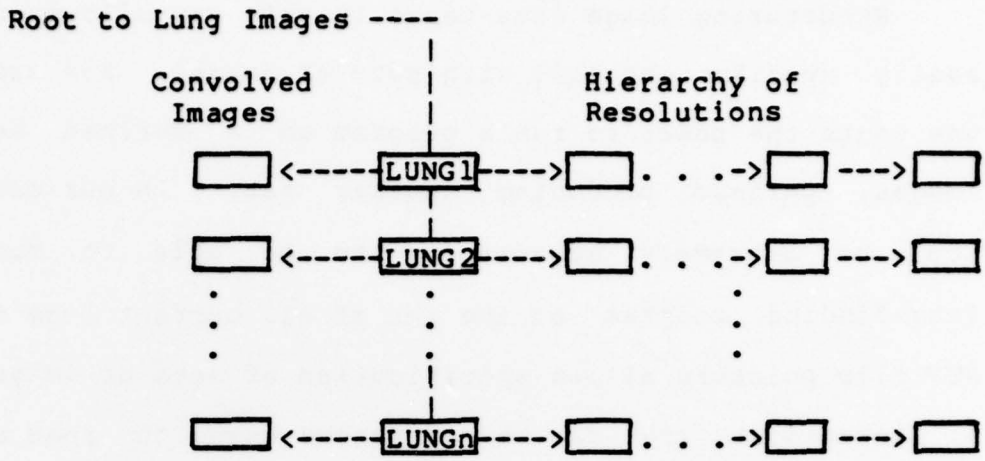


Figure 5: A hypothesized stuctured image data-base.

We anticipate setting up this kind of structure prior to starting our new effort in automated cartographic aids. We are contemplating an interactive image "fetcher" which would have access to various "roots" of the image data structure, and would use links to traverse the structure the fetch the image or images with the characteristics required.

This bears a relationship to various proposals for image data-bases ([Lein], [Tamura], [Mckeown], [Chang], [Kunii]). There are several differences in this scheme to those proposed. In general, image management is a complex topic with approaches at many levels. The level addressed here is that of information physically associated with image files. The advantages to image portability have been mentioned. In addition, NTV information is simple to specify and is very general: information of many kinds, including links to other data structures is allowed.

Structuring image data-bases in this way allows one to easily specify and deal with sets of images. For example, one wants the power to run a program on a defined set of images, perhaps producing another set. In our case, it would be extremely advantagous to be able to run our lung-finding program on the set of all current lung files. NTV file pointers allows specification of sets of images as a linked list, that can be referenced using the root of the list. Other uses include "display families" containing a set of images to be sequentially displayed, and sets of derived images illustrating a particular sequence of image operations.

CONCLUSIONS:

We have described in this report a simple and flexible data structure for image information, NAME-TYPE-VALUE triples, and convenient primitives for manipulating NTV information. This data structure retains the property that image information is directly associated with each image, with advantages to the image portability problem. It is an excellent mechanism for storing image-specific information and retaining relationships between images. NTV slots are not optimal for storing large quantities of data, but are ideal for associating such data, stored in other structures, with their related images.

We have used this system in several ways, all of which ease the problems of accessing images and image information of various kinds in a general and user-controlled way. In addition, this approach addresses several problems in image management, primarily the problem of intelligent structuring of image data files to allow flexible access for people and programs. We are optimistic that this approach will prove even more useful in the future.

REFERENCES:

[Chang] Chang, S. K. et al., "A Relational Database System for Pictures", in [WorkshopIEEE], pp. 142-149

[Feldman] Feldman, J. A., and Rovner, P. D., "An Algol-Based Associative Language", CACM 12:439-449, 1969

[Haralick] Haralick, R. M., and Minden, G., "KANDIDATS: An Interactive Image Processing System", Computer Graphics and Image Processing 8:1-5, 1978

[Hayes] Hayes, K. C. Jr., "XAP Users' Manual", TR-348, University of Maryland. Revised August 1975

[Kunii] Kunii, T., Weyl, S., Tenebaum, J. M., "A Relational Data Base Scheme for Describing Complex Pictures With Color and Texture", Second International Joint conference on Pattern Recognition, 1974

[Lein] Lein, Y. E. and Utter, D. F., "Design of an Image Database", in [WorkshopIEEE], pp. 45-54

[Low] Low, J., "PLITS Memo #4: Internal Format", Internal Memo, Department of Computer Science, University of Rochester, Rochester New York 14627

[Mckeown] Mckeown, D. M. and Reddy, R., "A Hierachical Symbolic Representation for an Image Data Base", in [WorkshopIEEE], pp. 40-44

[Schmidt] Schmidt, R., "The USC Image Processing Institute Data Base, Revision #1", Image Processing Institute, University of Southern California, University Park, Los Angeles, California

[Selfridge] Selfridge, P., "Two Standard Region Growers", Internal Memo, Department of Computer Science, University of Rochester, Rochester, New York 14627

[Sloan] Sloan, K. S. Jr., and Tanimoto, S., "Progressive Refinement of Raster Images", TR-38, Department of Computer Science, University of Rochester, Rochester, New York 14627

[WorkshopIEEE] Workshop on Picture Data Description and Management (Chicogo, April 21-22, 1977), IEEE Publication 77CH-1187-4C

[WorkshopNBS] Workshop on Standards For Image Pattern Recognition (National Bureau of Standards, Gaithersburg, Maryland, June 3-4, 1976), NBS

Special Publication 500-8, May 1977

[Zobrist] Zobrist, A. L., "Elements of an Image-Based Information System", in [WorkshopIEEE], pp. 55-60

Appendix I: Detailed Image Header Format and Internal Format of NTV Triples

An image file consists of a list of pixels organized as a sequence of "Scanlines". Each Scanline starts on a "Word" boundary and consists of a sequence of Words. The size of a Word (in bits) may depend on the architecture of a particular machine. If the file system being used allows or requires one to deal in "blocks", then Scanlines also are aligned on block boundaries. Each Word contains an integral number of Samples (left justified). One or more Samples make up a Pixel (several samples per pixel are used to represent multi-spectral images).

This list of pixels is preceeded by a header, which contains format descriptors in fixed locations, and a sequence of NTV slots, as shown below:

Password
Length of Header
Raster Part Code Raster Part Length
Number of Scan Lines
Samples per Scan Line
Scan Direction
Coding Type
Samples per Pixel
Bits per Sample
Words per Scan Line
Scan Lines per Block
Word Length
Samples per Word
NTV Part Code NTV Part Length
Number of Slots
Slot 1
Slot 2
.
.

The NTV slots are stored sequentially as shown above. Each slot consists of a NAME, a TYPE, and a VALUE. The NAME is a string, from 1 to 255 ASCII characters, designating the NAME of the NTV slot. (Its internal format is the same as that of the STRING type shown below). The TYPE is an integer code, stored in one byte, representing the TYPE, and thus the internal format, of the VALUE. The VALUE is specified as a string in calls to NTV primitives, and coerced to the format below.

The basic unit of storage is an 8-bit byte. The various TYPE's and the format of the corresponding VALUE's is listed below:

16 bit integer - 2's complement. Stored as 2 bytes.

36 bit integer - 2's complement. Stored as 5 bytes.

Boolean - 1 byte. (Ø = FALSE, -1 = TRUE).

PDP1Ø real - 36 bit PDP-1Ø real. Stored as 5 bytes.

ALTO real - 32 bit ALTO real. Stored as 4 bytes.

string - 1 byte length, followed by the string, 1 ascii character per byte.

list of 16 bit integers - 1 byte count, followed by that many 16 bit integers.

list of 36 bit integers - likewise.

list of Booleans - likewise.

list of PDP1Ø reals - likewise.

list of ALTO reals - likewise.

APPENDIX II: NTV PACKAGE HELP FILE

Introduction:

The NTV (Name-Type-Value) package is a set of routines that allow a user to store and access auxillary information in RV file headers. In addition, it will include forthcoming programs for listing headers in the new format, editting headers, and other useful functions.

Information to be stored in headers must be in the form of Name-Type-Value triples (also called slots). Both the name and the value must be strings, and the type is specified with an integer code. For example,

"AVERAGE"-INTEGER16-"155"

is a valid triple.

All routines return a STATUS, which, if not 0, will indicate some error. The status may be compared with a number of integer codes to determine its meaning, or used as an arguement to an error printing routine that will print a coherent error message.

Linking:

To link the NTV package to your program, include the line:

```
REQUIRE "NTVPAK.HDR[170,457]" SOURCE!FILE;
```

in your program. All routines and integer codes will be accessable.

Calling:

The calling format for the various routines are as follows: In the following calls, ARR should always be an array 256 long to hold an RV header. NAME should always be a string, TYPE an integer code (see below), and the VALUE a string; see below for converting values to strings.

```
STATUS <-- HDRIN (CHAN, ARR)
```

This reads the header of the file open on CHAN into array ARR. The file is left open, with the file pointer pointing to right after the header.

STATUS <-- HDROUT (CHAN, ARR)

This writes the header in array ARR out to the file open on CHAN. The file is left open, with the file pointer pointing to right after the header.

STATUS <-- PUTNTV (ARR, NAME, TYPE, VALUE)

This will write the triple out to the header in ARR.

STATUS <-- GETNTV (ARR, NAME, TYPE, VALUE)

This will return the type and value of the NTV triple with the name NAME in TYPE and VALUE.

STATUS <-- FETCHNTV (ARR, N, NAME, TYPE, VALUE)

This will return the name, type and value of the Nth NTV triple in NAME, TYPE and VALUE.

STATUS <-- DELETENTV (ARR, NAME)

This will delete the NTV triple with the name NAME, and repack the rest of the header.

STATUS <-- INITNTV (ARR)

This will initialize the NTV part of the header.

STATUS <-- GETROOM (ARR)

This will return the number of free whole words left in the header.

PERR (STATUS)

This will print on the terminal an error message in English, given an error code returned from one of the above NTV routines.

TYPE!STRING := PRINT!TYPE (CODE)

PRINT!TYPE returns the string representing a type code. That is, PRINT!TYPE (INTEGER36) returns the string "INTEGER36". This is useful for printing the types of slots. If the code is invalid and represents no type, the string

"erroneous code!" is returned.

CODE := GET!TYPE!CODE (TYPE!STRING)

This is the converse of PRINT!TYPE. Called with a string representing a type, it returns the integer code. If the string is invalid, it returns a -1.

TYPES:

The following types are available, and are indicated by putting the code in the TYPE part of a specified NTV triple:

TYPE	CODE
16 bit integer	INTEGER16
36 bit integer	INTEGER36
boolean	BOOLEAN!
PDP 1Ø real	REALPDP
ALTO real	• REALALTO
string	STRING!
list of INTEGER16's	INTEGER16!LIST
list of INTEGER36's	INTEGER36!LIST
list of BOOLEAN's	BOOLEAN!LIST
list of *PDP real's*	REALPDP!LIST
list of ALTO real's	REALALTO!LIST

VALUES:

As mentioned, all values must be strings. Therefore, conversions are neccessary from the original typed value to a string. The following SAIL conversion 51troutines are appropriate for the indicated types:

INTEGER16	CVS
INTEGER36	CVS
BOOLEAN!	CVS
REALPDP	CVE, CVF, or CVQ
REALALTO	CVE, CVF, or CVQ

For the lists, each element must be converted, and concatenated with a seperation character, such as " " (space) or ",". Of course, if the values are read in from a terminal, no conversion is necessary. Forthcoming routines will allow conversion of PDP INTEGERs and REALs to ALTO REALs.

INTEGER CODES:

The following error codes are included in the package for

comparision with the returned status. Calling the routine PERR with the returned status (see above) will print an appropriate error message on the terminal.

```
SUCCESS - no problem
NO!NTVPART - no NTV part in header
NO!ROOM - no room for the NTV triple
NO!SLOT!FOUND - the requested slot
        was not found
NO!PASSWORD - the RV password
        was not found
NO!RASTERPART - no rasterpart code
        was found
SLOT!EXISTS - the slot with that name
        already exists, so you cannot put
        another one
BAD!NAME - the name passed was bad
BAD!TYPE - the type passed was bad
BAD!VALUE - the value passed was bad
BAD!SLOT!NUMBER - the slot number passed to
        FETCHNTV was negative
```

EDITTING HEADERS:

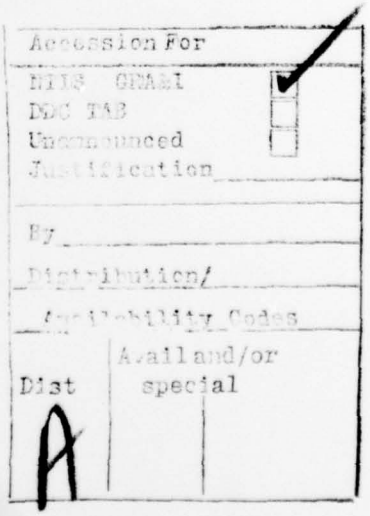
The program NTVEDT [170,457] uses all the above routines, and will allow the user to read in headers, put, get, delete, et cetera NTV triples. It will also list the existing triples in a header. The program RVLIST [170,457] will list in a convenient format either all slots, or only the TITLE slots, of RV files. The listing will go on the terminal or into a file.

APPENDIX III: A PROGRAM TO ADD "TITLE" SLOTS TO FILES

```
Begin
Require "Std.hdr[170,161]"      source!file;

! This links all NTV routines;
Require "Ntvpak.hdr[170,656]"   source!file;

! This defines the RV header format;
Require "RvHead.Hdr[170,457]"   source!file;
Integer chan, chan1, status, typ, i;
String fname, val, title;

! allocate header space, RvHeaderSize is defined in
RvHead.Hdr;
Integer Array hdr[0:RvHeadersize-1];

! this program asks for a file name, and, if one doesn't
! exist, puts a TITLE slot in the header;

    Print ("Type file name: ");
    chan := Openin (fname := inchwl, i, 8);
    status := Hdrin (chan, hdr);
    status := Getntv (hdr,"TITLE", typ, val);
    if (status = no!slot!found) then
        Begin "create a slot"

            Print ("input a title string: ");
            title := inchwl;
            status := Putntv (hdr, "TITLE", string!, title);

! if error, print it and continue;
            if status then perr (status);

! now write it all out;
            chan1 := Openout (fname, 8);
            hdrout (chan1, hdr);
            inout (chan, chan1, -1);
            Release (chan1);

        End "create a slot"

        else Print ("slot already exists, with value:
",val,crlf);

        Release (chan);

End;
```

APPENDIX IV: IMAGE FILE LISTING FROM RVLIST

```
DATE: 14/11/78          TIME: 22:11:26

RV LISTING FOR THE FILE(S): lung*.rv

FILE: LUNG9.RV
  ROWS: 32      COLUMNS: 32
    SLOT 1
        NAME: TITLE
        TYPE: STRING!
        VALUE: LUNG9.RV

    SLOT 2
        NAME: ORIGINAL
        TYPE: STRING!
        VALUE: LUNG3.RV

    SLOT 3
        NAME: WINDOW COORDINATES
        TYPE: INTEGER16!LIST
        VALUE:  35 45 65 75

    SLOT 4
        NAME: COMMENT
        TYPE: STRING!
        VALUE: THIS WINDOW IS IN THE LOWER LEFT LOBE, AND
CONTAINS A TUMOR


FILE: LUNG8.RV
  ROWS: 32      COLUMNS: 32
    SLOT 1
        NAME: TITLE
        TYPE: STRING!
        VALUE: LUNG8.RV

    SLOT 2
        NAME: ORIGINAL
        TYPE: STRING!
        VALUE: LUNG.RV, 256 BY 256

    SLOT 3
        NAME: PROCESS
        TYPE: STRING!
        VALUE: AVERAGING

    SLOT 4
        NAME: REGIONSLOTS
        TYPE: STRING!
        VALUE: REGION1 REGION2

    SLOT 5
```



```
        NAME: REGION1
        TYPE: INTEGER16!LIST
        VALUE:  8 25 25

    SLOT 6
        NAME: REGION2
        TYPE: INTEGER16!LIST
        VALUE:  5 5 25 10 10 25
```