(12)

RADC-TR-76-197
Final Technical Report
June 1976

DEVELOPMENTAL SUPPORT OF THE SCIENTIFIC AND TECHNICAL INTELLIGENCE SYSTEM

Auerbach Associates, Inc.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report has been reviewed and is approved for publication.

APPROVED: [illegible]

ROBERT N. RUBERTI
Project Engineer

APPROVED: [illegible]

HOWARD DAVIS
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: [illegible]

JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-76-197	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DEVELOPMENTAL SUPPORT OF THE SCIENTIFIC AND TECHNICAL INTELLIGENCE SYSTEM		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report June 1975 - April 1976 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Dr. Jerome Sable		8. CONTRACT OR GRANT NUMBER(s) F30602-75-C-0273
9. PERFORMING ORGANIZATION NAME AND ADDRESS Auerbach Associates, Inc. 121 N. Broad Street Philadelphia PA 19107		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 64750F 20530203
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDT) Griffiss AFB NY 13441		12. REPORT DATE June 1976 13. NUMBER OF PAGES 154
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Robert N. Ruberti (IRDT)

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

Intelligence Data Handling
Information Management
Relational Data Technology

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

This project has provided technical assistance to the development team of the Scientific and Technical Intelligence System (STIS) at Air Force Foreign Technology Division (FTD). The effort was directed at the design and documentation of several data structures and programmed capabilities of STIS. These included the following:

(a) An Indexed-Sequential Access Method and Directory for

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

over

encoding and decoding STIS user and system terms,

(b) the data structure of a generalized STIS node which can serve for the elements of the STIS Concept Net, including entities, semantic elements (attributes and values), and derivation rules, and

(c) a set of interpretations of the node structure and the definition of an approach which can be used by the analyst in representing intelligence information in the Concept Net, including source information, credibility, events, states, transitions, and general deductive rules.

PREFACE

The work reported here was accomplished under contract F30602-75-C-0273 under sponsorship of the Rome Air Development Center. The objective of this project was to provide technical assistance to the STIS (Scientific and Technical Intelligence System) development team at the Air Force Foreign Technology Division. The report was prepared by Jerome Sable, Edward Schernecke, and John McCrea of the staff of AUERBACH Associates, Inc. Grateful acknowledgment for special consultation and assistance is given to Edward Stull, Jerry Hamilton, and Janet Merelli of FTD and to the Technical Monitor, Robert Ruberti of RADC. Dr. Sable was the Program Manager of this project.

EVALUATION

Program design specifications have been delivered for an information structuring method that will be applied to the FTD scientific intelligence data base. It includes specifications for an entity and semantic network structure, as well as directory services and a peripheral file access system. These specifications will be implemented as part of the Scientific and Technical Intelligence System (STIS) at FTD. This effort is included as part of TPO #4, Intelligence Data Handling.

[illegible]

ROBERT N. RUBERTI
Project Engineer

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CON'D)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I. INTRODUCTION

1.1 OBJECTIVE

The objective of this project was to provide technical assistance to the development team of the Scientific and Technical Intelligence System (STIS) [1]* at Air Force Foreign Technology Division (FTD). The effort was directed at the design and documentation of several data structures and programmed capabilities of STIS. These included the following:

(a) An Indexed-Sequential Access Method and Directory for encoding and decoding STIS user and system terms,

(b) the data structure of a generalized STIS node which can serve for the elements of the STIS Concept Net, including entities, semantic elements (attributes and values), and derivation rules, and

(c) a set of interpretations of the node structure and the definition of an approach which can be used by the analyst in representing intelligence information in the Concept Net, including source information, credibility, everts, states, transitions, and general deductive rules.

*References are listed after the main body of this report.

1.2 BACKGROUND OF STIS

STIS is being developed as a tool to help fulfill the Scientific and Intelligence analysis mission of FTD. It serves the intelligence production programs in application areas such as IPS (Intelligence Production System), IEAS (Integrated Event Analysis System), C^3 (Command, Control, and Communications), and EW (Electronic Warfare), and the processing of metric sensor data performed by the engineering analyst who may wish to operate in an interactive mode on-line with STIS as well as invoking production type programs.

STIS is being implemented on the UNIVAC 1110 and has evolved from a system called BIAS which was originally based on the IBM S/360 GIS.

STIS provides an advanced capability for the analysis of intelligence information. It is based on a network type data structure which permits relationships among entities and new attributes to be freely defined with minimal impact on previously stored data and programs. Because of this, it is particularly suited for capturing fragmentary information which is undergoing collation processing, analysis, evaluation, and synthesis into finished intelligence.

In another research effort, called the BIAS Augmentation Study, AUERBACH has developed the design of advanced relational data and inference providing tools for use in an operational intelligence environment. The BIAS Augmentation Study has been working with BIAS, and now STIS, as a vehicle with which to develop and test operations on relations, inference, and consistency determining functions. The ultimate goal is to incorporate these advanced capabilities into STIS so that their effectiveness can be accurately evaluated, and these new tools can be provided to the STIS analyst.

In 1973, AUERBACH Associates, Inc. developed a design for an inference capability for an intelligence system such as STIS [2]. The design concept of this capability, called ARIAS (Augmented Relational Intelligence

Analysis System), embraced both inductive and deductive inferences that an analyst may desire to pursue in testing an hypothesis (or answering an interrogation) concerning the current state-of-affairs as reflected in an intelligence data base. As a result of a subsequent project, the algorithms concerned with the deductive aspects of this capability, and its interface with STIS, have been recently detailed [3].

1.2.1 The Information Structure as a Key Element of STIS

STIS is an intelligence system and not a data base management system in the conventional sense. That is, it is a specialized tool which serves the unique needs of the intelligence analyst, and is not intended for generalized use in maintaining and extracting information from predefined files. It is intended to adapt to the variable information structures created by analysts who view the world as a dynamically changing state-of-affairs and must react quickly to new information requirements and employ changing strategies of information correlation. The intelligence analyst, as distinct from his counterpart in the commercial data processing world, must routinely accept and maintain conflicting data and partial information, assess the credibility of the information, and then use it to construct a coherent estimate of a state-of-affairs.

During the early stages of the evolution of the STIS concept, attempts were made to utilize commercially available Data Base Management Systems as a basis for building STIS. These early STIS versions used, or experimented with, GIS and FORIMS (on the S/360 and the U1110, respectively), and evaluated other Data Base Management Systems as well. They each proved inadequate to the task of serving the broad spectrum of requirements inherent in STI (Scientific and Technical Intelligence) data analysis without redundant storage of large segments of the data base, continual definition of new data structures, and redefinition of old structures.

The approach taken by the designers of STIS was to face squarely the variability of intelligence data, the need to accept and qualify information on the basis of time, source, and credibility, and the need to adapt to multiple views of the same information with a minimum of redundancy so that the data maintenance and updating problem could be kept within bounds of processing and storage capability.

They consequently developed a powerful, yet tractable network-oriented information structure that enables the analyst to describe a real-word object, situation, event, or relationship without being constrained to some preconceptions of what attributes that description should contain. This leads to more concise data elements and to greater adaptability to new states of affairs.

A major innovation to be implemented for STIS (both for the near-term improvement of STIS 1 and for STIS 2), is the ability to represent a specified set of data values in a conventional file arrangement, without the user being explicitly aware that this is being done. This technique makes it possible to handle large files of metric data in fixed-format records, thus providing the high degree of processing efficiency which is required for this function.

1.2.2 STIS Users

The users of STIS fall into two broad categories, on-line and batch. The on-line user is the STI analyst who wants to answer an ad hoc request, one that has not been formalized to such an extent that a pre-programmed solution for it exists. He wants to extract specified data from the data base to respond to a Quick-Reaction Request or perhaps to test the validity of a new analysis strategy which he is attempting to formalize. The key element is that the on-line user must interact effectively with STIS in ways that cannot be completely pre-specified. He therefore needs a language for man/machine communication which can invoke responsive search and retrieval routines and which he can use first to build, and then to conveniently invoke, special sequences of operations which suit his analysis tasks. An interactive communication mode for STIS has been specified and will be integrated with STIS as part of a Design Optimazation and Development effort.

The batch user can be viewed as being one stage removed from direct interaction with the machine. He is buffered from direct contact with STIS and the computer by a program which carries out a formalized set of analyses and reporting tasks, and therefore the batch program which he is using can be thought of as providing him with a specialized interface with STIS. From the point of view of STIS, however, the "direct" user of STIS in this case is the programmer who makes use of the generalized STIS commands through program calls. He is the user who creates the specialized interface for the analyst who can formalize his requirements, or who chooses not to interact directly with STIS. The STIS programmer-user generally writes in a high level procedural language such as COBOL or FORTRAN, and STIS provides a High-Level Language (HLL) interface for these users.

SECTION II. THE TECHNOLOGICAL FOUNDATIONS OF STIS

The technological foundations of scientific and technical intelligence analysis embrace three major elements: the framework within which the problem is viewed, the information structures which are used, and new concepts in semantic networks and knowledge-based systems.

2.1 THE INTELLIGENCE ANALYSIS FRAMEWORK

The problem of intelligence analysis has a number of distinct features which set it apart from other problems in data analysis, yet, in some of its aspects it shares many features with other areas of scientific investigation:

- The analyst is concerned with sets of entities in the real world: individuals, facilities, organizations, systems, events, messages, experiments, and observations.
- Each entity can be identified and described by a set of facts (or properties) and relationships to other entities.

- General rules of behavior, association of properties, or interrelationship among entities of given types can be defined.

- The facts are, in general, not constant in time but represent a "snapshot" of a dynamic situation, and the influence of temporal events are an important ingredient of the situation.

- The known state-of-affairs is a result of observations made on the real world by imperfect sensors and is subject to equivocating influences, which produce "noisy" and possibly inconsistent "facts" as well as fragmentary or incomplete descriptions of objects and events.

- Both the facts and the rules can be associated with a subjective probability of truth (credibility, validity or acceptability), which reflects the analyst's, the observer's or the system's estimate of their uncertainty.

- The analyst is concerned with the degree of consistency of specific and general statements (facts and rules), the degree of support (derivability or predictive power) of hypotheses which he may invent, the uncertainty of source information and credibility of conclusions.

The scientist and the intelligence analyst are each often concerned with developing a theory which explains some aspect of the behavior of the real-world domain on which he is focusing. A major difference, of course, is that the intelligence analyst is typically concerned with the description, behavior and plans of a non-cooperating adversary, while the scientist is typically concerned with a system which is indifferent (but not necessarily non-reactive) to his observations.

Notwithstanding the above similarities between intelligence and other types of scientific analysis, there are distinct differences between the two. Relative to hypothesis concept formation and testing in most other areas of scientific investigation, intelligence analysis often involves a higher degree of uncertainty, a lack of repeatability and control of experiments, and a lack of accessibility of the subject. There are also distinct differences in the cost, feasibility, and potential ramifications of observation and experimentation, and the nature of credibility criteria and rules of evaluation.

Because of these considerations, the psychology, philosophy and personality of the analyst often enters the situation in a more direct way in intelligence than in other scientific investigation. The intelligence analyst's system is very often a personal system, and is not easily amenable to sharing and interpretation by other analysts. Furthermore, there is apt to be a high degree of complexity in the composite factors which the intelligence analyst is seeking to define and evaluate, such as capability of the enemy to carry out a given plan, the level of threat represented by the current situation, the net balance of forces or capability, etc.

The above differences along with other considerations give rise to a number of unique problems associated with handling intelligence information. Some of these problems and characteristics of intelligence information are listed below:

- The volume of intelligence information to be handled is high. It is very often in non-standardized, or partially standardized, format. (Screening and processing unformatted text is typically a bottleneck in intelligence systems, and represents a problem aspect which is only partially solvable by automatic methods.)

- The input data is often fragmentary. That is, only a small subset of the possible identifying properties of the real-world entities being examined or modeled by the system are specified in a typical input message. Indeed, the complete set of descriptive properties of entities, and the relations among entities which will be of interest cannot be specified at the outset, and remains a growing set during the life of the system. Hence collation of new information with previously stored data is often difficult, and can be accomplished only in a conjectural, plausible, or probabilistic sense.

- Input messages must be treated as independent observations of a dynamic situation, and therefore it is important to preserve and utilize dateline, source, and temporal aspects in the data.

- There is an essential problem in establishing the validity of each message and stored data item. Thus, it is important to associate ancillary information concerning the source and credibility of all input data, and some measure of validity to resulting conclusions.

- Processing and inference rules are usually not completely rigorous in a formal sense, hence it is important to consider them, and the information derived by their application, as "belonging" to specific analysts, with specified areas of applicability, and levels of validity. Furthermore these processing and inference rules are not static, hence they should be embodied in the data base, modifiable, and processed by generalized routines, rather than being implicit in programmed routines where they are relatively difficult to modify.

2.[illegible] INFORMATION STRUCTURES

These requirements lead to a rejection of conventional or "prepackaged" approaches to structuring information. In particular, the fixed record customary in conventional data processing systems must be abandoned because it is virtually impossible to pre-define a fixed set of attributes which are appropriate for an entity of a given type. Even where this is conceivable, the highly fragmentary nature of facts known about a particular entity would imply a sparse occupancy ratio of data in the record, hence a poor utilization of storage.

The major conventional data structuring strategies which have been evolving in computer-based information processing are called:

a) hierarchic,
b) network, and
c) relational

data base systems*. There have been commercial Data Base Management Systems based on each of these strategies (e.g., IBM's IMS is basically hierarchic; CODASYL's DBMS, or Honeywell's IDS, is a network system; and General Motors RDMS is a relational system). Each has advantages and disadvantages as tools for implementing information systems, and each is most appropriate for a given type of problem and development environment. For example, the chained (link-sequence) structure within the record used in some data management systems (such as CODASYL's DBMS) was rejected because it implies a highly sequential processing of information (i.e., along the chain), and a dedication of space for storage of the linkages.

* CJ Date: An Introduction to Data Base Systems. Addison-Wesley, 1975

STIS solves the information structure problem by representing an intelligence entity as a node in a network-oriented data base. Basically, a node is a list of attribute-name/attribute-value pairs, where the value may, in general, be either a single or multiple occurrence (array), or a structured sub-node. Thus an n-tuple (n-ary relation) is available as a special case of an attribute (relation) value. Furthermore, a value may be an array of identification codes of other nodes, permitting, in effect, any node to identify a set of nodes as members of a given relation. This permits any node to serve as an index (inverted list) of information subject to such powerful retrieval techniques as set intersection, union, and difference, and permits the nodes thus inter-connected to serve as an associative network of semantic and intelligence information.

In effect, the STIS information structure can be viewed as an amalgamation of the three data structuring strategies cited above. Within the node it is hierarchic, utilizing the lack of redundance characteristic of that strategy; it is network-oriented when viewed as multi-node system, providing the richly inter-connected access paths needed in intelligence processing; and it also utilizes the formalism of relations for internode relationships and to provide for ease of communication with the user. (None of the commercially available DBMS's based on the relational strategy allows for variable (n-ary) relations, a feature unique to STIS.)

In order to come to grips with the inherent (and unique) characteristics of intelligence information, the designers of the STIS at FTD and AAI have chosen a data concept and data structure distinctly different from that employed in conventional data processing system.

The STIS structure which is used to store information for on-line and batch users is called the Entity Node. Although the current Entity Node handles variable and fragmentary information, it lacks a general capability to gather, in a local physical context, sets of n-ary relationships (n-tuples) and subentities. To accomplish this in STIS, information which may be logically conceived as a single entity must be structured as separate nodes and therefore be subject to several access operations when retrieved from secondary storage.

The ability to retrieve information by condition is quite restricted in the current STIS, being limited to retrieval under system, user, and AOR sets. Different information structures are used for sets and for entitites.

The improved information structure to be introduced will remove these shortcomings in the near-term and will be adaptable to future enhancements of STIS. The current STIS information structure is described in detail in Appendix A, and the new information structure is described in Section 4. A brief description of the new information structure is given below.

The STIS information structure is based on a network of nodes. The node is a hierarchic data structure which brings together all properties and relations which are associated with a particular concept (semantic element, or entity). A variable format is employed which permits storage to be dedicated to only those properties of the entity which are actually known, with no space dedicated to "unknown" attributes. Provision is made for multiple source information, with multiple (and perhaps conflicting) values for a given attribute. Provision is also made for describing a generic, or "standard", version of an entity as well as an arbitrary number of particular embodyments, and "states", of particular entities.

Each node carries the description of a logical entity or event. The description consists of a variable number of properties and is arbitrarily nested with structure elements representing subentities and qualifications. The data nodes are logically interlinked by attributes which represent relationships between entities.

Each descriptive property consists of an attribute-name/attribute-value pair. One of the major innovations to be developed for improvement of STIS, is the ability of a value to be represented by a conventional file and accessed by a conventional access method, without the user being explicitly aware that this is being done. This technique makes it possible to handle large files of metric data in fixed-format records without explicitly invoking the STIS relational mechanism at a level where a large overhead would be incurred.

Because the STIS entity is basically an attributes-under-entity structure and because access to the entity network is typically conditional by subject, Area of Responsibility (AOR), and/or attribute value, index lists of the entity network play a very important role in STIS. Indexes are basically entities-under-attribute structures or "inverted" files, and allow conditional access to the entity network to be accomplished directly and efficiently, without the necessity of sequential search through the entity network. Whereas STIS now uses a different information structure for the entity and for the "set" or index, the enhanced STIS uses a common node structure.

In choosing the STIS node as the particular structure to represent information ("f :s") in the system, the designers have created an information structure which can provide the following features and capabilities:

- efficient utilization of random access (secondary) storage for a very large data base of upwards of 500,000 records
- accommodation of a variable set of attributes for the entities to be described in the system
- quick retrieval of the facts known about an entity identified by an arbitrary subset of these facts
- retrieval of all entities having an arbitrary relationship to a given entity, or which are members of a given set.
- easy modification of the set of facts concerning a given entity
- ability to create and maintain a vocabulary of terms and semantic relationships for effective representation and communication of facts about entities and their membership in conceptual categories and relations
- ability to qualify facts in a very general way, including their temporal validity and relationship to events, access control information, source, credibility, and relational operators such as "approximately," "not equal to," etc
- ability to store general rules and potential inference chains as well as specific facts

- a variety of data types, including alphnumeric, text, and data (both fixed and floating-point formats)
- an effective interface to batch mode application programs
- an effective on-line communication both through a generalized user interface and through special modes provide by application programs
- an environment for dynamic data base growth and maintenance
- controls for data access authorization and data base protection
- capability to handle multiple values for any attribute, and
- enhanced modes of analyst communication through textual annotations such as warnings and comments.

2.3 SEMANTIC NETWORKS AND CURRENT RESEARCH IN KNOWLEDGE-BASED SYSTEM

Although STIS is based on proven "state-of-the-art" technology, the enhanced STIS data structure closely resembles the data structures used in advanced research programs in "semantic networks" and "knowledge-based" systems. Experimental Systems which model or represent the state-of-affairs of some real-world situation or, more precisely, a human's view of that situation, have been reported in the artificial intelligence research literature for a number of years*. The common thread running through these systems is that they allow the user to freely describe and define relationship among the entities and objects which inhabit his "world."

One of the largest and most advanced research program in knowledge-based systems is the Computer Based Consultant (CBC) system being developed at Stanford Research Institute under ARPA sponsorship. The CBC is

* A study of such systems was conducted by AAI and reported in Relational Data Study, RADC-TR-70-180, September 1970, (720263). It includes, for example, a discussion of Quillian's "Semantic Net" in the Teachable Language Comprehender (BBN)

designed to communicate with the user in natural language and help him perform tasks entailing maintenance and trouble-shooting of electromechanical equipment. The current test data base describes the structure, parts, and maintenance of an air compressor, and can generate and execute plans for assembly and disassembly at several levels of detail, and answer questions about the equipment posed by an apprentice. The system is programmed in QLISP, an extension of the LISP language which runs on the PDP-10 under the TENEX operating system.

The data base of the CBC is a semantic net whose nodes and attributes have been divided into "partitions" forming a partially ordered set. The main objective of partitioning is to define the scope of quantification statements.

The enhanced STIS data structure, called the Concept Net, bears a close kinship to the type of semantic net used in the CBC. The Concept Net will also be partitioned. However, partitions in the STIS data base will serve additional functions. They will define sets of information which "belong" to a particular analyst or Area of Responsibility (AOR) and which may be allocated to separate storage modules (i.e., disk packs). Subpartitions will also identify sets of mutually consistent information (and, in later phases of STIS 2, rules and coherent patterns of fact credibility).

SECTION III. OVERALL SYSTEM STRUCTURE

The control structure embodied in STIS can be considered under the following headings:

(a) **General Processing Flow**
(b) **Multi-user Considerations**
(c) **Recursive Processing**
(d) **Security Processing**
(e) **Systems Management Functions**

3.1 GENERAL PROCESSING FLOW

The STI system can be conceived of as being composed of a number of processing levels, providing progressive interpretation, decomposition, execution, and monitoring of user requests. Each level provides a locus for specialized kinds of data screening and request analysis and support. The processing at each level can be modified, augmented, or replaced with minimum effect on other levels. The levels and their relationships are indicated in Figure 3-1.

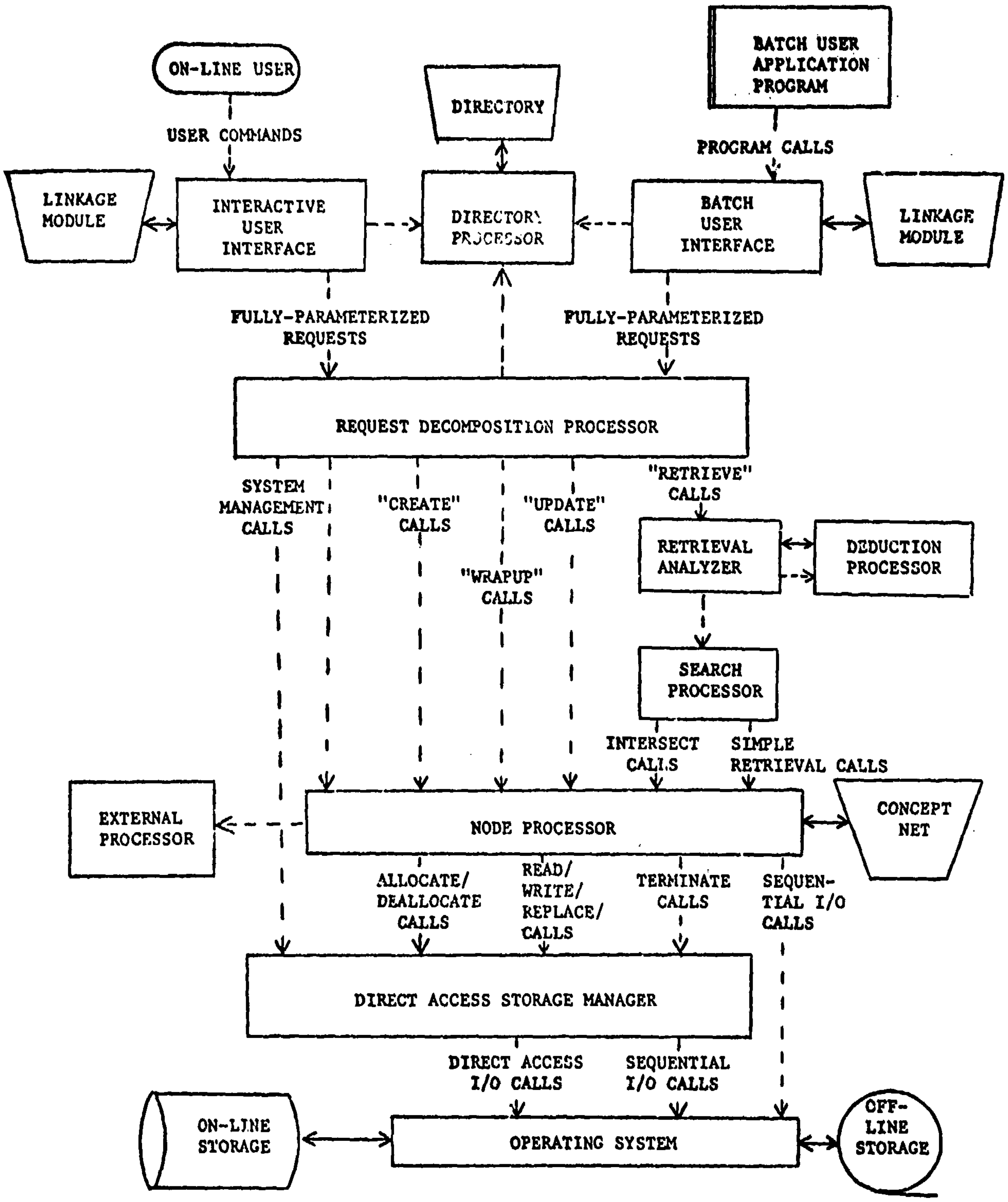


Figure 3-1 STIS Overall Flow and Processing Levels

Although for convenience a processor is shown at each level, the levels should be thought of as functions, which bear no necessary relationship with modules of the system.

3.1.1 User Interfaces

The processing of user calls first takes place at the user interface. Included among the functions of user interface processing are the following:

(1) to insure information base security and integrity by requiring and checking password and access rights information

(2) to execute system- or user-specified special processing or contingency routines under appropriate conditions

(3) to convert data elements from one form to another (this includes term encoding and decoding)

(4) to retain, for each user, inter-call parameters and status information

(5) to amplify (or more fully specify) user calls through the incorporation of previously developed or specified informaiton

(6) to convert calls to a common language or form for processing at the next level

(7) to assign, on initial access by a user, a work area to be used by STIS for data base manipulation.

In general, error conditions occurring at lower levels in the STIS system are passed back up to the user interface processors. Subsequent processing depends on the nature of the contingency and on whether a batch or online user is involved.

There are two main types of user interface processing: batch and on-line.

3.1.1.1 Batch

Batch user interface processing is responsible for interfacing user programs with the STI system. In the current version of STIS, and probably in future versions, this interface processing has two aspects:

(1) a preprocessing phase (currently called Phase I)

(2) interaction with the data base access calls made by the user program (currently called Phase II)

Phase I generates the necessary linkage modules for Phase II operation by processing user subroutine calls that identify data base items of interest, buffer sizes, and special processing and contingency routines. Once generated by Phase I operation, the linkage modules are incorporated into the user program and used repetitively, until changed conditions require a new Phase I generation.

User Phase II programs interact with the STI system via the linkage modules. In addition to providing storage for data elements and parameters, these modules establish the communication path with the interface modules that process the uses calls under guidance of those specifications made during Phase I operation.

Batch user interface processing is currently oriented toward programs written in FORTRAN. However, programs written in other languages are accommodated through interface programs that simulate the FORTRAN call interface.

3.1.1.2 On-Line

The on-line user interface translates commands expressed in an on-line interactive language into requests suitable for processing at the next level. As users sign on and are accepted by the system, they are allocated suitable linkage modules to allow for communication with the STIS information base facilities. As with batch processing, these modules include space for data items and parameters. When a user signs off, or is otherwise terminated, the space containing his linkage modules is released

3.1.2 Request Decomposition

Fully parameterized requests developed as a result of user interface processing are not necessarily executable as single requests in their current form, but may require the execution of a number of simpler requests; or a request may imply the execution of other requests necessary for successful completion. For example, an update request may imply one or more "retrieval" requests in order to locate and access the data element(s) to be updated. Or, an update operation may also imply a series of lower-level retrieve and update requests in order to maintain any optional indexes associated with the updated data element(s). In general, the request decomposition level of processing breaks down complex requests into individual calls processable by lower levels and insures that any implied requests are executed. All data access requests are executed in terms of lower level "create", "update", "retrieve" and "wrap-up" type requests. Service requests for STI system management functions are also handled.

The request decomposition level is responsible for handling any complexities inherent in create or update requests. Retrieval requests, because of the wide range of potential modes of analysis, are handled by a separate level.

3.1.3 Retrieval Processing

Retrieve-type calls transmitted by the request decomposition level are analyzed to determine processing requirements. Simple retrieve ("get"-type) calls are passed on without further processing to the next level.

In advanced versions of STIS, retrieval operations will be enhanced by two additional processors: the search processor and the deduction processor.

If exp[illegible] criteria are to be applied directly in the selection of data, the search processor is called. If rules are available for application to the current request, the deduction processor is executed.

The search processor itself can make use of intersection processes and ultimately generates simple retrieval calls. The search processor can also act as an agent for the deduction processor in performing searches local to deduction processing.

3.1.4 Node Processing

At the node processing level, nodes in the Concept net and associated information in the form of list elements, are created, maintained and retrieved. Some node structures contain factual information; others contain semantic information. All nodes, regardless of role, have a uniform structure and are serviced by the same functions. The system nod functions are described in Appendix D.

Node processing is concerned with the packing of information into blocks, the unpacking of information from blocks, and the locating, moving or copying of information Although it determines the need for direct access I/O activity and for the allocation and deallocation of mass storage segments, it does not actually perform these functions itself but gets them accomplished through calls on the direct access storage manager.

Sometimes the information associated with a node is a collection of records on an external file. The accessing of such information requires the node processor to make sequential-type I/O calls to the host operating system.

The value associated with a given occurrence of an attribute can be specified as being determined by the execution of a pre-defined process which is called by the node processor.

3.1.5 Direct Access Storage Management

The direct access storage manager responds to direct access I/O requests and to requests for the allocation or deallocation of strings of mass storage segments. All requests are associated with a partition-determining reference so that information can be physically segregated. (Physical partitioning is discussed in Section 4.7.)

The storage manager requires terminate calls from higher level processors in order to know when a particular physical partition is to be "closed". A closed resident partition is opened on the first reference to it.

I/O functions associated with external storage management are delegated to the Direct Access Storage Manager modules. "Put Node" and "Get Node" functions within the Node Processor perform compression/decompression operations, and request I/O functions by generalized calls on the Direct Access

Storage manager. External storage to hold node structures will be assigned in blocks consisting of a standard number of consecutive segments (choice of the standard number depends on analysis of typical node sizes and the nature of the physical mass storage equipment chosen). If a particular node structure cannot be contained within a standard block, another standard block is reserved and a pointer to the additonal block is placed at the end of the preceding block.

Buffer space in core to hold the complete standard block within the node processor need not be reserved. The nature of the I/O commands, permits the working buffer within the node processor to be as small as one segment size.

With respect to I/O operations, the storage manager, taking advantage of information supplied in the I/O requests, makes use of a separate buffering system in an attempt to minimize the number of mass storage I/O calls (and therefore reduce latency time) by reading, or writing, with one call, strings of consecutive mass storage segments, rather than single segments. The size of the buffer pool can be independently varied, without affecting node processing.

Allocation of multi-segment strings takes advantage, where possible, of the track orientation of mass storage devices so as to reduce head movement time when the future corresponding I/O operations take place.

3.1.6 Example of User Request Processing

Let us examine the way in which a typical user request would be processed in STIS. We will assume that the request occurs in the form of a command from an on-line interactive user who has already completed the sign-on procedure, and has been executing a number of retrieval and update commands on various network entities. The flow of processing is indicated in Figure 3-2.

The user command "WHAT IS RANGE OF AARDVARK" is responded to by the user interface processing modules. The request is examined for syntactic correctness and completeness. The process of validation of the request (e.g., the determination that "RANGE" is, in fact, the name of an attribute and that "AARDVARK" is an entity name) is aided by the availability of suitable directory services. Directory services would also be available for the encoding of variable length external terms, such as AARDVARK, into fixed length internal term codes.

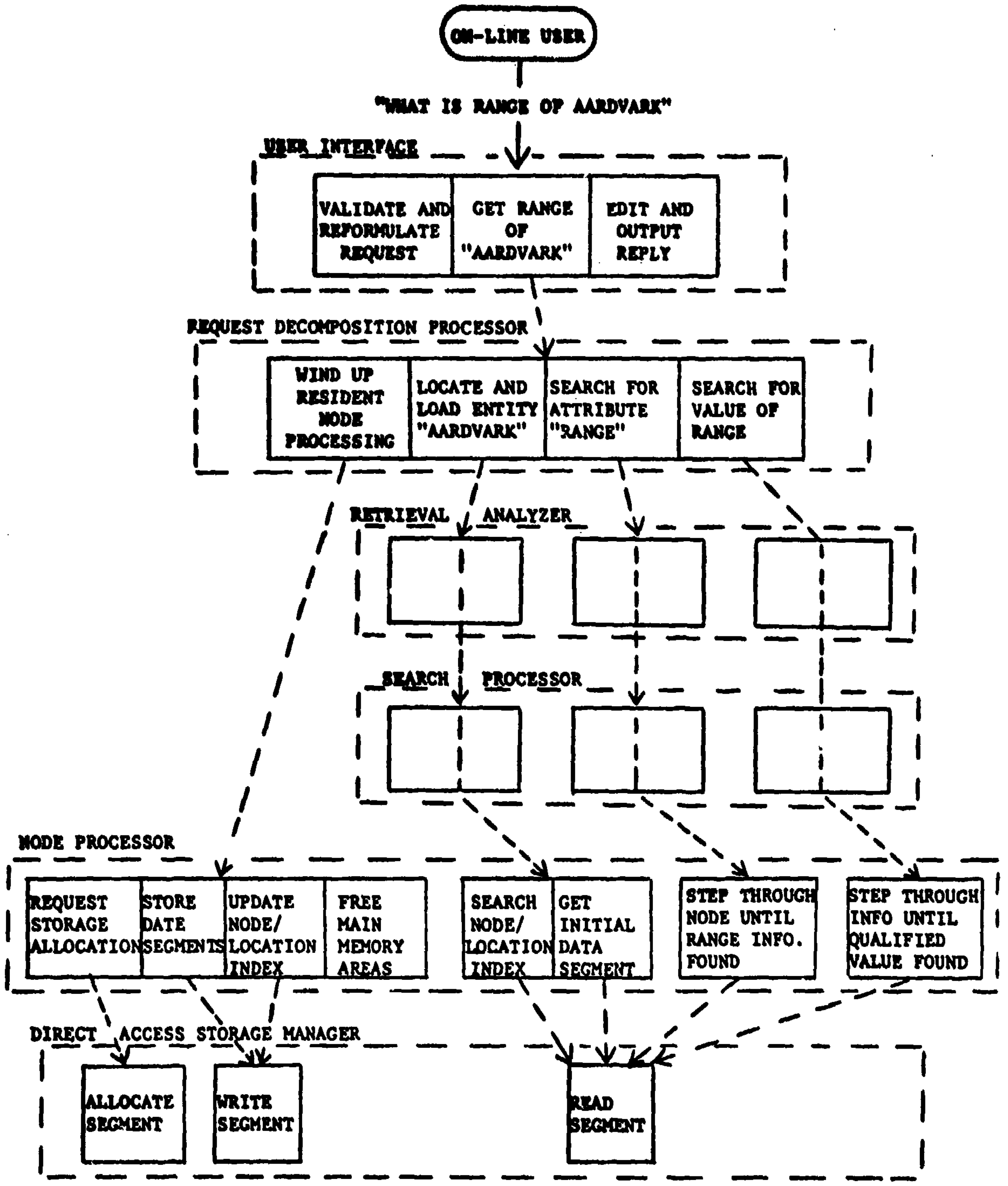

ON-LINE USER
"WHAT IS RANGE OF AARDVARK"
USER INTERFACE
VALIDATE AND REFORMULATE REQUEST
GET RANGE OF "AARDVARK"
EDIT AND OUTPUT REPLY
REQUEST DECOMPOSITION PROCESSOR
WIND UP RESIDENT NODE PROCESSING
LOCATE AND LOAD ENTITY "AARDVARK"
SEARCH FOR ATTRIBUTE "RANGE"
SEARCH FOR VALUE OF RANGE
RETRIEVAL ANALYZER
SEARCH PROCESSOR
NODE PROCESSOR
REQUEST STORAGE ALLOCATION
STORE DATE SEGMENTS
UPDATE NODE/ LOCATION INDEX
FREE MAIN MEMORY AREAS
SEARCH NODE/ LOCATION INDEX
GET INITIAL DATA SEGMENT
STEP THROUGH NODE UNTIL RANGE INFO. FOUND
STEP THROUGH INFO UNTIL QUALIFIED VALUE FOUND
DIRECT ACCESS STORAGE MANAGER
ALLOCATE SEGMENT
WRITE SEGMENT
READ SEGMENT

Figure 3-2 User Request Processing Example

After validation, the user command is reformulated into an internal form (e.g., term codes replace names) and a call is made to the next level.

Before processing the current call, the request decomposition processor must first ensure the completion of any processing on a previously-accessed node which may have been suspending pending subsequent user action. In the present case, let's assume that, before the current call, the user had created some other node to which he then appended a series of attribute/value pairs. After each update, the processing state was left in a posture to receive more updates for that node. User reference in the current call to a different node now requires that any processing associated with the created node (the currently resident node) be completed. As its first step, then, in responding to the current call, the request decomposition level directs the node processor to wind up processing on the currently resident node.

The node processor must perform at least four operations to complete processing on the current node. Since it must output the created node to permanent storage, it asks the direct access storage manager to allocate for this purpose the required number of mass storage segments. It then directs the storage manager to write the node data to the designated segments. The index, or location table, relating node identification to storage location is now updated. The windup operation is completed by making available for use all main memory areas that had been assigned to the processing of the created node.

When control returns, the request decomposition processor issues a call to locate and load (i.e., retrieve and make resident) the node representing the entity AARDVARK. All retrieve calls are issued to the retrieval analyzer. In this case, the retrieval analyzer determines that the request can be passed through without further processing to the search processor, which in turn decides that it is a simple retrieval request and relays it to the node processor.

The node processor determines the storage location of the desired node by searching the node/storage block index. If the search has been successful, it then requests the storage manager to read into main memory the initial node data segment and gives up control with an indication of successful completion. Control eventually returns to the request decomposition level.

Now assured that the node representing AARDVARK is resident, the request decomposition processor issues a call to the retrieval analyzer to locate the RANGE attribute for the resident node. Without further processing, the retrieval analyzer relays the call to the search processor, which determines that it is a simple retrieval request and passes it immediately to the node processor.

The node processor steps through the node data segments (issuing read requests to the storage manager for more data segments, as required) until it reaches the information packet for the RANGE attribute. Control returns to the request decomposition level, with an indication that the RANGE attribute has been located. (If node data had been exhausted before locating the RANGE attribute information, a "no find" indication would have been returned.)

The request decomposition processor now issues a call to retrieve the value for the located attribute. This simple retrieve call is passed through to the node processor, which steps through the attribute data until the properly qualified value is found and returned. (The method for determining the qualified value depends on whether the qualification is global for the node or specific to the value.)

After successful retrieval of the value, the request decomposition processor returns the value, with a successful termination code, to the online user interface. Here, the user interface, aware of the user's output requirements, formulates a reply message and transmits it to the user terminal.

3.2 MULTI-USER CONSIDERATIONS

Multi-user access to a single resident copy of STIS will be provided.

The exact organization of STIS so that it can serve "concurrently" more than one user depends on a large number of factors, many of which are as yet undetermined. These factors include:

(a) Physical Equipment and Operating System
(b) Programming Language
(c) Parellelism of STIS functions

3.2.1 Physical Equipment and Operating System

Basic to the organization of STIS for multi-user access is the nature of the physical equipment configuration and of the operating system services which support its use. These factors determine the way in which required system operations and data can be distributed among the hardware facilities, and define opportunities for parallel operation and for queued multi-user access.

For example, a small network of hardware processors and data storage devices may be desirable, in which one hardware subsystem plays the role of terminal input processor (including interactive communication with the user), another subsystem actually executes data update/retrieval functions, and a third subsystem serves a buffering/queuing function between the other subsystems.

Operating system services with respect to the registration, interlocking, queueing, and conditional activation of activities and subactivities will obviously play a crucial role in the design of STIS multi-user control.

3.2.2 Programming Language

The programming language used for implementation also affects the multi-user control aspects of system organization. The language, chosen for ease of implementation and/or to enhance transportability, may not, for example, provide for reentrant code. In the absence of reentrant code, concurrent users must be queued up to use the services of single copy portions of the system.

Whether or not reentrant code is used, a considerable amount of main memory storage space must be associated with each active user. Most of this space is used to hold data segments, data pointers and other parameters required for continuity between calls and to reduce or eliminate reaccess of already-accessed information. The chosen implementation language system may allow STIS to directly access this individual user space (treat it as its own space) or it may require that the contents of user space be moved into and out of STIS's own space for each user call.

3.2.3 Parallelism of STIS Functions

System organization for multi-user processing is affected by natural or induced parallism (or lack thereof) in the execution of STIS functions. Parallism here refers to:

(a) the degree to which subactivities required to carry out a particular user request can be executed independently and asynchronously with respect to each other, or

(b) the degree to which the subactivities carrying out a request for a particular user can be overlapped with those subactivities serving a request for a different user. For example, the updating of indexes as a result of an update operation by user A can be overlapped with the initiation of an update operation of a different data item by user B.

3.3 RECURSIVE PROCESSING

While many of the control requirements of STIS are similar to those of other systems, STIS differs most from these systems in its requirement for extensive recursive processing. That is, recursive processing in the execution of data access operations is a natural and primary component of STIS. Therefore, a considerable portion of the STIS design effort is concerned with the pushdown/popup stacking of control information and data associated with such internal recursion.

A requirement may exist to cope with external recursion, i.e., recursion with respect to the entire STIS system, where a call to STIS results in a call to a non-STIS process that in turn results in a call to STIS, etc. (see Figure 3-3a). This requirement can be met in one of at least two ways:

(a) On a call from STIS to an external processor or utility that may result in a subsequent call to STIS, a new user identification is generated (e.g., by a suffix on the original identification) to be used by the external processor in its STIS calls. STIS treates the first call from the external processor as the initial call for a new user, and assigns a separate work area to be used by STIS. (Of course, this "new" user must satisfy the same security and access checks as the original user.)

(b) Before executing a call to an external processor or utility that may, in turn, call STIS, STIS records the fact that the system is in a potential reentry state for that particular user. If such a reentry to STIS occurs, it is recognized, and appropriate measures are taken to allow the user's current work area to be used in satisfying the request. In either case, a stacking mechanism is required to retain certain information, e.g., exit address from STIS (See Figure 3-3b).

Non-STIS utilities or external processors that themselves are subject to recursion during this process must contain their own stacking mechanisms.

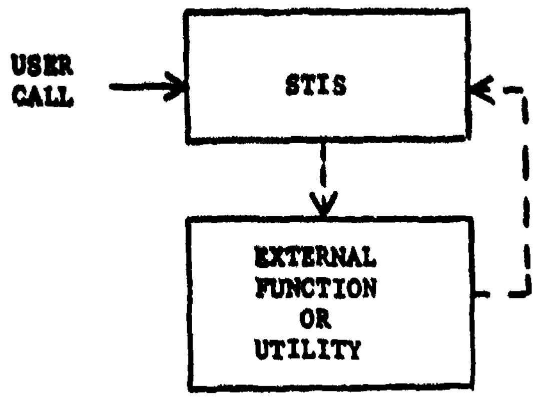


Figure 3-3(a) External Recursion

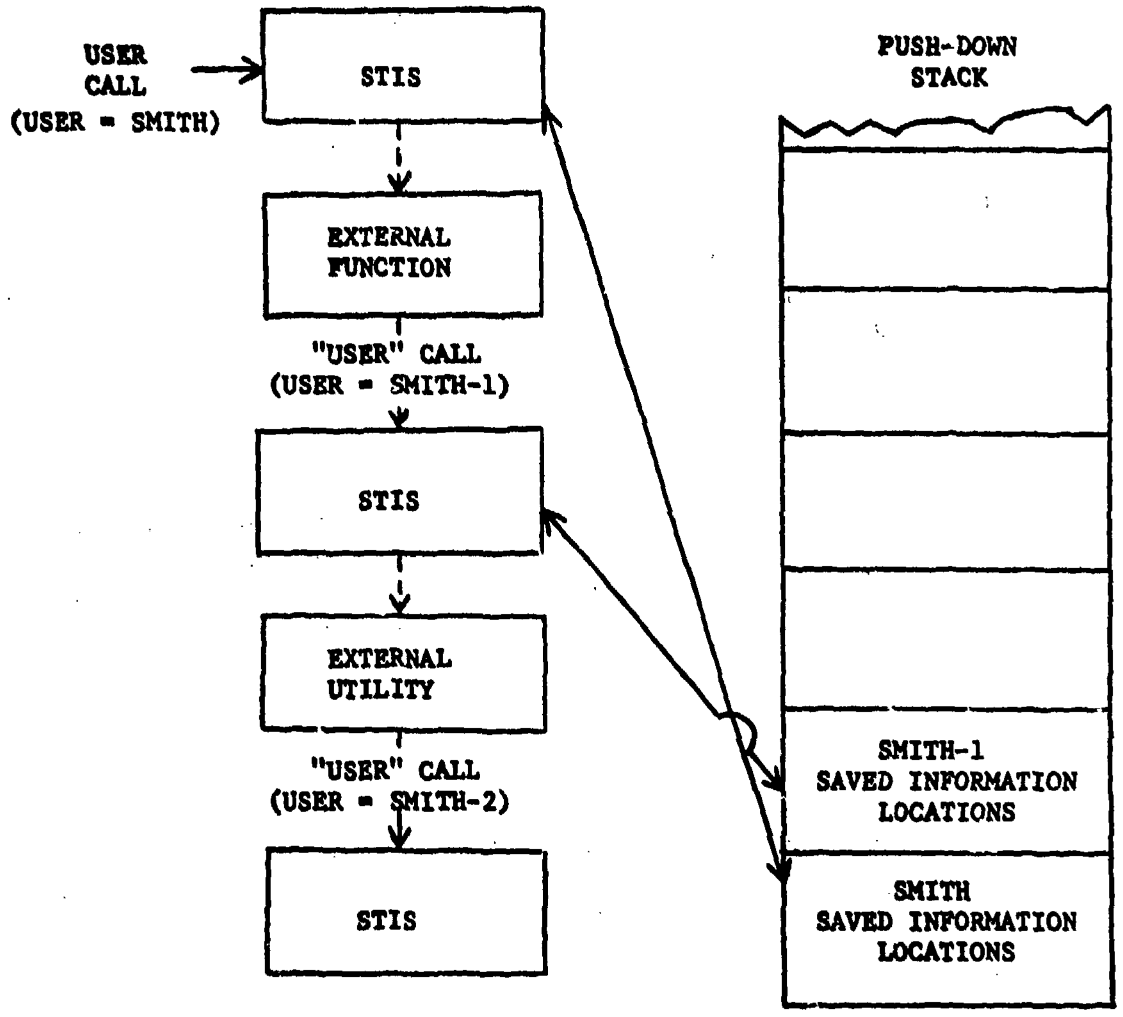


Figure 3-3(b) Multi-Level Recursion and Stacking

3.4 SECURITY PROCESSING

Security processing within STIS is concerned with the protection of the data base from illegitimate scrutiny and from uncontrolled modification or destruction. As in all such systems, such security processing functions at several levels.

At the outermost level, only certain terminals, or terminal groups, are permitted access to STIS, and the functions allowable to any particular terminal are strictly defined by the systems manager. At the next level, user passwords are associated with a particular Area of Responsibility (AOR), so that the AOR code, which is the key to further access, can only be invoked by legitimate users. At the next level, the AOR code, through restriction to a particular partition and/or fact qualification, limits access to certain data, and also protects that data from unpermitted access by other AOR's.

All such security procedures are governed by system tables accessible only to the system manager, and by the methods of fact qualification built into the system.

3.5 SYSTEMS MANAGEMENT FUNCTIONS

Systems management capabilities in STIS are intended to serve two main functions:

(a) Supervisory control of user access criteria

(b) Overall control of data base distribution and availability

3.5.1 User Access

Functions are provided to the systems manager to allow for the entry, modification and deletion of entries in user/terminal/password/access rights tables in STIS. The systems manager can add or delete terminals or terminal groups, or change the access privileges of existing terminals. He can add or delete specific users or change their access rights. The systems manager can display or print any information appearing in such systems tables, including user statistics, if they are maintained.

3.5.2 <u>Data Base Distribution</u>

The systems manager has facilities to allow for the definition of data base partitions, the movement of partitions on and off line, and the temporary disabling of partition access (see Section 4.7). **He maintains the configuration of available partitions based on normal usage requirements, and has the means to create special configurations to meet temporary emergency needs.**

SECTION IV. THE CONCEPT NET -- A NEW INFORMATION STRUCTURE FOR STIS

4.1 A MODEL FOR INTELLIGENCE INFORMATION

The Concept Net represents the intelligence analysts' collective view of the current state-of-affairs in the real world. It is populated with "facts" distilled by the analyst from observations, reports of observations, and assertions concerning his sphere of interest in that world. Since it is a dynamic world and viewed, as it were, "through a glass, darkly", each fact has associated with it an open-ended set of qualifying statements which include, typically. the source (or message) from which it was derived, its interval of validity (in time and/or space), the date of observation or entry into the system, the credibility (probabilistic truth value) assigned by the analyst, and the time-constant (or "half-life") which characterizes the volatility of the information. Because virtually all intelligence information is both of questionable veracity and subject to change, we view the original credibility level as being modulated by an exponentially decaying weighting function whose time constant is characteristic of the volatility of the type of information in question. For example, the place of employment of an individual may have a half-life of four years. That is, if it was reported in 1970 that George Murphy worked for RCA, and this "fact" was accepted with a credibility of 0.8, then in 1974, in the absence of any new data concerning Mr. Murphy, the credibility of that fact would be 0.4.

Another characteristic of intelligence information is that there may be conflicting reports concerning the facts about a given entity and/or legitimately differing views among one or more analysts as to what the facts may be, or, for that matter, more than one value for a given attribute may be valid in a given time interval. (The case may be that Mr. Murphy, while working for RCA, moonlights as an instructor for Rutgers University so that apparently conflicting reports on Mr. Murphy's occupation may be reconcilable, and coexist with a high credibility. On the other hand, the report of Mr. Murphy's employment at Rutgers may be a deliberate plant or "cover" to obscure the fact that he works for RCA.) For this reason, and to provide for simultaneous use of a common body of information among many analysts, the analyst or organization which is responsible for a given "fact" is recorded as part of the information qualifying that fact.

4.2 THE CONCEPT NODE

The Concept Net is organized as a network of nodes, each of which represents a concept (such as an individual or other entity) which is of interest to the analyst. The node in turn contains a set of facts (properties) made up of attribute names and qualified values, which describe the entity and its relationships to other entities. These facts are derived from (and tied to) messages concerning observations of the real-world. Other nodes may represent concepts which exist independently of messages (or observations of the world) such as semantic concepts representing the attributes and values themselves, as well as <u>their</u> inter-relationships. (Value nodes will also be related to the entities which are described by (or use) those values, providing a cross-index to the Entity Net.)

Each node in the Concept Net comprises an open-ended set of properties of the concept or real-world entity which is represented by the node. A property is an attribute-name/attribute-value pair which may, in turn, be qualified by an arbitrary list of properties. Attributes and values (also terms and words) are themselves represented by nodes in the Concept Net. An entity node may stand for a real-world individual, unit, facility, weapon, event, etc. A node may also represent a state or sub-entity attached to a parent node. For example, a parent node may represent a generic class of weapons, such as the Minuteman missile, while a sub-node may represent a specific example of that missile installed at a particular site, with a particular target, etc.

When a given entity or other concept node (the source) bears some relationship to another concept node (the target), that relationship is represented in what is called an entity-relational attribute in the source node. Its value is the identifier (Node #) of the target node. In order to provide complete cross-referencing, there will be defined for each relational attribute R (using its Attribute Node in the Semantic Net) an inverse relation R^{-1} so that if entity $\underline{a}$ bears relation R to entity $\underline{b}$ "R(a b)" then entity $\underline{b}$ bears relationship $\underline{R}^{-1}$ to entity $\underline{a}$ "R^{-1}(b a)". For example, if the Pershing Missile has a test site at White Sands "Has Test-site (Pershing Missile, White Sands)", then White Sands is the test site of the Pershing Missile "Is test-site (White Sands, Pershing Missile)" where "Inverse (has test site, is test site)" and "Inverse (is test site, has test site)". In the above example, the first two statements would be in the Entity Net (<u>Pershing Missile</u> and <u>White Sands</u> nodes, respectively) while the latter two statements would be in the Semantic Net (<u>Has test site</u> and <u>Is test site</u> nodes, respectively).

In addition to entity-relational attributes, an entity may possess attributes whose values are names, numbers, or descriptive terms which are not other entities. These values may be represented by nodes in the Semantic Net (rather than the Entity Net) which in turn cross reference, as entity (or index) lists, those entities which use them. Hence, the distinction between entity-relational and non-entity-relational attributes has little operational significance for search strategies in the system. In either case, the entities possessing a given property are accessible through the cross-referencing (indexing) feature, whether it be the node representing the target of an entity-relational attribute or the node (in the Semantic Net) representing the value of a non-entity-relational attribute. The entity list under the value node can be considered the inverse of the non-entity relational attribute in the entity node in which it occurs. The Concept Net provides for both an attributes-under-entity (normal file) and an entities-under-attribute (inverted file) point of view. This redundancy of access path -- sacrificing space for time -- is built by the system, under control of the Data Base Administrator (who may limit this redundancy selectively) and need not concern the analyst who chooses to limit his role to that of an information consumer.

4.3 SUB-NODES -- COMPOSITE ATTRIBUTES AND N-TUPLES

There will be instances in the Concept Net when it will be useful to consider one node as subordinate to another in a hierarchic sense (rather than the non-hierarchic, or coordinate, relationship between two nodes which are joined by an entity-relational attribute). When this subordinate relationship is defined, it implies the desirability to store the subordinate node so that it is physically accessible with the parent node, reflecting logical dependency and/or predictable access patterns. When this occurs, the subordinate node is called a sub-node of the parent, or master, node.

The sub-node relationship can arise in several contexts. In addition to the close master/slave relationship that may exist between two entities, mentioned above, a subnode may represent what is called a composite attribute, or n-tuple. A composite attribute is an attribute comprising a set (n-tuple) of simpler attributes. For example, position may be defined as a composite attribute comprising the simple attributes latitude and longitude, or address comprising number, street, city, and state. Composite attributes provide for generic terms which conveniently reference and retrieve a set of specific information. The analyst or programmer who is concerned about the structure of the Concept Net or is developing appropriate terminology for semantic concepts may work with the Data Base Administrator to define composite attributes or other sub-node relationships.

4.4 FORMAL DESCRIPTION OF THE CONCEPT NODE

Each Concept Node in the STIS Concept Net is represented by the same formal structure, called a description list. Entity Nodes, Attribute Nodes, and Value Nodes are all instances of Concept Nodes in STIS. Each Concept Code (or Node #) is the name of a description list. (The Node # will be used as a key to obtain the description list from permanent storage.) Subnodes are also represented by description lists but they do not have separate Concept Codes associated with them since they are stored with the parent node. A composite value (the value of a composite attribute) is a special case of a sub-node in which the attributes have been predefined.

A Concept Node in STIS is a description list. The formal syntax of a description list is specified in Table 4-1. Lower case letters represent syntactic variables and upper case letters represent concept codes or other terminal atomic symbols.

There is no syntactic distinction between brackets and parentheses. Note that a description list is defined recursively so that there is no constraint on the nesting of subnodes representing qualifiers or composite values.

4.5 THE SEMANTIC NET

The Semantic Net is that subset of the Concept Net comprising Attribute and Value Nodes. All attributes and values are represented by nodes in the Semantic Net. (In the case of numeric values, the Value Node represents an interval on a lograrithmic or linear scale.) When they occur in the description list of a node in the Entity Net, attributes and non-numeric values are represented by their Concept Codes (Node Numbers).

TABLE 4-1

NODE STRUCTURE SPECIFICATION

Note: The convention used here for syntax specification uses the following metalinguistic symbols:

←	is defined as, or can be replaced by
⌊ ⌋	one or more occurrences of the expression enclosed by the lower half-bracket
|	choice symbol
⌈ ⌉	optional (at most one occurrence) of the expression enclosed by the upper half-brackets

Syntax

```
deslist ← [@ ⌊prop⌋]
   prop ← (A val)
    val ← V | [⌊val⌋] | (val qual) | deslist
   qual ← [* ⌊prop⌋]
```

Semantics

deslist	=	description list
prop	=	property
val	=	value
A	=	Attribute Code (i.e., Node #)
V	=	Value Code (i.e., Node #), numeral, or string representing a terminal value.
qual	=	qualification list
[V ...]	=	array (list) value
(val qual)	=	a qualified value; qual is the subnode which qualifies val
(A deslist)	=	a composite property; A is the composite attribute and deslist is the sub-node representing the composite value.

TABLE 4-1 (Continued)

Examples

The following description list examples represent Entity Nodes. In the interest of clarity, attribute and value names are used rather than Node Numbers.

```
#1 = [@(Name [@(First Jerry) (Last Sable)] )  (Age (45 [*(Source Est)
     (Accuracy ± 3) (Validity-interval 1975)]))
     (Works-at #2)]

#2 ≠ [@(Name AAI) (Fac-type Consultant-org) (Employs [#1 #3 #4])
     (Location ([Phila Wash] [*(Cred 0.90]))]

#3 = ([@(Name Schernecke) (Works-at #2)] [*(AOR Consultants)])

   = ([@(Name McCrea) (Works-at(#2[*(Validity-interval [1963 1975])]))]
     [*(AOR Consultants)])
```

4.5.1 Attribute Nodes

The description of any concept consists of a list of properties, i.e., attribute name/value pairs. Since attributes are concepts themselves, they are represented by nodes in the Semantic Net subset of the Concept Net. Some of the attributes which can be expected to be used in the description list of an Attribute Node are listed below. (It should be noted that as in all nodes, these attributes, except when they are self-referencing, are represented by the Node Codes of Attribute Nodes. Their values are represented either by Node Codes or by Term Codes.)

Attribute name

Synonyms

Narrower attributes (for composite attributes)

Broader attributes (for components of composite attributes)

Inverse attribute (for Entity Relational attributes)

Values (the list of values for this attribute, limited to the first domain element in the case of Entity Relational attributes)

Attribute Data Information -- the value of this attribute is a pointer to the Attribute Data Record in a Direct Access file outside of the Concept Net. The ADR defines the format, precision, units, and "owner" of the attribute. This is an example of a special attribute, or Process Hook, which invokes an outside routine to compute a complex value, using the nominal attribute value as a parameter.

Other Attribute properties, such as transitivity, reflexivity, and symmetry which may exist will also be represented in the property list of the Attribute Node.

4.5.2 Value Nodes

Each non-numeric value, or range of numeric values, which can serve as a retrieval condition will be represented as a Value Node in the

Concept Net. When indicated by the analyst, or Data Administrator, the Value Node will serve as the head of an index to information in the Entity Net. This provides support for the three basic strategies for retrieving information about intelligence entities:

(1) through the context of an explicitly identified entity, including its association with other entities via relational attributes,

(2) through a retrieval criterion made up of a set of specified properties which the entity should possess, and

(3) through properties which are plausible for the entity because they can be inferred from generalized rules stored in the Concept Net.

Some of the attributes which can be expected to be used in the description list of a Value Node are listed below:

Value name

Synonyms

Narrower values (or subsets)

Broader values (or supersets)

Attribute (the attribute that has this node as a value, the inverse of the Values attribute in the Attribute Node)

Entities (the entities which have this node as a value. This serves as the index list for those entities.)

4.6. THE ENTITY NET

Information about any intelligence entity of concern to the analyst can be stored in STIS by creating an Entity Node to represent it in the Concept Net. Once the node is created, the description of the entity is stored as a list of properties. Internally, the entity is known by its Node Number, which serves as its retrieval key from permanent storage, as is the

(This page intentionally left blank)

case for any node in the Concept Net. In its simplest form, the entity number n is represented by a description list such as:

n = [@ (A a) (B b) (C c) ...]

The interpretation is that the entity represented by node n has all of the properties listed. That is, in conventional relational or logical format, the attributes A, B, C, ... are binary relations connecting the entity and a value and the following conjunction holds:

$$A(n,a) \wedge B(n,b) \wedge C(n,c) \wedge \ldots$$

Thus, in the Entity Net, information is collected in an "attributes-under-entity" format, while in the Semantic Net, one may say that the same information appears in an "entities-under-attribute" format. As will be discussed below, the simple description list form can be generalized in a number of important ways.

4.6.1 Entity Relations

The simplest relations are attributes which take scalar values, either literal or numeric, such as Name(n,Atlas) and Weight(n,150). However, values are generalized to permit arrays, such as Name(n,[Atlas,M12]) and Location(n,[ND,FL]). Assuming Node n is #10, this would appear in description list format as:

#10 = [@(Name [Atlas M12]) (Weight 150) (Location [ND FL])]

Entity-relational attributes name other Entity Nodes as values. If entities #11 and #12 were test sites for #10, then the update command "Add Test-Site (#10, [#11, #12])" would add the property (Test-Site [#11 #12]) to the description list for #10.

By permitting a value to be represented by a description list, or subnode, the descriptive power of the system is augmented in a number of ways. The simplest instance of this, the composite attribute, was described in Section 3. Other cases will be discussed in the following paragraphs.

4.6.2 Generic Entities

It is often useful to describe an object as a generic type for which, in the real world, there exists a number of specific occurrances. This can be done by creating a node, called a generic entity, which represents the common characteristics for these objects. This can then be supplemented by a node for each individual object for which specific information is required but which is not characteristic of the class as a whole. For example, suppose we have the missile type Atlas represented by:

```
#20 = [@(System ICBM) (Name Atlas) (Weight 150)
  (Accuracy 3) (Instances [#21 #22 #23]) ]
```

Nodes #21, #22, and #23 then are specific entities whose general characteristics are given in node #20 and therefore may be inferred by reference and need not be explicitly repeated. Each instance will reference the generic entity and give only unique characteristics, such as:

```
#21 = [@(Location ND) (Target #31) (Serial 1234) (Generic-entity #20)]
```

Note that the attributes "Instances" and "Generic-entity" are a converse pair.

4.6.3 Entity States

It is often necessary to track changes in a given set of properties of a specific object. To do this, subnodes called "states" are created. The relationship between a specific entity and a state of that entity is parallel to that between a generic entity and a specific entity. That is, only properties whose values change from one state to the next need be recorded. Invarient properties are given in the parent node. For example, suppose a Polaris type submarine is being tracked. Intermittent reports of its location may be given in state nodes which reference the specific entity node. The specific entity node may, in turn, reference a generic entity. This interrelationship of subnodes is diagrammed in Figure 4-1.

The recurring motive for introducing subnode relationships such as "instance" and "state" is to avoid redundant storage of information. The payoff for eliminating unnecessary redundancy is reduction of maintenance and retrieval time as well as space. Storage compression at the state level can be carried to a further stage when changes in state are predictable or can be represented analytically as a function of time. Opportunities for this may exist in situations such as when a periodic itinerary for a submarine or other ship is known, or when a satellite position may be found from orbital parameters rather than extrapolation or interpolation of tracking data. In such cases, state nodes may be replaced by compact state-transition information.

4.6.4 Fact Qualification

It is possible to modify or qualify information by appending a qualification list to either a description list (node or subnode) or a value. The qualification list has the format of a description list so that the two forms are respectively (deslist qual) and (val qual) where the second element is the qualification list. Typically, qualification information in an Entity Node will contain fact control (access control) information if it is at the node level and fact control and/or source, credibility, and temporal data at the value level. Because information may be obtained from several sources and may be varying with time, multiple values will be common in the Entity Net. The particular values which are valid for a given analyst at a given time will be determined on the basis of the qualification list.

The default interpretation of the property (A v) for an entity (say e) is that the entity has the value v for the attribute A. In symbols A(e) = v. The value v may either be a scalar V or an array [V...]. However, there

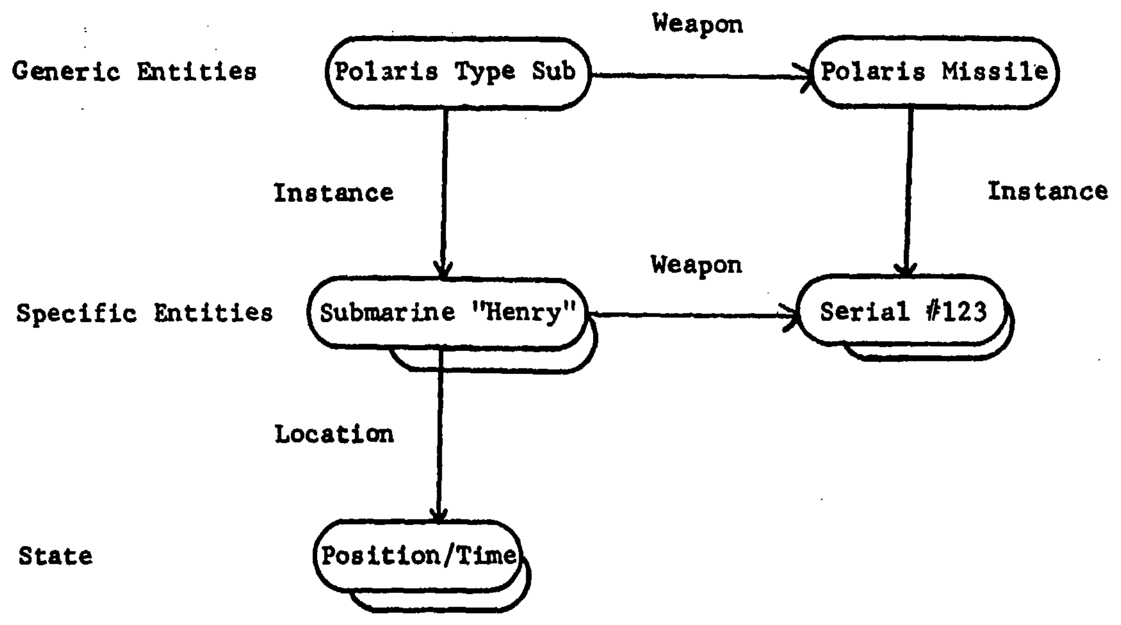


Figure 4-1
Subnode Relationships

are occasions when one wants to specify a relational operator other than equality between the attribute and the value. Possible relations are greater-than, less-then, not-equal, approximately-equal, not-greater-than, etc. The qualification list is also the mechanism for accomplishing this, with the exception operator attribute "Rel-op". For example, Age(e)>40 would be given as (Age(40[*(Rel-op >)])).

4.6.5 Computed Values

There will be instances when it is more convenient to compute a value for a given attribute from specified parameters rather than explicitly store its value. This will be especially true for large arrays of composite attributes. For example, it will often be more efficient to compute the position, velocity, acceleration, etc. of a missile from trajectory, atmospheric and vehicle parameters rather than store explicit values with the required precision. Even where analytic computation is not practical, it is often more efficient to store values in large dense arrays or conventional files (on serial or random access storage) and provide the appropriate file name or key in the description list. Another example of the latter situation is the Fact Control Information required for most entities and properties. Because this data can be readily formatted into fixed files, it may be more efficient to provide a key to a Fact Control Data File in the qualification list pertinent to the basic information, rather than provide that data in description list format.

This capability will be accommodated by using a special "Process-Hook" symbol and parameter list in place of the actual value in the description list. The retrieval mechanism, when encountering the Process Hook, will invoke the specified program and supply the given parameters. The called program will return the required value.

4.6.6 Quasi-transitive Relationships

The use of entity relational attributes in the description of the various objects of interest to the analyst results in a network of nodes in which information is highly associated. This richness of association permits information to be retrieved from many points of view or search paths. Although this feature is, in general, desirable, unless special precautions are observed, there are situations in which it can lead to the retrieval of information which does not validly meet the conditions specified by the interrogator.

Consider, for example, a situation in which a weapon platform (say a fighter-bomber) can be equipped to bear either of two types of armament (say torpedo or incendiaries) depending upon under which service unit (aircraft carrier or tactical air base) it is employed. A given entity relational attribute (such as "uses") may be used to enter this information:

$$\left\{\begin{array}{l}\text{Uses (Carrier Lexington, F-11)}\\ \text{Uses (F-11, torpedos)}\end{array}\right.$$

$$\left\{\begin{array}{l}\text{Uses (TAC Base Charlie, F-11)}\\ \text{Uses (F-11, incendiaries)}\end{array}\right.$$

The five entities would then be interconnected with the "Uses" relation as shown in Figure 4-2. It is apparent that a request for armament used by the Carrier Lexington (or TAC Base Charlie) may come up with the erroneous answer "incendiaries and torpedos". The fallacy is caused by what can be called a "connection trap" in the F-11 "hub" of the network. It is avoided by using one or both of the following devices:

(1) The set of values of a multivalued attribute are qualified to inform the system that only one of the values can occur in each instance.

(2) A configuration node (or subnode) is created to describe each valid configuration of properties.

These approaches are detailed below.

Since an attribute may have an array as a value, we can have a property such as:

(Armament [torpedo incendiary])

in the description list for an entity (say F-11). This raises the question as

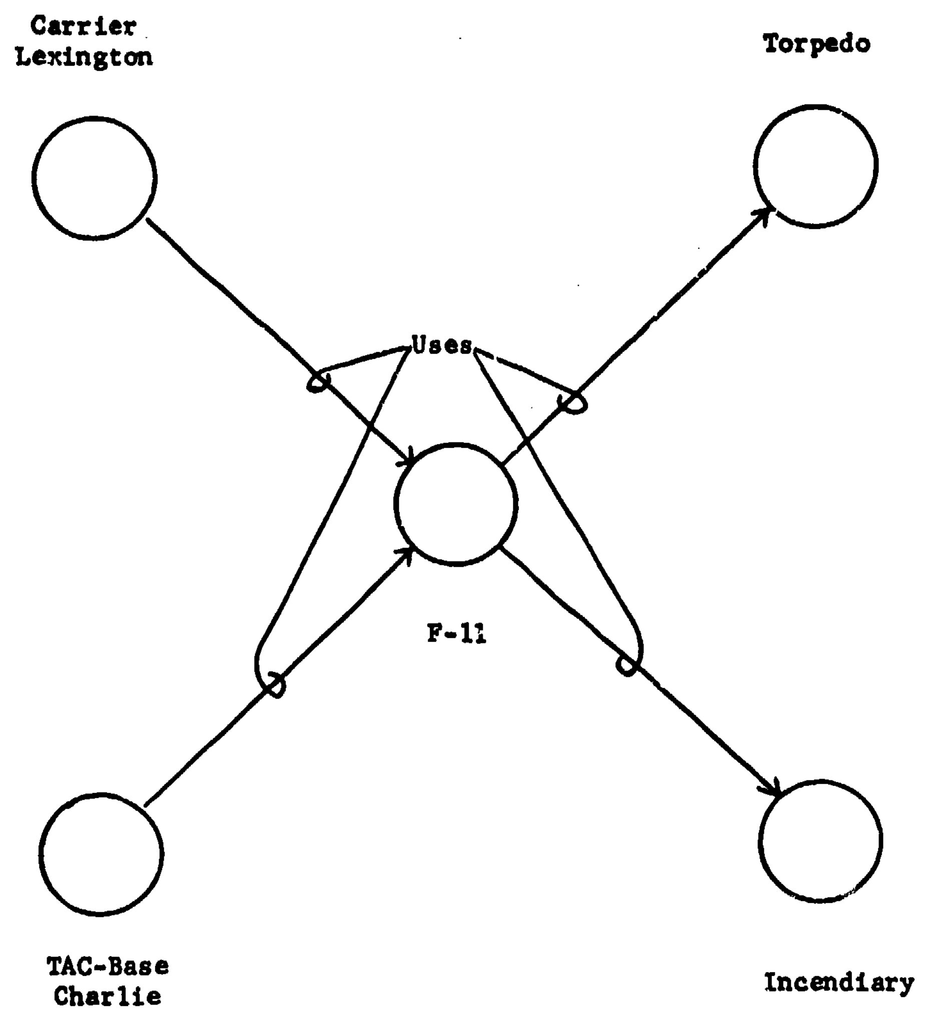


Figure 4-2
Connection Trap

to the interpretation of the array:

$$v = [\ v_1\ v_2\ \cdots\ v_n\ ]$$

when it occurs as a value. The members v_i may be an ordered n-tuple, an (unordered) set, a bag (unordered set in which repetitions are permitted), a disjunctive set (any subset is valid), a conjunctive set (all values co-occur), or a choice set (only one value is valid in each instance). The type of set which is intended can be identified by using the attribute "Set-type" in a qualification list for the value. For example:

(Armament ([torpedo incendiary] [* (set-type choice)]))

The use of the "Set-type choice" qualifier alerts the system (and the user) that only one value is valid but in itself is not sufficient to specify which is the valid value in a specific case. This problem can be solved by using a subnode (or a state) of the entity to establish a description of each configuration of the parent entity. For example, we can have the states

[@(Used-by Carrier-Lexington) (Armament torpedo)]

and

[@(Used-by TAC Base Charlie) (Armament incendiary)]

under the generic entity for the F-11. Note that this second approach avoids the multiple values attribute and is sufficient in itself to unambiguously describe the situation.

4.6.7 Footnotes

The analyst entering facts into the Entity Net will be permitted to qualify any value (or entity) with unformatted comments, warnings, or other text. He simply labels this text (generically called footnotes) with the appropriate attribute (Comment, Warning, etc.) and enters it with other qualification information. Rather than store unstructured text as part of the node, a special use will be made of the Process Hook capability. The value of the specified attribute will be a pointer to the appropriate record in an external Foot note File. The footnote will be retrieved automatically with other qualification information whenever required.

4.7 CONCEPT NET PARTITIONS

Each fact or node in the Concept Net will belong to some subspace of the Concept Net called a partition. Partitions will form a lattice or partial ordering so that a given partition may in turn contain other partitions. The objectives of partioning the Concept Net are as follows:

(1) to establish sets of information (at the highest level) which should coexist on the same level of physical storage because of access patterns or ownership (Areas of Responsibility),

(2) to establish sets of mutually self-consistent (or "coherent") facts, rules, and credibility measures,

(3) to establish a space in which facts form a set of homogeneous type, such as and, or, nand(not-and), choice, etc., i.e., to establish the scope and semantic of a set,

(4) to establish the scope of quantification in general statements or rules.

The partition to which a fact belongs will be identified as a property in its Qualification List. When all the facts in a given node (or subnode) belong to the same partition it may be "factored out" to become a property at the node (or subnode) level. The partition may be a property of the node and also of one or more facts in the node. In that case the partition established for the node applies to only those facts which are not qualified by membership in another partition. In these regards, the partition behaves in the same way as other properties of a fact or node. The use of partitioning in semantic networks has been described by Hendrix [5] and its use here in the STIS Concept Net is quite similar.

4.7.1 Physical (Top-level) Partitioning

Partitioning of the Concept Net into physically independent sections at the highest level is provided to allow efficiency of local referencing within an application area, and to allow the systems manager to make timely allocation of potentially scarce resources to such application areas.

The partitioning criterion (such as Area of Responsibility) is applied during node processing. In simple cases, the partition reference number can be determined by reference to a table, such as that shown in Table 4-1(a). The partition number is used in all calls to the Storage Manager. The Storage Manager is unaware of the criteria by which the partition number is determined.

Physical partitioning of data, and associated loading and unloading of complete partitions, is supported at the storage managment level through a number of devices:

(a) A partition may consist of a complete file, or a single file may be divided into a number of partitions.

(b) Separate direct access storage allocation maps are associated with (and physically reside in) each partition.

(c) Allocation requests, and I/O operations, are associated with appropriate physical file space through the use of a partition status table. (see Table 4-1(b))

(d) The partition status table, in addition to information about currently "resident" physical partitions, contains information about the characteristics, locations, and status of "off-line" partitions.

(e) Facilities are provided to the STI system manager for the rollin or rollout of partitions between on-line mass storage files and off-line lower levels of storage. Suitable interlocks are provided during such operations.

(f) The STI system manager can make inquiries with respect to partition status information (including, potentially, usage statistics) and can update certain partition status table elements.

Intra-partition references are made via a relative block number (or its equivalent). References within a node to another partition are accomplished through a reference pair consisting of a partition number and a relative block number within that partition.

AOR	PARTITION NUMBER
1	1
2	3
3	2
4	3
5	4
6	1
7	1
8	5
9	6

(a) Partition Reference Table

	ON-LINE LOCATION				PERMANENT LOCATION			USAGE STATISTICS	
ENTRY NUMBER	FILE NAME	BLK. RANGE	ON LINE	LOCK	FILE NAME	BLK. RANGE	TYPE	FREQ.	LAST ACCESS
1	DBF	ØØ-4599	X		DBF	ØØ-4599	R	NN	D/T
2					DA1	ØØ-3999	R		
3	DBF	87ØØ-	X	X	TP1	---	S	Ø	Ø
4	DBX	ØØ-9999	X		DBX	ØØ-9999	R	NN	D/T
5					TP2	---	S		
6	DBF	46ØØ-8699	X		DBF	46ØØ-8699	R	NN	D/T

(b) Partition Status Table

Table 4-2 Partition Tables

4.7.2 Consistency and Context Set Partitions

Partitions can be used to establish the set of facts which belong to a particular analyst, or which are valid or consistent with respect to a given criterion. For example, suppose we have two sets of facts, B1 and B2 (not necessarily disjoint), which are consistent with stated criteria C1 and C2, respectively. In Figure 4-3, B1 and B2 point to partitions S3 and S4, respectively, and indicate their membership in consistency sets C1 and C2. B1 and B2 identify the partitions as conjunctive sets, that is, the following are considered independent sets of consistency statements:

$$B1 = \{ WOba, WOde, WAbc, DAec \}$$
$$B2 = \{ WObf, WOde, DAec, DAfg \}$$

4.7.3 Use of Partitions in Representing Rules

The Concept Net must have the capability to represent not only explicit sets of facts, but also general rules which are unambiguously quantified variables in specific domains. For example, we wish to be able to state that if a person works on a system which is developed at a particular plant, then that person works at that plant. Symbolically, for any x,y, and z,

$$WOxy \wedge DAyz \Rightarrow WAxz$$

This can be represented in the Concept Net several ways. One way is an implication statement (as above). An equivalent representation is the Nand (Not-and) form

$$(\uparrow WOxy\ DAyz\ \overline{WAxz}).$$

This latter representation is closely related to the (disjunctive) clause form proposed for the inference mechanism:

$$(\vee \overline{WOxy}\ \overline{DAyz}\ WAxz)$$

The implication statement form of the rules is shown as both a Net Graph and a Node Diagram in Figure 4-4. The Nand form of the rule is shown in Figure 4-5. Both figures illustrate the essential use being made of partitions to establish the scope of sets of various types.

Consistency sets made up of rules and facts are further illustrated in Figure 4-6.

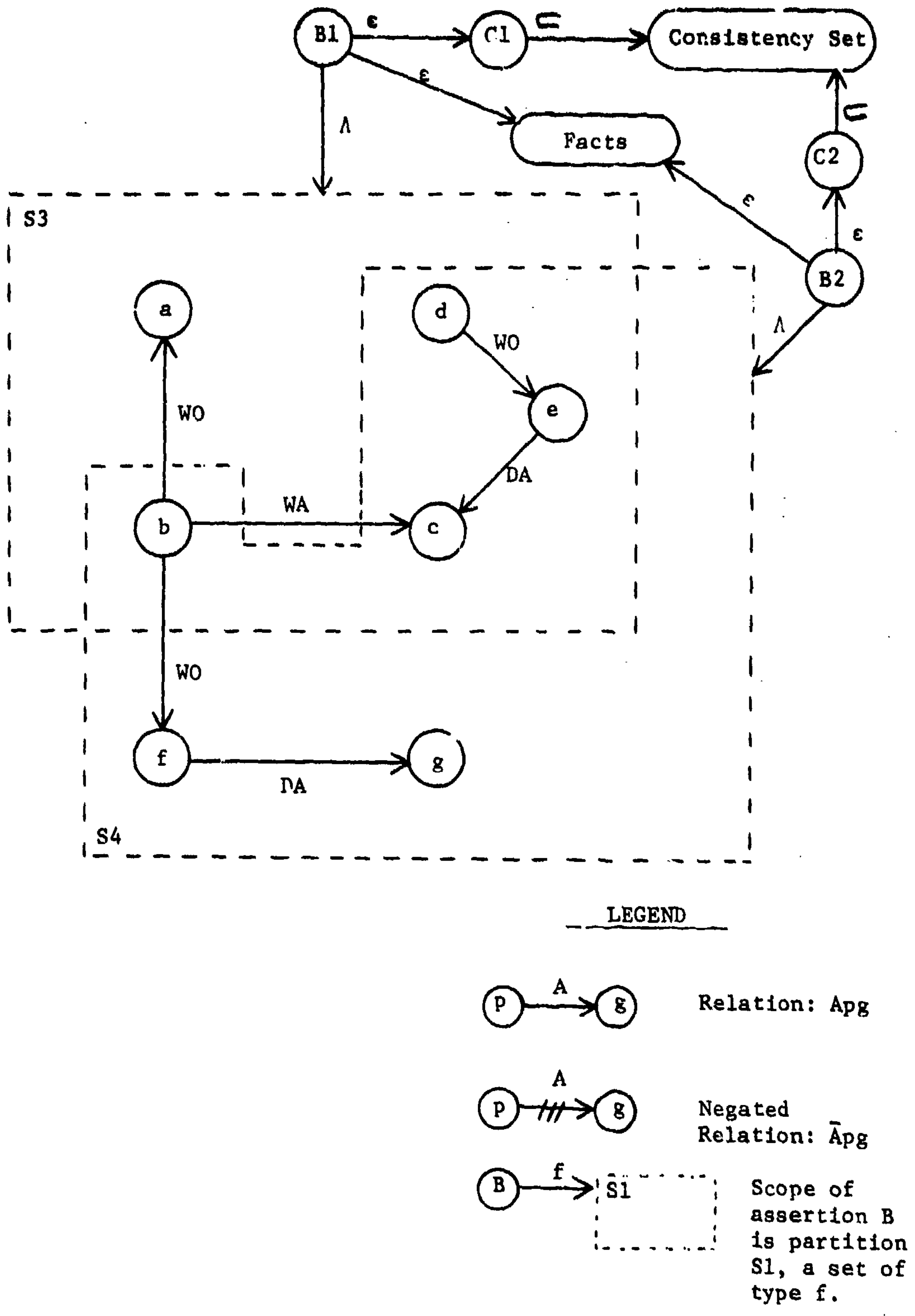
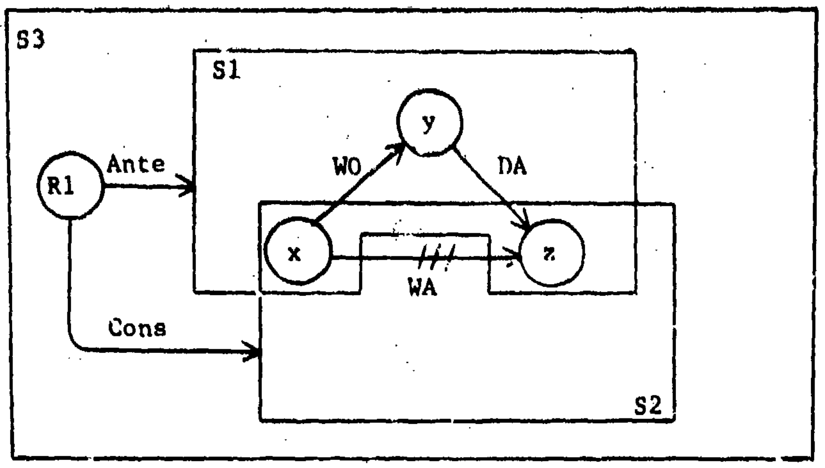
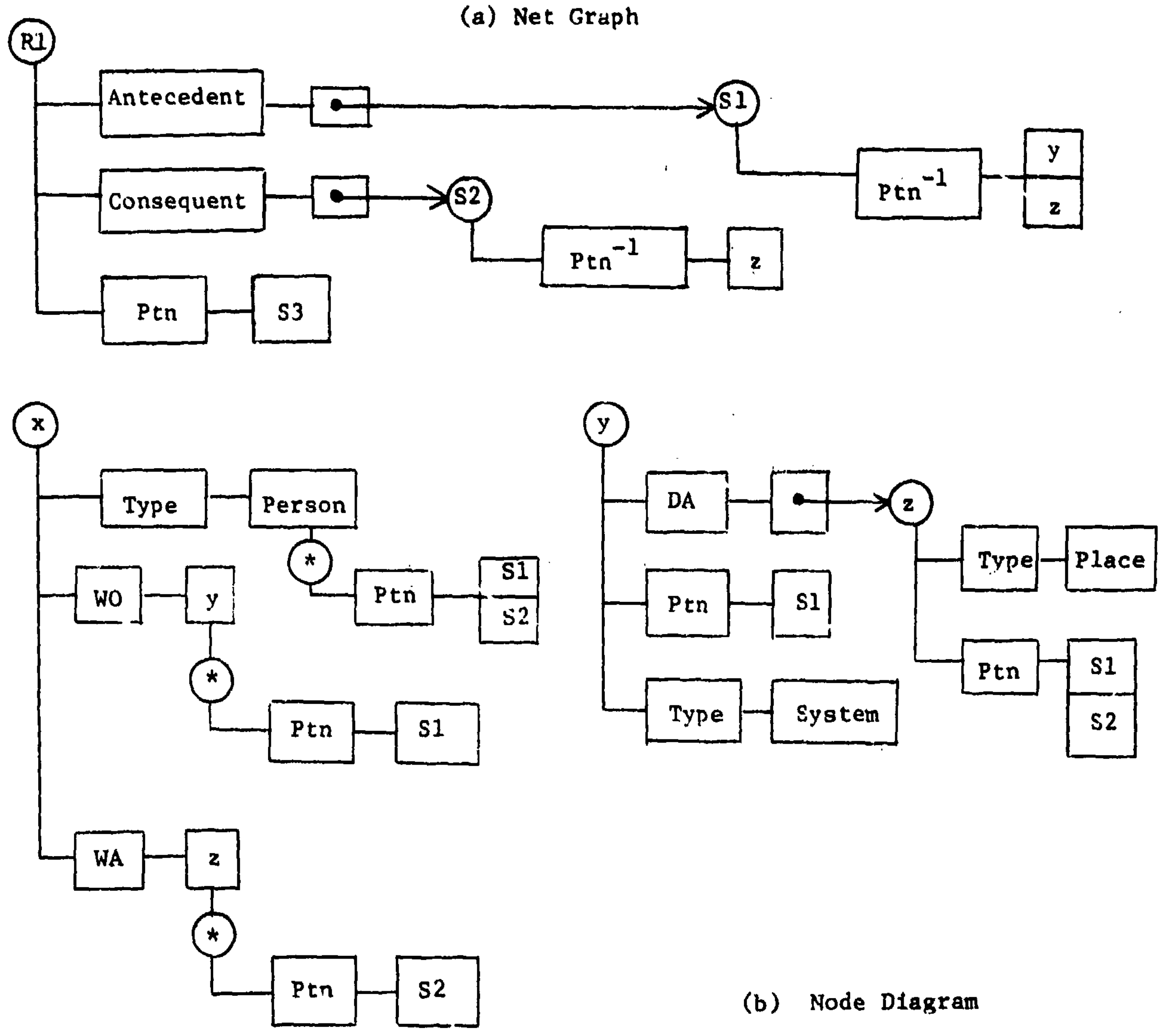


Figure 4-3 Fact Partitions Example

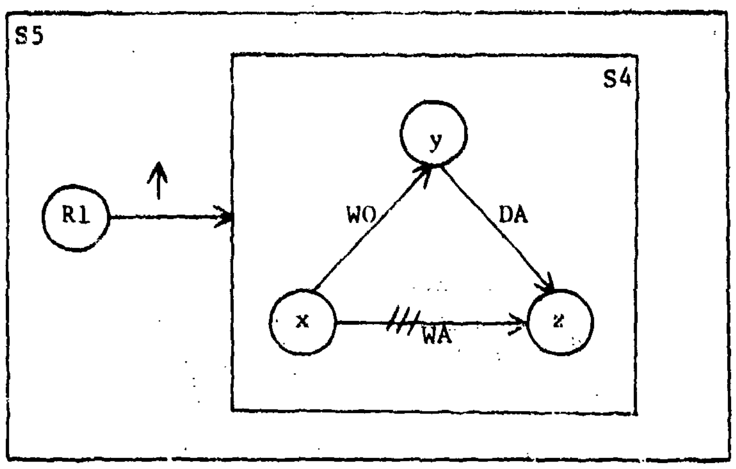


(a) Net Graph

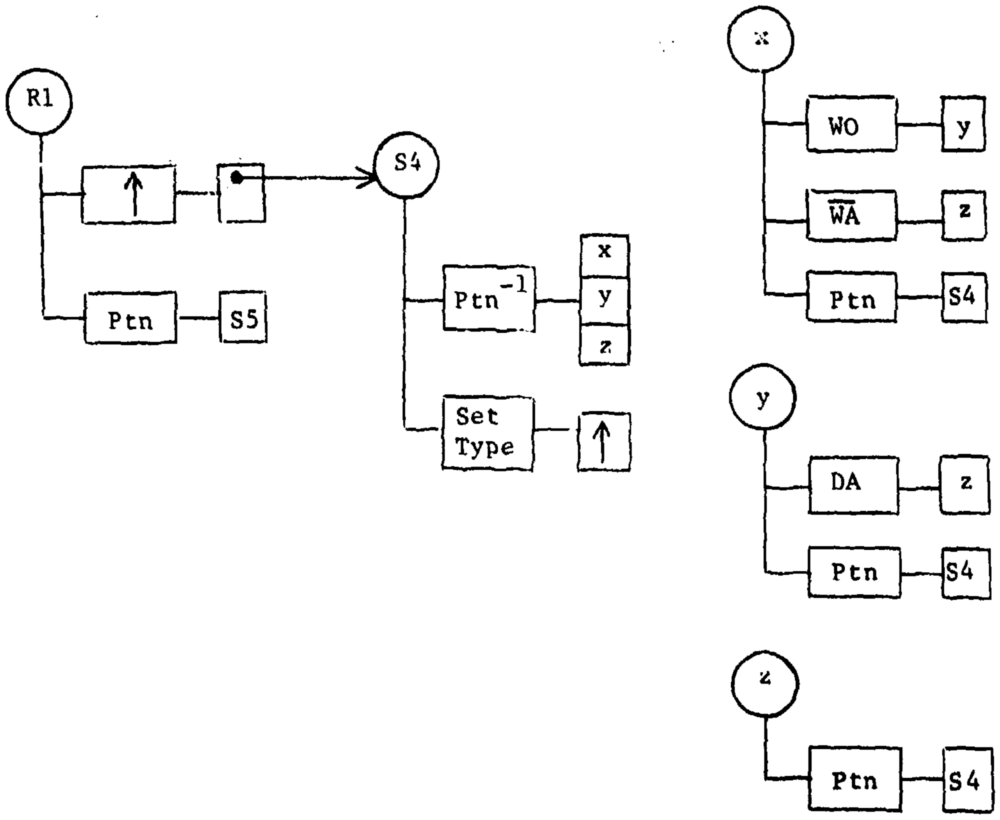


(b) Node Diagram

Figure 4-4 Representation of Rule: WOxy $\wedge$ DAyz $\Rightarrow$ WAxz



(a) Net Graph



(b) Node Diagram

Figure 4-5 Representation of Rule: (↑WOxyDAyz$\overline{\text{WA}}$xz)

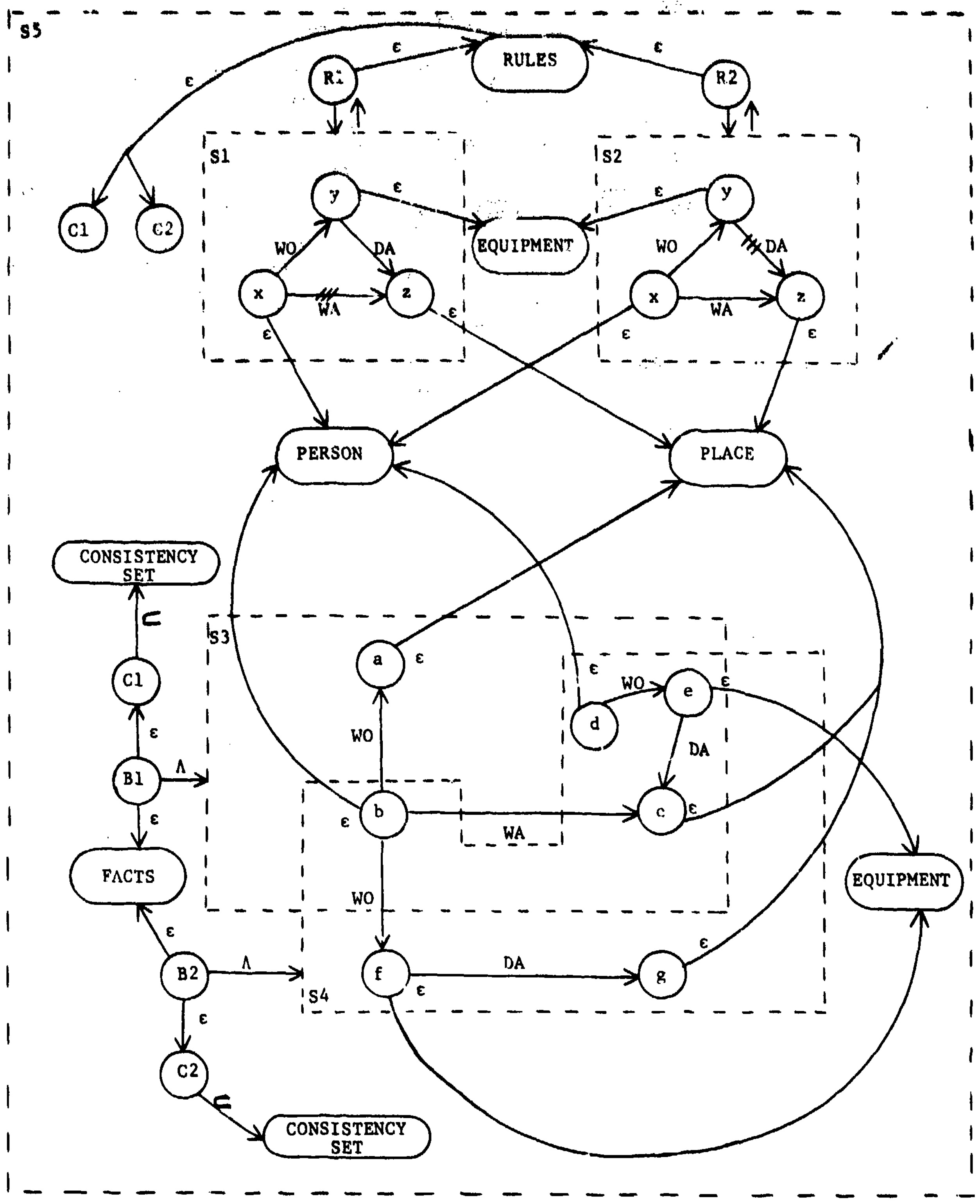


Figure 4-6 Partitioned Concept Net Example

4.8 REPORTS

The reports made to the information system may be regarded as the foundation for all the intelligence information of the system. Because of the need to protect this initial fundamental information, which comes to the system in many ways, it is stored in a Report Net whose structure may have significant differences from the Entity Net in which the analyzed and reduced information of the system is stored.

In addition, the particular procedures that the analyst or Data Base Administrator employs to maintain and modify the Entity Net information in the light of new reports are very varied. It appears important to maintain a well protected record of system reports so that far-reaching reorganization of the Entity Net can be effectively accomplished. Frequent changes are expected about what information is important, and what system techniques are practical.

4.8.1 Report Net Structure

It is expected that reports will be stored in the text in which received, as protection for the accuracy and completeness of the report information and maintained in time sequence, as a historical log. In order to provide reasonable search and association capabilities, the analyst will provide a description list in Concept Node format, characterizing the report and its salient information, thus forming a report header. For example, a missile site firing report may contain a bulky set of trajectory information, and also identification and summary information, such as a range capability estimate completed from the trajectory data. At the discretion of the Data Base Administrator, the above discussed Process-Hook capability may be employed, treating the original bulky report as a fixed file.

On the other hand, the report identification and qualification information, together with certain computer values, may be stored in time sequence, but with a structure like that of a Concept Node, in order that search and association capabilities be available. Depending upon search and operating conditions, it may be practical to maintain separate logs of reports dealing with different classes of information, with appropriate cross reference features. Also, at the discretion of the Data Base Administrator, references to outside sources such as documents, recordings, witnesses, or photographs may be included with the identification and qualification information. Occasionally, such information may be more illuminating than the original report details.

4.8.2 Report Procedures

The intelligence maintenance, search, and interpretation problems are reflected in both the structure used for the reports, and also the procedures employed in incorporating the report intelligence throughout the Concept Net. Great variability is anticipated in these procedures, and it appears that valuable insights into the practicality of data organization questions can be obtained from considering such procedures. This appears to apply to both the Report Net structure and also the Entity Net structure.

First, there is the considerable variety of anticipated reports. There may be a report whose impact on the intelligence system may be slight, consisting of a routine updating of some estimated value appearing once or twice in the Entity Net. Alternatively, a report may appear which leads to the establishment of new Entity Net facts, or the reorganizing of old ones. It is possible that the simple absence of certain reports may carry more intelligence than those that actually arrive during a certain period of time.

Second, there is the relative importance of maintenance vs. retrieval in the operation of the system. One viewpoint emphasizes the maintenance and construction aspects of the information system (updating the Entity Net from new reports), since it follows the natural flow of information processing and distillation. The other viewpoint emphasizes the information search and interpretation aspect of the system (retrieval of Entity Net facts) which may comprise the bulk of the system activity. However, a balance must be sought since this activity is dependent upon an effective information maintenance and construction procedure in utilizing the fundamental information which flow from the reports.

4.8 REPORTS

The reports made to the information system may be regarded as the foundation for all the intelligence information of the system. Because of the need to protect this initial fundamental information, which comes to the system in many ways, it is stored in a Report Net whose structure may have significant differences from the Entity Net in which the analyzed and reduced information of the system is stored.

In addition, the particular procedures that the analyst or Data Base Administrator employs to maintain and modify the Entity Net information in the light of new reports are very varied. It appears important to maintain a well protected record of system reports so that far-reaching reorganization of the Entity Net can be effectively accomplished. Frequent changes are expected about what information is important, and what system techniques are practical.

4.8.1 Report Net Structure

It is expected that reports will be stored in the text in which received, as protection for the accuracy and completeness of the report information and maintained in time sequence, as a historical log. In order to provide reasonable search and association capabilities, the analyst will provide a description list in Concept Node format, characterizing the report and its salient information, thus forming a report header. For example, a missile site firing report may contain a bulky set of trajectory information, and also identification and summary information, such as a range capability estimate completed from the trajectory data. At the discretion of the Data Base Administrator, the above discussed Process-Hook capability may be employed, treating the original bulky report as a fixed file.

On the other hand, the report identification and qualification information, together with certain computer values, may be stored in time sequence, but with a structure like that of a Concept Node, in order that search and association capabilities be available. Depending upon search and operating conditions,

4.8.2.1 Simple Updating of Entity Net Facts

Suppose that a report is received that a particular radar system is developed at a particular plant. This may support a fact already in the Entity Net, in which the radar system might appear as system #31, and the plant as facility #47. Thus, there is no change anticipated anywhere in the Entity Net, except for a revision upward of the truth probability for a fact already listed in the Entity Net. If the appropriate system authority places a high estimate on the reliability (or likelihood) of the report, then the fact credibility (truth probability) takes a correspondingly large upward revision.

A further illustration would be a tracking radar report of a routine missile firing from a particular site. It is likely that this may affect several facts in the Entity Net, but each as a simple update. It may be that facts are being maintained as follows:

(1) Best current range capability estimate

(2) Best current staging time estimate

(3) Count of weekly firings or misfirings

The two estimates are likely to be improved by the computed values from the incoming report, possibly using statistical tools rather than the truth probability method of the earlier example. The firing count may be set up without any credibility implementation.

The above has been cited as a routine firing report. If, however, there is a problem about whether the supposed site has been incorrectly identified, or the precise identity of the missile type, then the reports are no longer routine. The problems in identification and interpretation may lead to initiating new (or retiring old) facts from the Entity Net.

4.8.2.2 Initiating New Facts in the Entity Net

If we suppose, then, that missile trajectory reports are being received from a tracking radar under conditions where an attempt is being made to identify the missile type (or types), a large number of facts may be initiated and maintained in the Entity Net. For example, summary calculated values may be maintained for each reported trajectory. This contrasts strikingly with the routine situation in which we supposed only the maintenance of current estimated averages or firing counts for the missile site. When the missile type identification is complete, then the facts of the Concept Net may be reorganized in the simpler and more compact manner fitting the routine situation. If we suppose that unexpected information arrives, indicating the need to restudy the identification problem under modified assumptions, recourse may be had to the original reports to appropriately reassess the picture.

Parallel situations appear likely in other information areas. It may be that a large number of carefully organized facts may be maintained on a permanent basis in the Entity Net, related to a scientist of a critical importance. On the other hand, a large number of facts may be maintained on a temporary basis related to another scientist for whom there is an identification issue.

4.8.2.3 Alertness to the Absence of Reports

It seems reasonable to anticipate that absence of reports may frequently be of crucial significance, and yet the initiation of the appropriate alert status or action may not be in the area of report procedures or organization. Two illustrations are considered.

First, we suppose that firing reports for a missile site cease (or drop off sharply) during a time interval. This is likely to become apparent when a Entity Net fact, such as the count of weekly firings, is inspected or reviewed. Interpretation and action may likely result from a search and study of related facts. Perhaps the nearby missile sites, or supply activities may furnish illuminating information.

Second, we suppose that the quantity of firing reports does not alter, but rather the quality. If the reports indicate a steadily increasing site range capability, this is likely to become apparent from observation of Entity Net facts concerning present and old range capability estimates. Neither the absence of reports, nor a trend in the information content, is apt to be apparent at the report level.

It appears, then, that the quality and quantity of incoming reports will often be recognized and coped with as part of the information search, review and interpretation procedures associated mainly with the use of the Entity Net. It is important that report information is adequately distilled into the Entity Net, of course. Even lack of report information should be so distilled. The decay of fact credibility discussed in Section 4.1 is one example. The firing count may serve to indicate lack of reports, as mentioned above.

4.8.2.4 Composite Reports

Especia'ly when the source is non-instrumental, a report can appear that may well be treated as several reports, even if the facts are interwoven in lengthy sentences. Entity Net facts may then be initiated, modified, or updated in much the same way as if the report were a series of simple reports. It may well be that the report will be logged in and recorded as a single composite report in order to maintain the original text for purposes of information security. The Report Node, mentioned in Section 4.6.1 above, is apt to be bulky because of the heavy requirements to provide search and association capabilities where the information is composite.

4.8.2.5 Multiple Versions and Appearances of the Same Fact

It is likely that differing agencies and differing instrumentation will lead to separate estimates for the same technical quantity, whether it is the percentage of a metal in a compound, an estimated range capability, or something very diverse. For easons of information responsibility, clarity, and security, such separate estimates will be maintained almost as if they were separate facts.

It is a some what different matter when differing records are kept of an identical fact for search efficiency reasons. One illustration is the fact that person #32 works at facility #12, routinely stored in "attributes-under-entity" format in the Entity Net part of the Concept Net relating to person #32. The same fact may be stored also as part of an employment list in a "works at" Attribute Node in the Semantic Net (also in the Concept Net). This permits the retrieval of general employment information at one facility without laborious general searching through the Entity Net entries for various persons.

4.8.2.6 Multiple Results From One Report

Sometimes we expect one report to contribute to one Concept Net fact already initiated with other reports, with possible complications in the versions and storage of that fact. It is also possible that distinct Concept Net facts may be involved.

For example, a missile firing may call for firing count, range capability, staging time and other updating of estimates, as already indicated in discussing simple updating. It may be that both specific estimates for the particular missile site, and general estimates for the missile type design are involved and must be separately updated. Such practice may stem from interest in the geographic, supply, and operational effects in various site locations.

4.8.3 System Report Procedures and Report Net Structure

It appears that there is considerable interplay between the procedures and the data structure employed as the report information is entered into the STIS. In particular the Report Net occupies a special fundamental positiion from which restructuring of the Entity Net is occasionally likely, making information security specially important in the Report Net.

If the set of Entity Net facts in some information area is to be abbreviated, it is possible that the restructuring may be accomplished without recourse to the Report Net. A simple illustration would be to change the firing time interval for a missile count record from a monthly to a quarterly span. If the reorganization of Entity Net facts is more far-reaching, or more detailed, then the Report Net is needed.

We note that the Data Base Administrator's task in entering reports is subtle and substantial, even when the operation is on a routine basis.

This is in part because the search and investigation needs of a whole community of analysts are to be met. It would be especially onerous if many analysts felt a frequent need to go to the reports because of dissatisfaction with the distilling process done in Entity Net facts. Occasional temporary alternations in Entity Net information, without recourse to the reports themselves, is apt to be more tolerable, partly because of the difference in data bulk. These considerations take on added weight when the information situation necessitates serious Entity Net reorganization.

SECTION V. PERIPHERAL FILE ACCESS SUBSYSTEM

5.1 BACKGROUND

The current production and test versions of STIS utilize a number of physical files, which taken together comprise the data base, to store and manipulate information entered by the intelligence analyst or application program. The primary file consists of the network structure of nodes representing sets, generic and specific entities, states, etc. The access method to read, store, and traverse this network is a direct one based on a computable relative mass storage address from a node identification number. This ID number is sequential and each node is assigned the next number by STIS upon it[illegible] [illegible]he node"s) creation. Each node is assigned a predetermined contiguous s[illegible] of mass storage space for which the first sector is directly addressab[illegible] [illegible] its ID number. Overflow of this initially allocated segment is handled by linking to subsequent chained segments in an overflow file.

The remaining files are used in conjunction with the node network to respond to inputs from the analyst. When sets are defined in their English language terms, STIS assigns an ID number, [illegible]eferenced above, to them. But the correlation between term and ID number is stored in the Set Term file where the term becomes a key for the ID number. The same procedure holds true for states of entities and for attribute definitions. Additional files contain fact control information, log-on/log-off entries, and an external invoked process log. The access method used to manage these secondary files is the Univac supplied ISFMS package.

ISFMS is a hierarchial index sequential access method using chained overflow data block pointers. It is a single thread (user) package. Hence, it provides no protective physical or logical record lock mechanism in the event two or more concurrent users are attempting to update the same data files. ISFMS, being non-reentrant, necessitates each concurrent user suffering the overhead of having his own core copy of it along with the buffers needed to support each file. Physical blocks may only be selected at the 1/4, 1/2, or full track level (448, 896, or 1792 words respectively). The index blocks for any given file may not be permanently saved and accessed in a separate physical file, although run temporary index files may be created, used, then discarded. File reorganization can take place more often than desired due to the chained overflow mechanism if there is heavy create/delete activity in the file. The chaining mechanism can also require additional I/O accesses and block search time when locating a unique key.

5.2 REQUIREMENTS

Based on the current and projected requirements of the STI system, the peripheral file access subsystem was designed to meet the following criteria:

1) be reentrant thus supporting a multi-user environment and reducing user core requirements

2) provide proper protective access to common files and records

3) perform functions with minimum response time

4) efficiently manage STIS term type data

5) handle a large number of data records (approximately 200,000 to 500,000 terms) with high 'create/delete' activity

5.3 ACCESS SUBSYSTEM DESCRIPTION

5.3.1 Approach and Design Considerations

The access method developed retains the indexed sequential approach of ISFMS. An analysis of term lengths shows that any term may extend to 180 characters with 30 characters the average length. This variation in term size as well as the expected size of the term list (200,000 to 500,000) precludes any effective type of computed address or direct address approach. The design of the subsystem, considered in view of the STIS requirements as enumerated above, follows.

5.3.1.1 Reentrancy and User Core Size Reduction

Under the earlier versions of the Univac EXEC 8 operating system and prior to the introduction of the 1110 hardware, a reentrant program was a self-contained 'I Bank only' program. Each user program wishing to execute the reentrant program would 'link to' it by executing an Executive Request from his own program. The reentrant program consists solely of instruction code, and data and sufficient working storage for its own housekeeping. User dependent working stroage must be allocated in the user program and would include any I/O buffers needed for mass storage files. Thus, creating a reentrant version of the access subsystem (much on the same design lines as FMS-8) would not be sufficient to save STIS the memory overhead normally allocated for I/O buffers. This could amount to 5500 words per user based on six peripheral files, double buffering, and minimum 1/4 track blocks.

The upgrading at FTD from the Univac 1108 to 1110 provided a new joint hardware/software technique that would allow concurrent users to execute common instructions and share common data areas. The access subsystem was designed to be written as a set of reentrant modules and collected together as a common instruction bank sharable by as many concurrent users who 'link to' it via a hardware 'load instruction bank and jump' (LIJ) instruction. A single set of I/O buffers will reside in a common data bank and these will be used to service all the concurrent users. Each user's core requirements then have been reduced to providing only a small area for control tables and working storage. The EXEC 8 operating system will always maintain the latest copies of the common I and D banks both in core and on the system mass storage swap area. This assures that any changes made to tables, buffers, etc. by the access subsystem will be preserved.

5.3.1.2 Common File and Record Access

Once a user has linked to the access subsystem, his command and argument list will be checked for validity. Access to a file after validity checks are passed will be controlled by a hardware 'test and set' instruction on a specially marked cell. This will effectively lock out all other concurrent users from using that file. If the file is currently being accessed by another user, then the EXEC 8 operating system will place the new user on a queue maintained by it. When the previous user's file request has been completed, the file is unlocked by the access subsystem and the operating system will activate the next user on the queue for this file (if any). Each file opened and being manipulated by the access subsystem will have its own specially marked lock word and queue maintained by EXEC 8. Each file has its own dedicated buffer area within a common data area pool. Thus, the access package may perform a single request concurrently on each file opened within it with multiple requests on any given file(s) automatically queued by EXEC 8.

Certain users may be declared by STIS to be inelligible for writing information into the data base. This inelligibility might be determined from the user's password. STIS then might specify to the access subsystem that certain files may be addressed in the 'read only' mode for those users. The access subsystem has been designed to protect against write operations on 'read only' files.

In addition to its own internal common file access mechanism, the subsystem allows individual users to place a logical lock on any selected record in a file. This logical lock inhibits all requests (except for 'inquiry only' requests) for this record by other users until the lock is cleared by the orginating user. An analysis of typical operations on the peripheral files shows that only one record lock per user per file should occur. Based on this, the access subsystem will permit each user to set one lock on each file he has open. A provision to expand the number of locks per file is available by redefining a parameter and reassembly.

5.3.1.3 Minimize Response Time

The access subsystem permits the index blocks for any file to be optionally maintained in a separate catalogued index file (as opposed to being embedded within the same file containing the actual data blocks). Separate index block files permit overlap of some I/O activity through channel separation on differing mass storage devices or on dual access devices (e.g., FTD's 8440 disc subsystem).

Separate buffers are allocated and dedicated to the highest level index block for each file opened within the subsystem. An additional buffer is utilized for lower level index blocks. All unnecessary 'reads' are avoided if the appropriate index and data blocks for a given file are already buffer resident.

5.3.1.4 Manage STIS Term Data

The access subsystem will support a file which has records in which the key and data portion may be variable or fixed in length. The file organization is specified by the STIS programmer. Once specified, all records within that file must conform to the type given. Generally, there will be a one-to-one correspondence between key and data entry of a record. However, an analysis of the present STIS term files (set, plane, and attribute term files) showed that one English term could represent more than a single role (e.g., be both a set and attribute). Thus, there are duplicate term entries in the current files. Combining these files into a single term file would not only reduce duplicate storage but also reduce search time for language processing functions requiring all uses of a given term. The access subsystem was designed to allow for a one-to-many correspondence between key and data entries of a record. Each data entry is tagged by a role code indicator (which may be interpreted as a secondary key suffixed to the primary key). Thus, a file of terms might be created where a key is the English term and the data entries are the specific nodes in the network file which represent the differing uses of the term.

5.3.1.5 Large Record Volume with High Create/Delete Activity

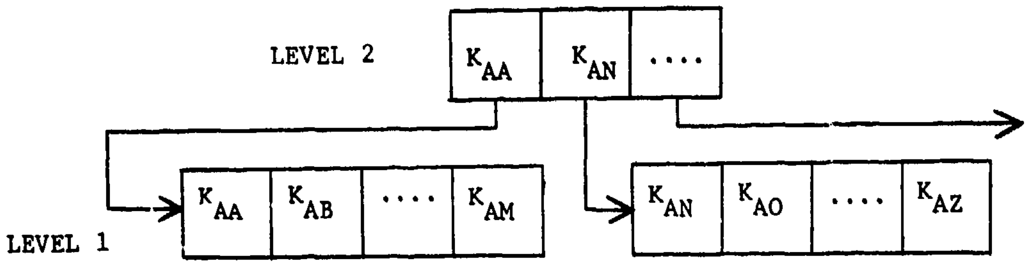
File reorganization was one of the prime considerations in the design of the access subsystem. It was felt that as much maintenance should be performed as was possible during the execution of requests which physically alter the data content of a given file. This was especially true of a dynamically expanding and contracting file such as a common STIS term file. The maintenance function must be considered when performing a DELETE, MODIFY, or INSERT operation. A DELETE request causes the key and data entries to be removed from the data block in which they reside. The data block is compacted; the block control words updated; and the data block is rewritten to mass storage. The deleted space is then immediately available for reuse. An INSERT request causes a key and data entry to be added to a data block. If the physical data block will overflow due to the size of the new entry, a data block split is performed. An attempt is made to split the old block equally with a new data block and the insertion made in the appropriate place. However, in some cases, a second additional block may be necessary to properly split and sequentially maintain the data records. After the insertion is made at the data block level, the index blocks need to be updated since an additional 1 or 2 data blocks were created. The same procedure outlined above is used to update the index blocks. As a lower level index block overflows, its next higher level block is updated until the current highest level is reached. If the current highest level block overflows, a new index level is generated. The access subsystem is designed to handle an unlimited number of index levels. (Figure 5-2 depicts in a simplified style the results of an INSERT operation.) A MODIFY request on a variable length data entry file may cause contraction or expansion of a data block depending on the new data entry size. If contraction is indicated, the DELETE procedure is basically used; if expansion, the INSERT procedure is basically used.

5.3.1.6 Key and Data Considerations

Since the access package does not employ a chained overflow block technique, no record then may extend across block boundaries (i.e, no record consisting of key entry and data entry or multiple data entries when qualified by role code may exceed the size of a physical data block).

In a hierarchial index sequential file, there must exist some single high level index pointer block. The length of the key entries within this block must be less than or equal to one half the effective block size. If this were not so, there would exist the possibility that only 1 key entry could reside in an index block. Each index block must be able to hold at least two keys (known as 'range' keys) in order to generate a hierarchial structure. An index 'range' key is used to point to a lower level index or data block in which the 'range' key itself is the lowest order sequential key.

In Figure 5-1, Range Key Indices



the 'range' key K_{AA} in level 2 points to a block at level 1 in which K_{AA} is the lowest ordered key. The same holds for 'range' key K_{AN} on level 2. Thus, all the records that have a key between 'range' key K_{AA} and up to but not incluing 'range' key K_{AN} are contained in the level 1 block pointed to by K_{AA}.

The very first record that must be loaded into a file should have a key consisting of binary zeroes. This key will function as an initialization sentinel. It will appear as the first range key in each successively higher index level block. Its appearance guarantees that no sequentially lower (in the alpha numeric collating sequence) key can be constructed and possibly cause problems on retrieval operations.

The access subsystem has been designed such that an index block overflow caused by an insertion of a new 'range' key must be restructured with the addition of only one new index block. The old and new 'range' keys will be divided as equally as possible within the old and new index blocks. This design constrains the maximum key entry size to 30% of the physical index block size. For the smallest size index block assignable by the programmer of 112 words (Univac 8440 standard disc prep size), the largest size key cannot exceed approximately 33 words or 198 characters. This appeared very unlikely to occur within STIS. Figure 5-2 shows the reformatted index and data blocks after an overflow caused by an INSERT operation.

5.3.1.7 File Considerations

The tables that have been identified for definition and manipulation by the access subsystem are the 'term encoding table', 'word encoding table' (both used by the Directory Services subsystem), 'fact control table', 'log-on/off table', and 'invoked processor table'. An analysis of the information content and current and future usage statistics of these tables was performed. The results show that the 'term encoding table' (or TET) and 'word encoding table' (or WET) will be the most heavily accessed tables and the 'fact control table' (or FCT) also frequently accessed. Thus, these three tables should have separate index block files. The 'log-on/off table' and 'invoked processor table' are very low access tables and need not have separate index block files.

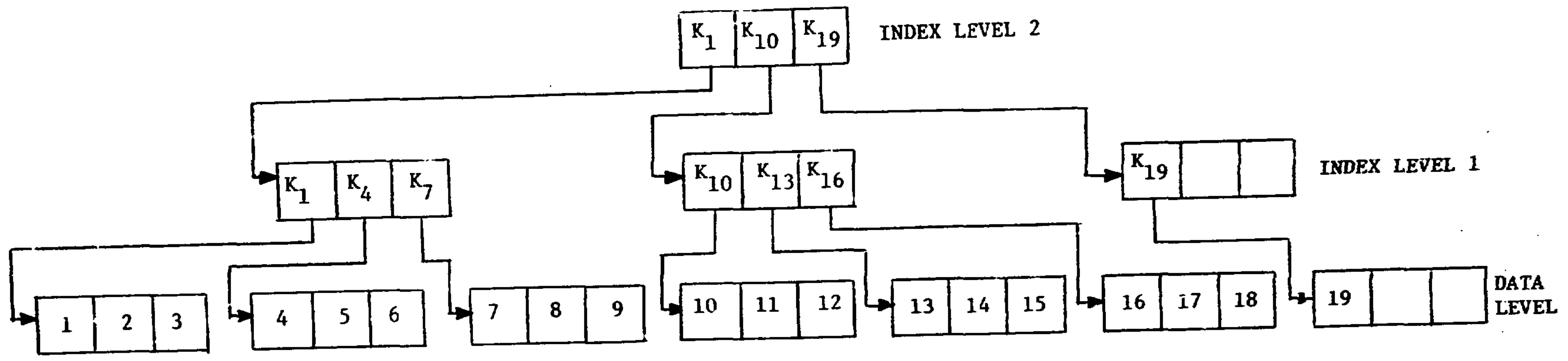


Figure 5-2(a) Index and Data Block Structure Before Insertion

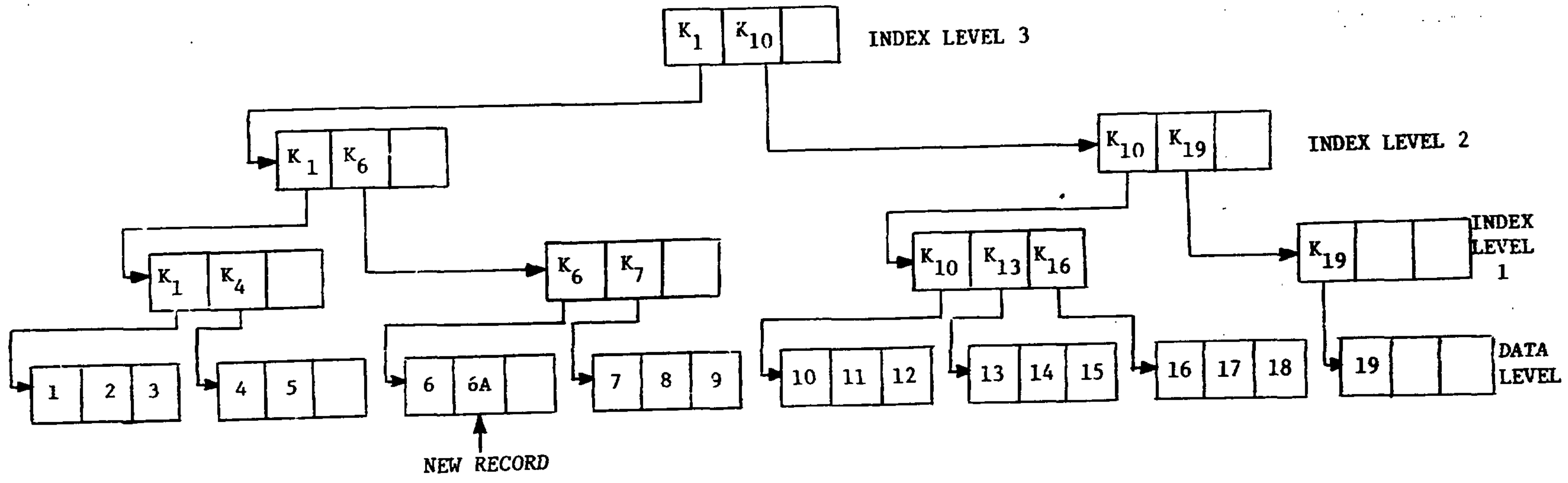


Figure 5-2(b) Index and Data Block Structure After Insertion

We have assigned each table a set of table definition parameters based on its record structure and computed sample loading conditions. The access subsystem attempts to initially load data blocks to 50% of capacity and to completely fill the associated index blocks. The initial load factor percentage may be changed by reassembly.

The following formulas are used in computing the loading conditions.

# of Block Control Words (NBCW)	=	3
Load Factor (LF)	=	50%
Data Record Size (DRS)	=	Key Length + Record Control Word + Data Length
Key Size (KS)	=	Key Length + 1
Max Keys (MK)	=	(Index Block Size - NBCW)/KS
Initial Data Records (IDR)	=	((Data Block Size - NBCW)/DRS) *LF
Total Initial Data Records	=	IDR * $(MK)^{\text{Number of Index Levels}}$

(Fractional sizes are rounded up to next whole word)

1) Term Encoding Table (TET)

Organization is contiguous fixed key/fixed data.

Record Control Word = 0

Key Length = 2 words

Data Length = 1 word

Index Block Size = 336 words

Data Block Size = 224 words

Number of Index Levels = 2

DRS = 2 + 0 + 1 = 3
KS = 2 + 1 = 3
MK = (336-3)/3 = 111
IDR = ((224-3)/3) * .50 = 36

Total Initial Data Records = 36 * $(111)^2$ = 443,556

2) Word Encoding Table (WET)

Organization is contiguous fixed key/fixed data

Record Control Word = 0

Key Length = 2 words

Data Length = 1 word

Index Block Size = 224 words

Data Block Size = 336 words

Number of Index Levels = 2

DRS = 2 + 0 + 1 = 3
KS = 2 + 1 = 3
MS = (224-3)/3 = 73
IDR = ((336-3)/3) * .50 = 55

Total Initial Data Records = 55 * $(73)^2$ = 292,545

3) Fact Control Table (FCT)

Organization is contiguous fixed key/fixed data

Record Control Word = 0

Key Length = 5 words

Data Length = 2 words

Index Block Size = 336 words

Data Block Size = 336 words

DRS = 5 + 0 + 2 = 7
KS = 5 + 1 = 6
MK = (336-3)/6 = 55
IDR = ((336-3)/7) * .50 = 23

Number of Index Levels = 2
Total of Initial Data Records = 23 * $(55)^2$ = 69,575

Number of Index Levels = 3
Total Initial Data Records = 23 * $(55)^3$ = 3,826,625

4) Log-On/Off Table

Organization is contiguous fixed key/fixed data

Record Control Word = 0

Key Length = 5 words

Data Length = 10 words

Index Block Size = 112 words

Data Block Size = 112 words

DRS = 5 + 0 + 10 = 15
KS = 5 + 1 = 6
MK = (112-3)/6 = 18
IDR = ((112-3)/15) * .50 = 4

Number of Index Levels = 1
Total Initial Data Records = 4 * 18 = 72

Number of Index Levels = 2
Total Initial Data Records = $4 * (18)^2$ = 1,296

5) Invoked Processor Table

Organization is contiguous fixed key/fixed data

Record Control Word = 0

Key Length = 5 words

Data Length = 5 words

Index Block Size = 112 words

Data Block Size = 112 words

DRS = 5 + 0 + 5 = 10
KS = 5 + 1 = 6
MK = (112-3)/6 = 18
IDR = ((112-3)/10) * .50 = 5

Number of Index Levels = 1
Total Initial Data Records = 5 * 18 = 90

Number of Index Levels = 2
Total Initial Data Records = $5 * (18)^2$ = 1620

5.3.2 Functional Outline

The following basic user (i.e., STIS systems programmer or potentially other applications programmers) functions are supported by the access subsystem. The detailed description and parameter list expansion of each is given in Appendix B.

OPENAP	Initializes the access subsystem for the user
DEFNFILE parm list	Defines a new file
OPEN parm list	Opens a file and its associated index file
READ RANDOM parm list	Reads a data entry associated with the given key
READ SEQUENTIAL parm list	Reads a data entry associated with next sequential key
READ RANDOM w/lock READ SEQUENTIAL w/lock	As above but also inhibits further access to the data record by other users
INSERT parm list	Adds the data entry associated with the given key
MODIFY parm list	Updates the data entry associated with the given key
DELETE parm list	Deletes the data entry for the given key
INFORM parm list	Gives the user statistics about file usage
CLOSE parm list	Performs an orderly closing action on a file
CLOSEAP	Closes the access subsystem to the user

5.3.3 Error Processing

The access subsystem performs a number of validity checks on the commands issued to it by STIS service modules. If an error is detected, an error code is returned to the calling module and the command is effectively not performed. It is the calling module's responsibility to take the appropriate action (e.g., inform the user of the error, reformat the command, etc.) after an error code is returned. A complete list of error codes and their interpretations is given in Appendix B.

SECTION VI. DIRECTORY SERVICES SUBSYSTEM

6.1 INTRODUCTION

The STI Directory Subsystem consists of the tables and program modules shown in Figure 6-1. The Directory is designed to uniquely encode and decode all system words and terms (of one or more words) used in STIS as attributes, values, commands, and noise. These system words, and terms are maintained in integrated ordered lists (the Word and Term Encoding Tables) so that an input language scanner, using the directory services, can recognize and determine the role played by every word apt to be entered (including the attribute(s) associated with a value term), the uniqueness of a term, and the equivalence of a term to other terms. The Directory functions and tables are defined in Appendix C along with accompanying system level flow charts.

An analysis of the terms used by analysts to define entity names, attributes, and sets in STIS showed that terms often were not unique in their usage and were often multi-word. Words often appeared as components of several terms. The number of characters in a word could be as large as twenty or as

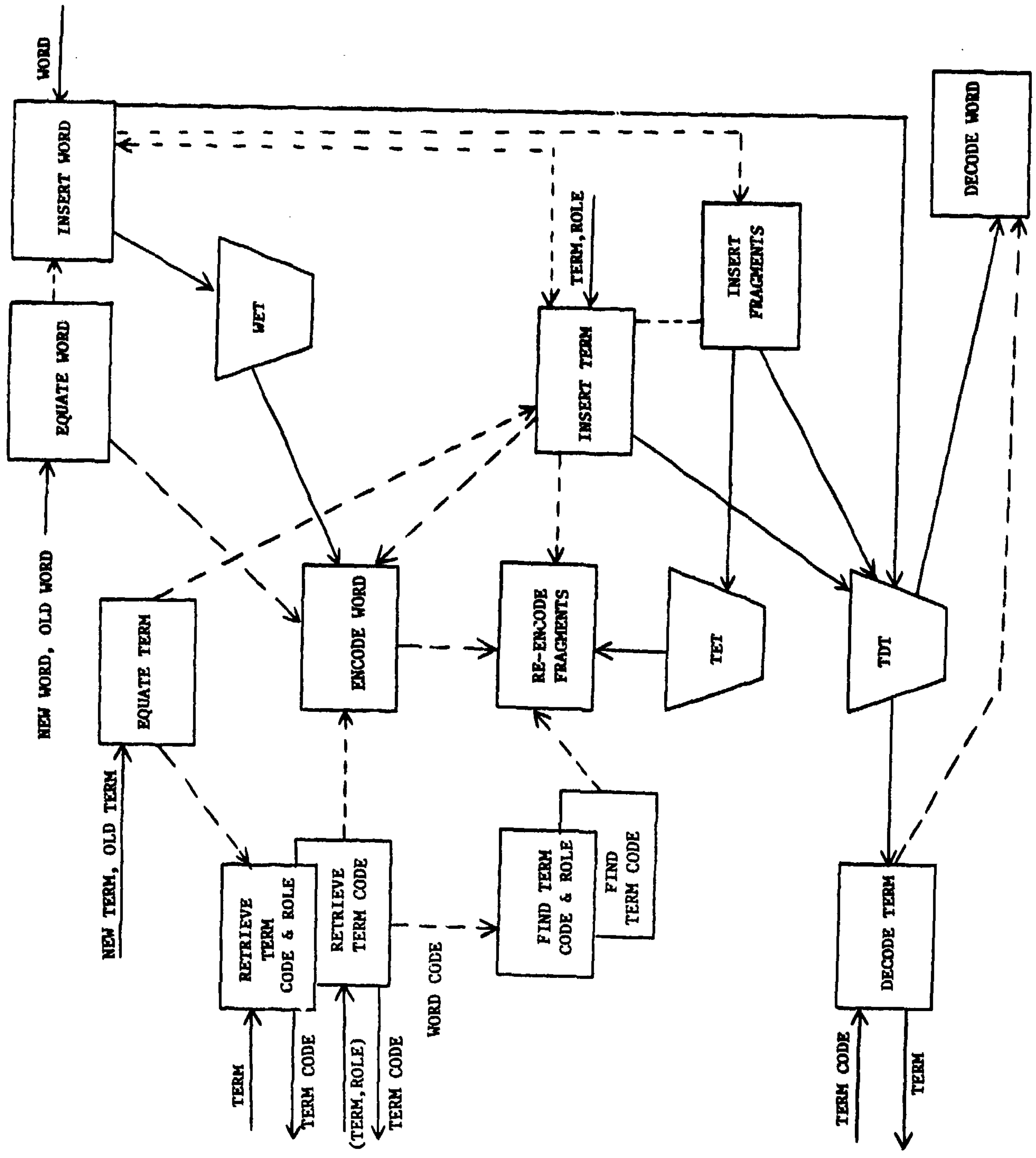


Figure 6-1 Directory Subsystem

small as one while the total number of characters in a multi-word term could approximate 180. Based on these facts, we attempted to define encoding/decoding algorithms and data tables to handle terms as a variable number of variable length words and their associated roles. We also investigated the possible partitioning of words into meaningful linguistic units or morphemes and the treating of definable groups of words as term fragments.

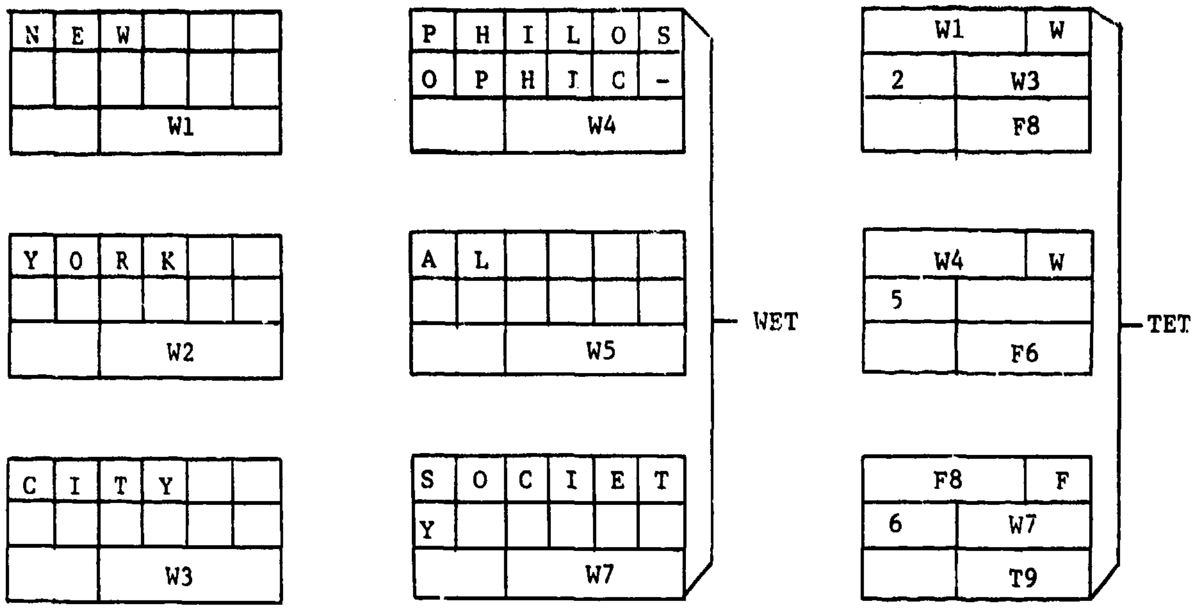
6.2 ENCODING MECHANISM

All words and terms are stored with numeric codes in ordered lists (via the Word and Term Encoding Tables) along with an associated role map which defines how the word or term is being used. Words may be of arbitrary character length and terms may consist of any number of words. In the event a word is too large to be stored in a single WET (Word Encoding Table) entry, it is partitioned, starting at the leftmost character, into segments each of which will comprise a single WET entry (a segment being the longest string storable in a WET entry). The last segment will consist of the remaining character string from the last break to an end of word delimiter. Continued segments are marked by inserting a special character (e.g., a hyphen) as the last character of the WET entry. The numeric codes assigned to each word segment are then recursively encoded from TET (Term Encoding Table) entries until a single numeric code representation is obtained. Similarly if a term is too large to store in a single TET entry, it is recursively encoded via a single left to right scan until a single numeric term code is obtained. Figure 6-2 represents the typical term encoding process.

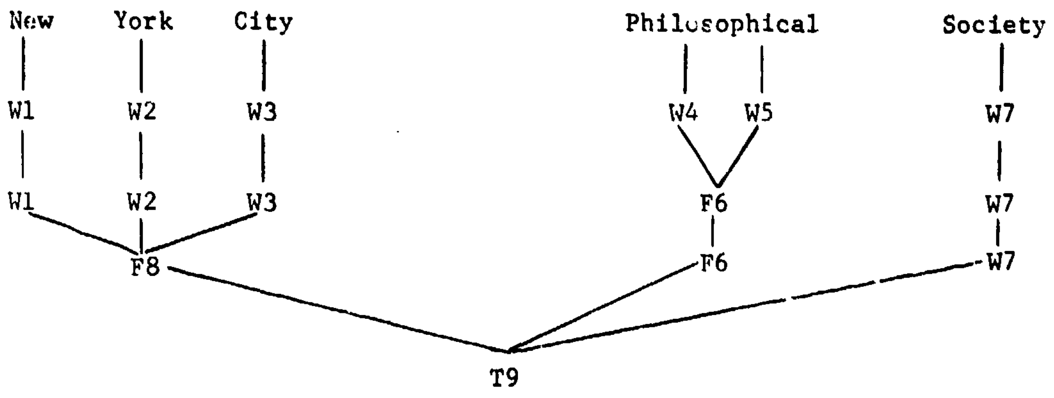
6.3 DECODING MECHANISM

An integrated decoding table is maintained by the Directory Services Subsystem. As each word and term (and word and term fragments) are entered into the Word and Term Encoding Tables, an additional entry is made in the integrated Term Decoding Table (TDT). When term or word decoding is required (e.g., in response to an analyst query of the value of some attribute), basically an inverse encoding algorithm is followed. The given term code is entered in

Encode Term (New York City Philosophical Society) = T? Assume WET and TET entries are as shown below



The encoding request proceeds as follows:



The term code returned by 'Encode Term' function is T9.

Figure 6-2 Typical Term Encoding Process

the initial entry in a push down table. A recursive loop is begun in which the first entry in the push down table is examined. If the entry has the Word Code Indicator bit set, then the entry is further decoded into its English text. The text is the concatenated with any previously decoded text. A 'blank' character is added to the text if the word is not marked as continued. This entry is then popped off the stack since it is completely processed. If the entry did not have the WCI bit set, then the decoding entries are retrieved and entered in the push down table in place of the first entry. Another iteration in the loop is then begun. When the push down table is empty, the generated text is returned to the caller.

REFERENCES

1. STIS Users Guide. Foreign Technology Division, May 1975.

2. J. Sable and S. Forst: Design Concept for an Augmented Relational Intelligence System (ARIAS). AUER-2022-TR-2, August 1973 (RADC-TR-73-342/AD 773 189)

3. I. Goldhirsh and R. Carson: A Deductive System for Intelligence Analysis. AUER-2255-TR-1, March 1976.

4. J. Sable and S. Forst: Needs Analysis For Inference Systems at FTD. RADC-TR-73-4, January 1973 (AD 757 218)

5. G. Hendrix: Expanding the utility of semantic networks through partitioning. Artificial Intelligence Group Technical Note 105, SRI (June 1975).

APPENDIX A. CURRENT STIS STRUCTURE

CURRENT STIS STRUCTURE

The capabilities, operating characteristics and performance of STIS follow largely from the general architecture of its system components and its data structures. The general architecture of STIS and its relationship to the on-line and batch user is shown in Figure A-1. STIS is a modular system in which user requests for information services may pass through several levels of translation during processing. It provides a "general" (non-application specific) language called IPL (Interactive Processing Language) for the user who wants to "browse" through the data base and perform his own analyses. Other users, whose analysis requirements have been formalized can be provided with "special purpose" languages through application programs written in COBOL (e.g., IPS) and FORTRAN (e.g., IEAS). The HOL (High Order Language) programmer user writes a special Program Interface Module (PIM) which is compiled and linked (collected) with the application program. HOL requests for STIS services are made through the PIM, which is typically a 5K word module of the users code. In the current operation, each user (on-line or batch) must have his own "copy" of STIS, some 40K words, in memory. Large user programs, such as IEAS may run from 85K to 90K words (perhaps reducible to 50K with segmentation), putting a severe limit on the number of active users which can be accommodated in the four 65K word memory banks available to users in the 1110. A fundamental goal of a near-term optimazation effort would be to handle multiple users with a single copy of STIS.

The architecture of the STIS data structure can be described as an amalgamation of different classical data structures. It can be called a Relational Data Network in that it is composed of nodes which are interrelated through binary relationships (and their converse) using a unique Node Identifier (NID) as a logical link. The nodes themselves have a variable hierarchic format in which common elements are "factored" to avoid redundancy within the node. In addition to control data embedded at various levels in the node structure, the basic user information is carried in attribute name (A#)/attribute value format. Multiple values are permitted for each attribute, along with warnings, comments, and other text. The current STIS 1 node is seen in its physical and logical form in Figures A-2 and A-3 respectively.

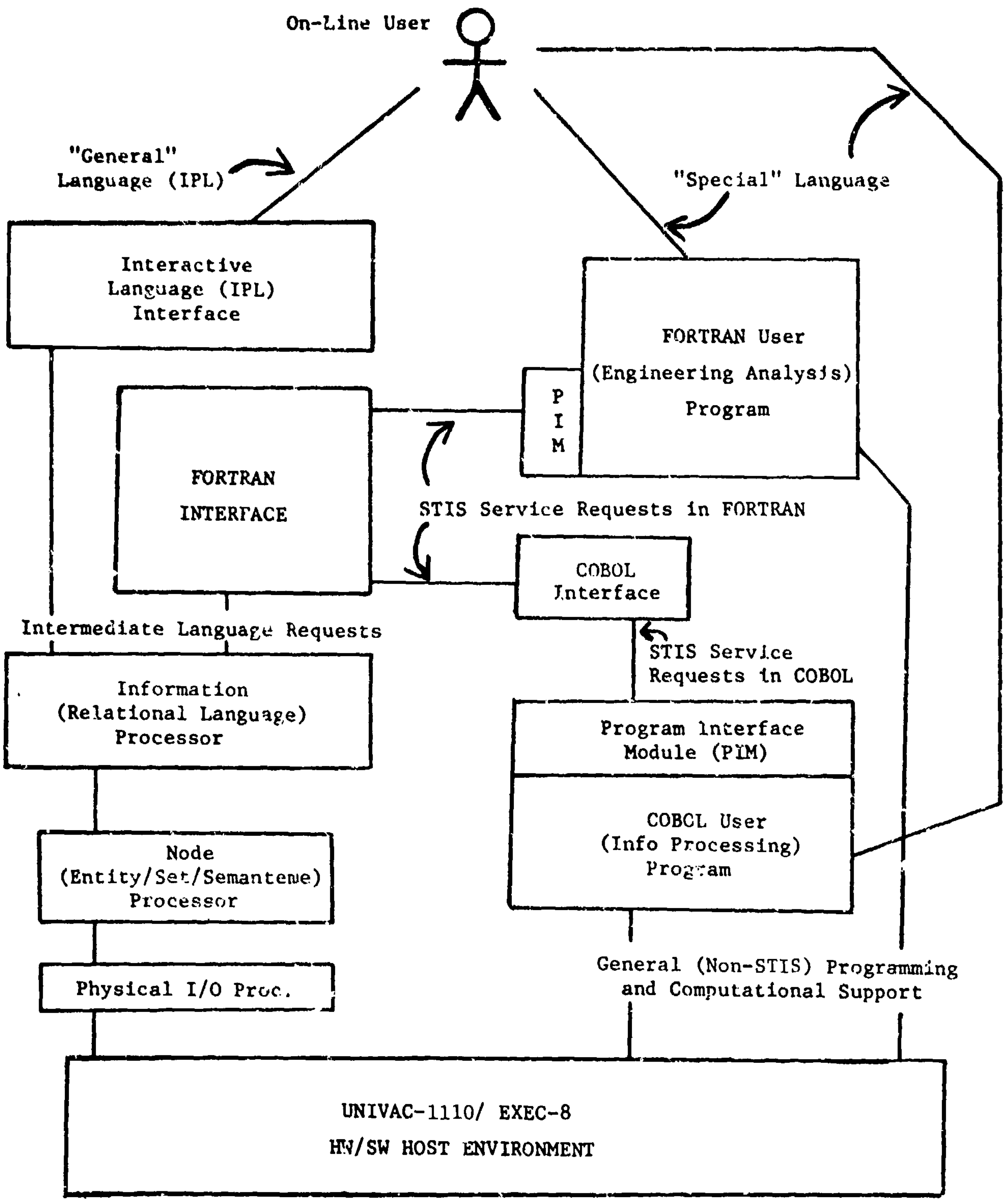
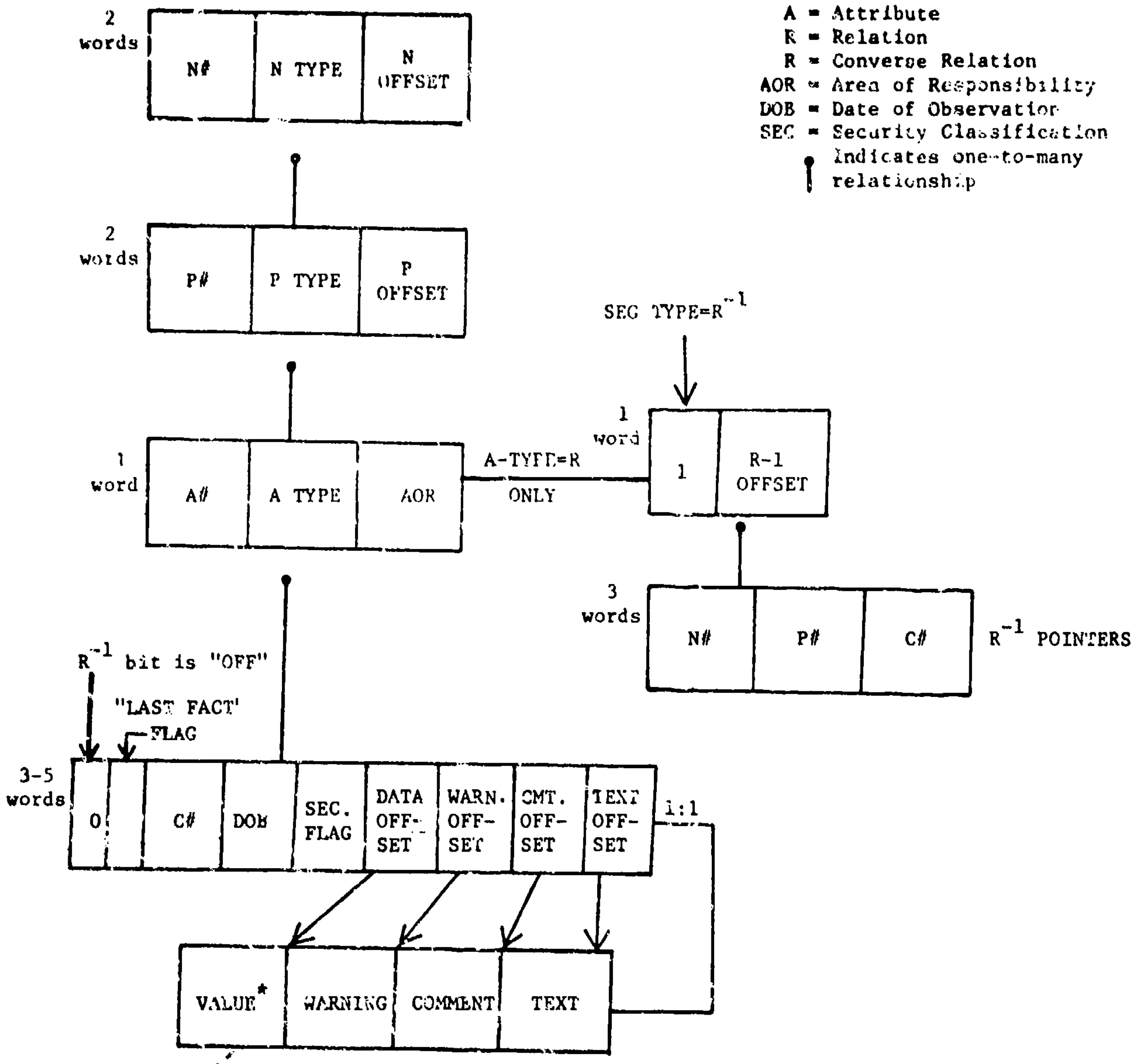

On-Line User
"General" Language (IPL)
"Special" Language
Interactive Language (IPL) Interface
FORTRAN User (Engineering Analysis) Program
P I M
FORTRAN INTERFACE
STIS Service Requests in FORTRAN
COBOL Interface
Intermediate Language Requests
STIS Service Requests in COBOL
Information (Relational Language) Processor
Program Interface Module (PIM)
COBOL User (Info Processing) Program
Node (Entity/Set/Semanteme) Processor
Physical I/O Proc.
General (Non-STIS) Programming and Computational Support
UNIVAC-1110/ EXEC-8
HW/SW HOST ENVIRONMENT

Figure A-1 Current STIS Architecture

LEGEND

N	= Node
P	= Plane
C	= Data Control
A	= Attribute
R	= Relation
R	= Converse Relation
AOR	= Area of Responsibility
DOB	= Date of Observation
SEC	= Security Classification
❘	Indicates one-to-many relationship



* Value is arbitrary size array char string, or N#/P# pointers depending on Attribute.

Figure A-2 Physical Node Structure

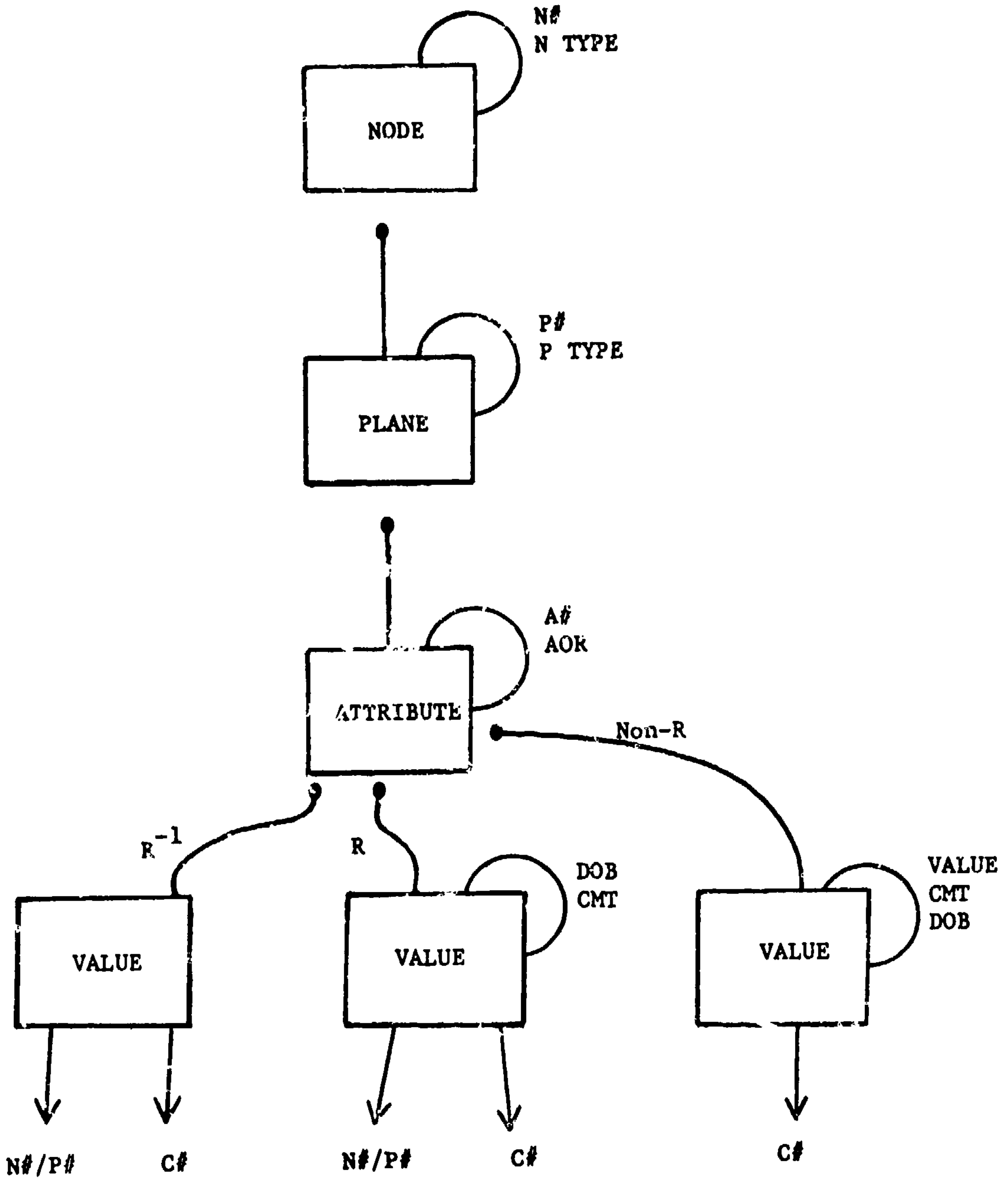
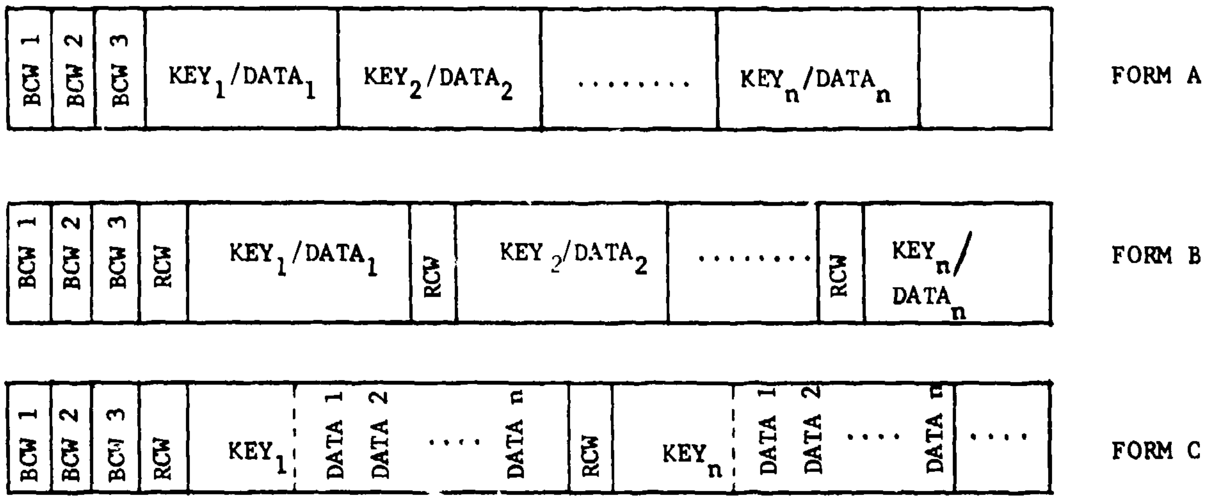


Figure A-3 Logical Node Structure

APPENDIX B. PERIPHERAL FILE ACCESS SUBSYSTEM FUNCTIONAL DESCRIPTION

GENERAL DATA BLOCK STRUCTURE FORMS

The 3 general forms employed to store key/data entries are outlined below:



The Block Control Words (BCW's) are defined as follows for all forms:

BCW 1

T1	S3	S4	T3
BNEXT	BT	SPARE	BENTRY

BNEXT = next available word in this block

BT = block type = 1 indicating this is a data block

BENTRY = number of key/data entries in this block

BCW 2

H1	H2
SPARE	BFRWRD

BFRWRD = block number of next sequentially forward data block.
(In last data block, BFRWRD = 0).

BCW 3

H1	H2
SPARE	BN

BN = this block's number

The Record Control Words (RCW's) are defined as follows:

FORM A No RCW appears within the data block for fixed key and fixed data records. An implicit RCW is defined in various fields of the File Control Table for this file. Note that for this structure, only one data entry may exist for a given key entry (a 1 to 1 correspondence exists).

FORM B

	T1	T2	T3
RCW	KLEN	SPARE	NKEY

KLEN = length of following key in characters. Can be used as an offset to data entry by converting to words, rounding up, and adding 1.

NKEY = 'offset to next key in block' in words. This is a total count in words of the key and data entries and increased by 1 to compensate for the RCW.

Note that the size of a variable length data entry is computed as the difference in words between 'offset to next key 'and' offset to data entry'. This implies that on a retrieve operation for a variable length data entry, the record returned to the user could be up to 5 characters longer than the original insertion size in characters since we always return an integral number of words.

As in FORM A above, this structure only permits a 1 to 1 correspondence between key and data entries.

FORM C

	T1	S3	S4	T3
RCW	KLEN	SPARE	DENTRY	NKEY

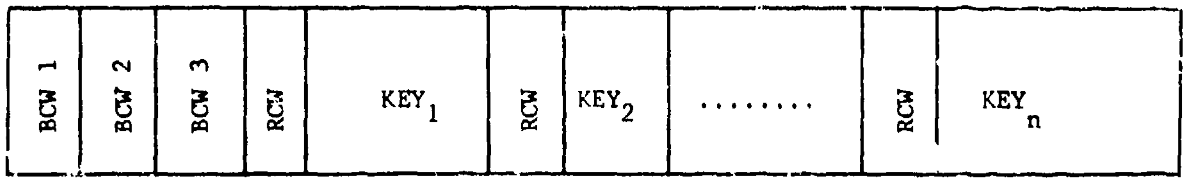
KLEN = length of following key in characters (as in FORM B)

NKEY = 'offset to next key in block' in words (as in FORM B)

DENTRY = number of data entries that exist for the following key.

This structure permits a 1 to many correspondence between key and data entries. Note that use of this structure carries the restriction that the data entries must be of the <u>same</u> <u>fixed</u> <u>length</u>.

INDEX BLOCK STRUCTURE



BCW 1

T1	S3	S4	T3
BNEXT	BT	SPARE	BENTRY

BNEXT = next available word in this block

BT = block type = 2, indicating this is an index block

BENTRY = number of 'kev' entries in this block

BCW 2

H1	H2
SPARE	BFRWRD

BFRWRD = the number of the next physical block at the same depth level as this block. (A forward pointer). (In last block of any level, BFRWRD = 0)

BCW 3

H1	H2
SPARE	BN

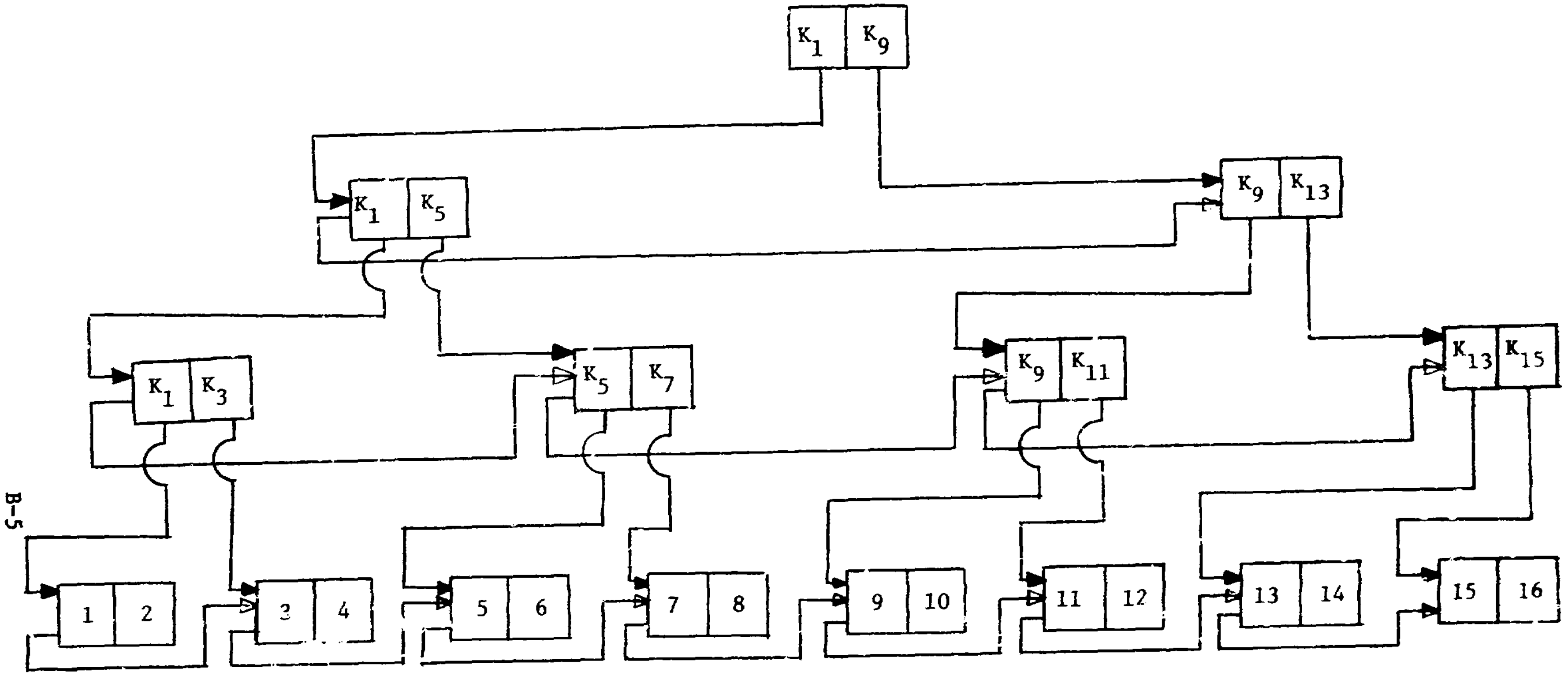
BN = this block's number

RCW

T1	S3	H2
KLEN	SPARE	BDOWN

KLEN = length of following key in characters. Can be used as an offset to next key by converting to words, rounding up, and adding 1.

BDOWN = block number of next lower level block in which this key range is further expanded. (At lowest index level, this field points to the actual data blocks).



A multi-level index sequential structure depicting a 3 level index with all the 'down' and 'forward' block pointers.

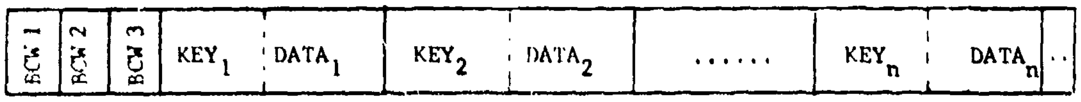
A 'down' pointer is shown as ──▶

A 'forward' pointer is shown as ──▷

SPECIFIC KEY/DATA ORGANIZATION TYPES

There are 7 methods for organizing and manipulating key/data entries within the access package. A description of each along with the general data block structure employed follows:

TYPE 1

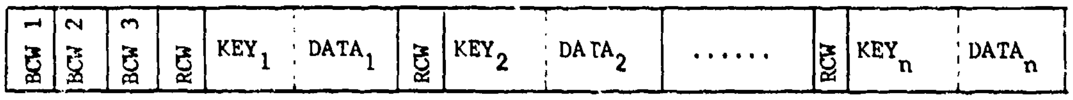


Uses FORM A

This type supports a fixed key and fixed data entry each of arbitrary character length. Within the physical data block, both the key and data entries will each be positioned to begin on a word boundary. Entries which are not multiples of 6 characters (1 word) will be padded with binary zeroes.

When inserting records into this type file, the user will specify a 'key entry area' and a separate 'data entry area'. The key and data entries can be thought of as being logically but not physically connected:

TYPE 2

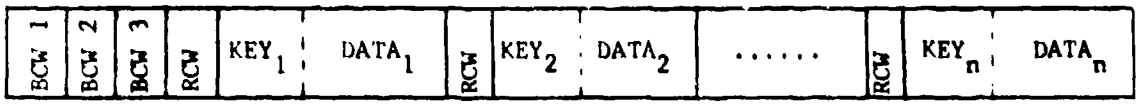


Uses FORM B

This type supports a variable length key entry with a fixed length data entry each of arbitrary character length. Within the physical data block, both key and data entries will each be positioned to begin on a word boundary. Entries which are not multiples of 6 characters will be padded with binary zeroes.

The user must specify the maximum character length the key entry may assume at file definition time. Each manipulative command also requires the character length of the particular key entry being accessed as a formal argument.

TYPE 3

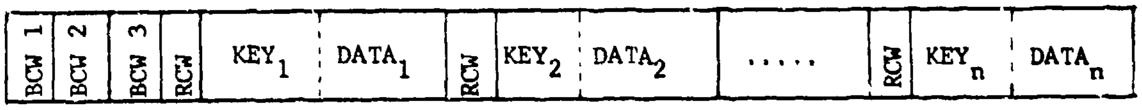


Uses FORM B

This type supports a fixed length key entry with a variable length data entry each of arbitrary character length. Within the physical data block, both key and data entries will each be positioned to begin on a word boundary. Entries which are not multiples of 6 characters will be padded with binary zeroes.

The user must specify the maximum character length the data entry may assume at file definition time. Each manipulative command either requires the character length as a formal input argument or returns the character length rounded to the next multiple of 6 into a formal return argument.

TYPE 4

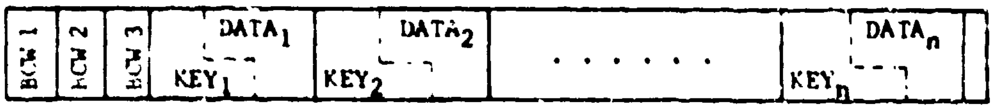


Uses FORM B

This type supports a variable length key and variable length data entry each of arbitrary character length. Within the physical data block, both key and data entries will each be positioned to begin on a word boundary. Entries which are not multiples of 6 characters will be padded with binary zeroes.

The user must specify the maximum character length for both key and data entries at file definition time. Each manipulative command will require the actual key length in characters as a formal input argument and either actual data length in characters as an input argument or will return the character length rounded to the next multiple of 6 into a return argument.

TYPE 5



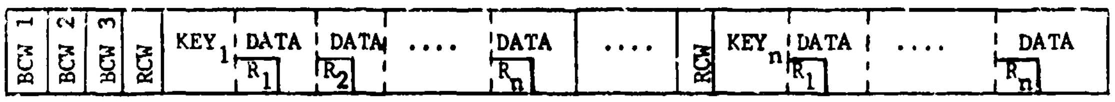
Uses FORM A

This type also supports a fixed key and fixed data entry each of arbitrary length as in TYPE 1. However, the data entry will be positioned on a character boundary inside the physical block. Only data entries which are not multiples of 6 characters will be padded with binary zeroes.

When inserting records into this type file, the user will specify a single 'key/data entry area'. In this area, the user will have packed the key and data entries as a contiguous string of characters. The access package will take and position this contiguous string in the physical data block.

For retrieval commands, the access package will locate and unpack the string in the physical block and return just the data entry portion into a user defined data return area as is normally done for other type files.

TYPE 6



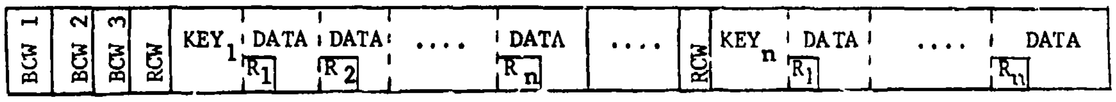
Uses FORM C

This type supports a fixed key entry with a multiple number of fixed data entries. Within the physical data block, the key and each data entry are positioned to begin on a word boundary. Entries which are not multiples of 6 characters will be padded with binary zeroes.

Each data entry is uniquely identified by a role code number assigned by the user (designated in figure by R_i). There may only exist 1 data entry/role code/key entry. The role code is a 4 bit field (left justified) embedded in the first character of the data entry. For this reason, the user must not use those bits as user storable data bits. The access package will merge the user defined role code number into these bits when storing data entries and strip off these bits when returning a selected data entry to the user. The data entries are stored in the data block in the order in which they are received. Sequential ordering according to increasing role code number is not maintained.

Note that all role code data entries must be of the same fixed size specified at file definition time.

TYPE 7



Uses FORM C

This type supports a variable length key entry with a multiple number of fixed data entries. The same comments apply here as in TYPE 6 above.

ACCESS PACKAGE PRIMITIVE COMMANDS

The following commands are used independently of file organization:

1) DEFNFILE (FCT, Data blk size, File Type, Key length, Data length, Role Coass Flag < , Index Filename, Index blk size >)

This command defines the organization and parameters for the file named in the FCT.

FCT	The File Control Table defined and residing in the user's memory area. It contains the name of the file to be defined in its first 2 words.
Data blk size	The size in words to be allocated for the physical data blocks. This must be a multiple of the disc prep size (typically 112 words).
File Type	Specifies the organization of the file according to the type of key and data entries to be processed.

= 1	Fixed key, fixed data
= 2	Variable key, fixed data
= 3	Fixed key, variable data
= 4	Variable key, variable data
= 5	Fixed key, fixed data packed

Key Length	Specifies either the actual key length in characters for fixed key files, or, the maximum key length for variable key files. No key length can exceed 30% of the total length of the physical index block.
Data Length	Specifies either the actual data entry length in characters for fixed data files, or, the maximum data entry length for variable data files. No key/data entry pair may have a total length which exceeds the physical data block size.

Role Codes Flag	Specifies whether the file will or will not have role codes within it.

= 0	no role codes in file
≠ 0	role codes in file

If role codes are indicated then 'File Type' must be = 1 or = 2. The data entries must be fixed in length with each data entry equal in size to all other data entries.

<, Index filename, Index block size

These are optional parameters which specify that the index blocks are to be stored in a separate physical file whose 12 character name is given in 'Index filename'.

'Index block size' specifies the size in words to be allocated for the physical index blocks. This must be a multiple of the disc prep size.

If these two parameters are absent, the index blocks will be embedded in the same file as the data blocks. The index block size will be made equivalent to the data block size.

2) OPEN (FCT, READ/WRITE Flag)

This command permits a user access to a particular file.

FCT	The File Control Table defined and residing in the user's memory area. It contains the name of the file to be opened and accessed in its first 2 words.
READ/ WRITE Flag	Specifies read/write mode for this file.

= 0	reading and writing permitted
= 1	only reading permitted

3) CLØSE (FCT, Inform area)

This command closes a particular file and inhibits further access to that file by the same user without a new 'OPEN'.

FCT	The File Control Table
Inform area	A seven word array residing in the user's memory area into which statistical information will be placed. The information will be relative to the actual creation of the file.

The Inform area will contain the following:

word 1	The total number of data and index blocks in the file if the index blocks are embedded in the file, or, the total number of data blocks if the index blocks are maintained in a separate file.
word 2	The total number of index blocks.
word 3	The total number of data block overflows thus causing a data block split.
word 4	The total number of index block overflows thus causing an index block split.
word 5	The total number of data entry records read.
word 6	The total number of data entry records written.
word 7	The total number of existing data entry records.

4) INFORM (FCT, Inform area)

This command gives the user information about a particular file relative to the execution of the user's last 'OPEN' command.

FCT	The File Control Table
Inform area	The description is given under 'CLOSE' The contents of the 'inform area' will be a set of numbers relative to the user's last 'OPEN'. At 'OPEN' time, these counters are reset to zero and incremented/decremented for each user action on this file. Note that word 7 should be interpreted as the number of insertions and deletions made to the file. If this number is negative, then the user made more deletions then insertions.

5) OPENAP

This command 'opens' the access package to the user. It sets up contingency processing linkages for the user. The user must execute this command only once before all other access package commands.

6) CLOSEAP

This command 'closes' the access package to the user. It clears all locked records on all files that the user may have neglected to close. The user should execute this command once before terminating his job.

The following commands have parameter lists which are dependent on the file organization. They are:

1) RDRAN (FCT, Key entry area, < Key size , > Data entry area <, Data size > <, Role >)

This command retrieves a data entry for a given random key.

FCT	The File Control Table
Key entry area	The location in the user's memory containing the search key.
< Key Size, >	The character length of the key if a variable key file.
Data entry area	The location in the user's memory where the data entry will be placed.
< ,Data size >	The location in the user's memory where the size of the data entry (rounded to the next highest multiple of 6 characters) will be placed for a variable data file.
< , Role >	For a file with role codes, the role number of the data entry to be retrieved.

2) RD SEQ (FCT, Key entry area,< Key size,> Data entry area < , Data size >)

This command retrieves the data entry for the next sequentially ascending key for a given random key.

FCT	The File Control Table
Key entry area	The location in the user's memory containing the key which causes the search for the next ascending key. The new key when found will be placed in this area.
< Key size, >	The character length of the input key. This is replaced by the character length of the retrieved key for variable key files.
Data entry area	The location in the user's memory where the data entry will be placed.

<, Data size>	The location in the user's memory where the size of the data entry (rounded to the next highest multiple of 6 characters) will be placed for a variable data file.

3) RDLISTRAN (FCT, Key entry area, < Key size, > Data entry area, # of entries)

This command retrieves the data entries for all the roles that may exist for a given random key.

FCT	The File Control Table
Key entry area	The location in the user's memory containing the search key.
Key size, >	The character length of the key for a variable key file.
Data entry area	The location in the user's memory where all the data entries found will be placed.
# of entries	The location in the user's memory where the number of data entries found and returned to the data entry area will be placed.

4) RDLISTSEQ (FCT, Key entry area, < Key size, > Data entry area, # of entries)

This command retrieves the data entries for all the roles that may exist for the next sequentially ascending key for a given random key.

FCT	The File Control Table
Key entry area	The location in the user's memory containing the key which causes the search for the next ascending key. The new key when found will be placed in this area.
< Key size, >	The character length of the input key. This is replaced by the character length of the retrieved key for variable key files.

Data entry area	The location in the user's memory where all the data entries found will be placed.
# of entries	The location in the user's memory where the number of data entries found and returned to the data entry area will be placed.

5) RDRANLK (FCT, Key area,< Key size, > Data area < , Data size > <, Role >)

6) RDSEQLK (FCT, Key area,< Key size, > Data area < , Data size >)

7) RDLISTRANLK (FCT, Key area,< Key size, > Data area, # of entries)

8) RDLISTSEQLK (FCT, Key area,< Key size, > Data area, # of entries)

These four commands have parameter lists which are defined in the same manner as the standard retrieval functions 1 through 4. They also perform the same function but, in addition, a check is made to determine if the retrieval entry is locked by other users. If the entry is locked, it is not returned and an indication of this is passed to the caller. If the entry is not locked, then it is returned to the caller and a logical lock is placed on the key entry. Each user may place an arbitrary number (defined at system generation) of logical locks on each file. A lock can only be removed by performing a MODIFY or DELETE command utilizing the locked key. The same user who initially locked the entry must unlock it.

The normal retrieval commands 1 through 4 do not check and do not set locks. Thus, they act as simple "inquiry only" requests.

9) INSERT (FCT, Key entry area,< Key size, > Data entry area <, Data size > <, Role >)

This command adds either a key/data entry pair or a data entry qualified by role number to a file.

FCT	The File Control Table

Key entry area	The location in the user's memory area containing the key to be inserted, or in the case of a file with role codes, possibly the search key. If a role code is specified in the parameter list, a search is made to find the key within the file. If the key is found, then a new data entry insertion only is attempted. If the key is not found, then the key is also inserted as well as the specified role code data entry.
< Key size, >	The character length of the new key if a variable key file.
Data entry area	The location in the user's memory containing the new data entry.
<, Data size >	The character length of the new data entry if a variable data file.
<, Role>	The role code number that is to be stored (and subsequently used as a secondary key) along with the new data entry if the file is defined as one having role codes.

NOTE: See description of TYPE 5 file organization.

10) DELETE (FCT, Key entry area<, Key size><, Role >)

This command deletes either a key/data entry pair or a data entry qualified by role code number from a file.

FCT	The File Control Table
Key entry area	The location in the user's memory area containing the search key.
<, Key Size >	The character length of the key entry if a variable key file.

<, Role >	The data entry specified by this role code number is to be deleted for a file defined with role codes. If more than one data entry exists for the given key, then only the specified role data entry is deleted. If the only data entry that exists for the given key is that which is specified by the role parameter then that data entry and the key entry are both deleted.

11) MODIFY (FCT, Key entry area, < Key size,> Data entry area <, Data size> <, Role >)

This command updates a data entry possibly specified by a role number for a given random key.

FCT	The File Control Table
Key entry area	The location in the user's memory area containing the search key.
< Key size,>	The character length of the key entry if a variable key file.
Data entry area	The location in the user's memory area containing the new data entry which will be used to overwrite the old data entry.
<, Data size >	The character length of the new data entry if a variable data file.
<, Role >	The role number of the old data entry to be updated if a role code file.

FILE TYPE VERSUS APPLICABLE COMMAND LIST

The following is a list of commands with their respective parameter lists for each file type:

ALL TYPES

```
DEFNFILE (FCT, Data blk size, File Type, Key length, Data length,
        Role Flag <, Index Filename, Index blk size >)

OPEN (FCT, Read/Write Flag)

CLOSE (FCT, Inform area)

INFORM (FCT, Inform area)

OPENAP

CLOSEAP
```

TYPE 1

```
RDRAN  (FCT, Key entry area, Data entry area)
RDSEQ  (FCT, Key entry area, Data entry area)
RDRANLK  (FCT, Key entry area, Data entry area)
RDSEQLK  (FCT, Key entry area, Data entry area)
INSERT (FCT, Key entry area, Data entry area)
DELETE (FCT, Key entry area)
MODIFY (FCT, Key entry area, Data entry area)
```

TYPE 2

```
RDRAN   (FCT, Key entry area, Key size, Data entry area)
RDSEQ   (FCT, Key entry area, Key size, Data entry area)
RDRANLK (FCT, Key entry area, Key size, Data entry area)
RDSEQLK (FCT, Key entry area, Key size, Data entry area)
INSERT  (FCT, Key entry area, Key size, Data entry area)
DELETE  (FCT, Key entry area, Key size)
MODIFY  (FCT, Key entry area, Key size, Data entry area)
```

TYPE 3

```
RDRAN   (FCT, Key entry area, Data entry area, Data size)
RDSEQ   (FCT, Key entry area, Data entry area, Data size)
RDRANLK (FCT, Key entry area, Data entry area, Data size)
RDSEQLK (FCT, Key entry area, Data entry area, Data size)
INSERT  (FCT, Key entry area, Data entry area, Data size)
DELETE  (FCT, Key entry area)
MODIFY  (FCT, Key entry area, Data entry area, Data size)
```

TYPE 4

```
RDRAN   (FCT, Key entry area, Key size, Data entry area, Data size)
RDSEQ   (FCT, Key entry area, Key size, Data entry area, Data size)
RDRANLK (FCT, Key entry area, Key size, Data entry area, Data size)
RDSEQLK (FCT, Key entry area, Key size, Data entry area, Data size)
INSERT  (FCT, Key entry area, Key size, Data entry area, Data size)
DELETE  (FCT, Key entry area, Key size)
MODIFY  (FCT, Key entry area, Key size, Data entry area, Data size)
```

TYPE 5

```
RDRAN   (FCT, Key entry area, Data entry area)
RDSEQ   (FCT, Key entry area, Data entry area)
RDRANLK (FCT, Key entry area, Data entry area)
RDSEQLK (FCT, Key entry area, Data entry area)
INSERT  (FCT, KEY/DATA entry area) See TYPE 5 file description
DELETE  (FCT, Key entry area)
MODIFY  (FCT, KEY/DATA entry area)
```

TYPE 6

```
RDRAN   (FCT, Key entry area, Data entry area, Role)
RDLISTRAN (FCT, Key entry area, Data entry area, # of entries)
RDLISTSEQ (FCT, Key entry area, Data entry area, # of entries)
RDRANLK (FCT, Key entry area, Data entry area, Role)
RDLISTRANLK (FCT, Key entry area, Data entry area, # of entries)
RDLISTSEQLK (FCT, Key entry area, Data entry area, # of entries)
```

INSERT	(FCT, Key entry area, Data entry area, Role)
DELETE	(FCT, Key entry area, Role)
MODIFY	(FCT, Key entry area, Data entry area, Role)

<u>TYPE 7</u>

RDRAN	(FCT, Key entry area, Key size, Data entry area, Role)
RDLISTRAN	(FCT, Key entry area, Key size, Data entry area, # of entries)
RDLISTSEQ	(FCT, Key entry area, Key size, Data entry area, # of entries)
RDRANLK	(FCT, Key entry area, Key size, Data entry area, Role)
RDLISTRANLK	(FCT, Key entry area, Key size, Data entry area, # of entries)
RDLISTSEQLK	(FCT, Key entry area, Key size, Data entry area, # of entries)
INSERT	(FCT, Key entry area, Key size, Data entry area, Role)
DELETE	(FCT, Key entry area, Key size, Role)
MODIFY	(FCT, Key entry area, Key size, Data entry area, Role)

ERROR CODES

The following codes comprise a complete list of error conditions:

Error #	Cause
1.	Access package already initialized. Request ignored.
2.	Access package not initialized. Run terminated.
3.	No such function. Request ignored.
4.	Incorrect number of arguments for function. Request ignored.
5.	The file has been defined by another concurrent run or a second attempt is being made by the calling program to redefine the file. Request ignored.
6.	Data block size exceeds maximum. Request ignored.
7.	Key length size exceeds maximum. Request ignored.
8.	Data length size exceeds maximum. Request ignored.
9.	The key/data format type does not exist. Request ignored.
10.	Role codes not permitted for key/data type defined. Request ignored.
11.	Data block size less than minimum. Request ignored.
12.	Key length size less than minimum. Request ignored.
13.	Data lengtn size less than minimum. Request ignored.
14.	Index file not assigned properly or internal CSF$ syntax error. Request ignored.
15.	Index block size exceeds maximum. Request ignored.
16.	Data block size not a multiple of disc prep size. Request ignored.
17.	Index block size less than minimum. Request ignored.
18.	Index block size not a multiple of disc prep size. Request ignored.
19.	Data file not assigned properly or internal CFS$ syntax error. Request ignored.
20.	File control table full indicating maximum number of files are open. Request ignored.

Error #	Cause
21.	No buffer space available for file. Request ignored.
22.	Attempt to close a file which is already closed. Request ignored.
23.	Attempt to close a file which has not been opened. Request ignored.
24.	The file cannot be accessed since it no longer appears in the internal File Control Table. Probable system software malfunction.
25.	No statistics gathered since the file has not been opened.
26.	Attempt to read a file which has not been opened. Request ignored.
27.	Role code argument out of range. Request ignored.
28.	The file is empty. Request ignored.
29.	Block number read not block number desired. Integrity of file destroyed.
30.	Block type read not block type desired. Integrity of file destroyed.
31.	No record exists for the given key.
32.	A RDSEQ request not permitted on a file containing role codes.
33.	A 'Read List' type request not permitted on a file without role codes.
34.	End of file reached while performing a sequential read.
35.	Attempt to 'change' a file declared 'read only'.
36.	File is being initially loaded. No requests permitted by other users. 'Insert' only by originator of load.
37.	Attempt to 'insert a key/data record for which the key portion already exists. Also for a file with role codes, the key and role code already exist.

Error #	Cause
38.	Attempt to insert a key/data record whose total length exceeds the data block size. For a file with role codes, the total length includes all existing role code entries.
39.	A non-ascending key was given during an initial load insertion. Request ignored.
40.	Attempt to open a file that is already open. Request ignored.
41.	Record is currently locked by another user.
42.	Maximum number of users are active. Cannot support another.
43.	User is not active within the access package. Request ignored.
44.	Maximum number of records already locked by this user. Record returned to user but not locked.

ACCESS PACKAGE INTERNAL TABLES AND CONTROL BLOCKS

The following is a layout and description of the various core tables and control blocks used by the peripheral file access package (PFAP).

1. THE PFAP COMMON FILE CONTROL TABLE OR PFAPFCT

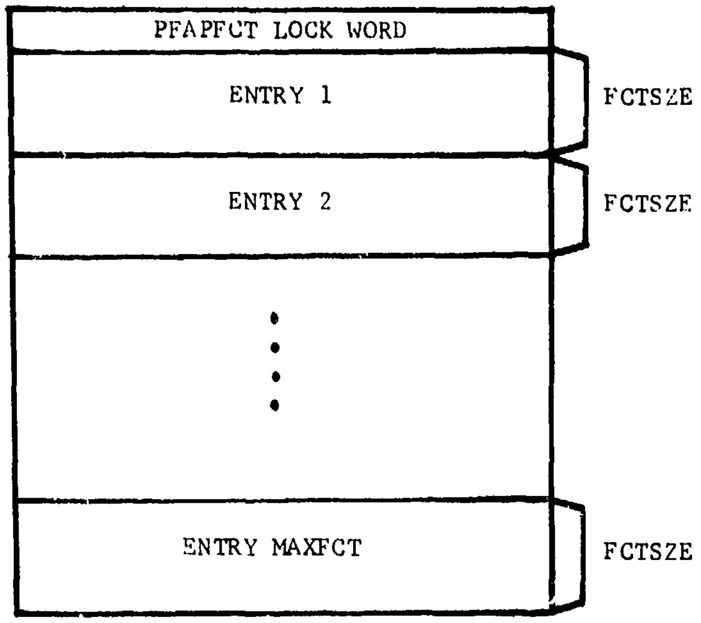


Table B-1 The PFAPFCT

The PFAPFCT shown in Table B-1 is used by the access package to control the activities on each data file which is "opened" for access by any user. The table is partitioned into a fixed number of entries determined by the variable MAXFCT at assembly time. Each entry is defined to be of size FCTSZE. Access into the table is controlled by a "lock" word (a Test and Set "queue" type word). Each new file which is opened by a user is entered into the first available "entry" space in the PFAPFCT. When a file is no longer needed by any user, its entry position is cleared for re-use.

Table B-2 is a layout of an individual "entry" in the PFAPFCT.

0	'THIS ENTRY' LOCK WORD			
1	DATA			
2	FILE NAME			
3	I/O			
4	PACKET			
5	AREA			
6				
7	NUSERS			
8	SPARE	FT	DBLSZE	IBLSZE
9	R C	SPARE	KLEN	DLEN
10	OFNXKY	DBLSEC	IBLSEC	
11	NBLKS			
12	NIBLKS			
13	NDBLOV			
14	NIBLOV			
15	NREAD			
16	NWRITE			
17	NRECDS			
18	NHIGH			
19	NLEVEL			
20	INDEX			
21	FILE NAME			
22	I/O			
23	PACKET			
24	AREA			
25				
26	LASTIBL	LASTDBL		
27	HIBUFF	SCRBUFF		
28	SPARE	DATBUFF		
29	I L	SPARE	ILSZE	ILKYADR
30	ILRUNID			
31	RECORD			
.	LOCK			
.	AREA			
.				
.				
FCTSZE				

TABLE B-2
Layout of an entry in the PFAPFCT

Definitions for variable names shown in the "PFAPFCT ENTRY" layout are as follows:

NUSERS	=	number of concurrent users who have this file currently open.
FT	=	file type, a number from 1 to 7 described under "General File Organization" and DEFNFILE primitive command. (6 bits)
DBLSZE	=	physical data block size in words. (12 bits)
IBLSZE	=	physical index block size in words. (12 bits)
RC	=	role code flag, a 1 bit field when set indicates file has role codes and FT field must be 6 or 7, when reset indicates file has no role codes and FT may be 1 through 5.
KLEN	=	The maximum key length in characters for variable key files (FT = 2, 4, or 7) or the standard key length for fixed key files (FT = 1, 3, 5 or 6). (12 bits)
DLEN	=	The maximum data length in characters for variable data files (FT = 3 or 4) or the standard data length for fixed data files (FT = 1, 2, 5, 6, 7). (12 bits)
OFNXKY	=	Used as an "offset to next key" length in words for files with FT = 1 or 5. Since no Record Control Words are stored in data blocks for these files, this field provides a convenient incremental value to skip from one key/data entry to another. (12 bits)
DBLSEC	=	number of Fastrand sectors necessary to hold a physical data block. (12 bits)
IBLSEC	=	number of Fastrand sectors necessary to hold a physical index block. (12 bits)
NBLKS	=	total number of data and index blocks within a file if the index blocks are embedded within the data file or, the total number of data blocks if the index blocks are stored in a separate index file.
NIBLKS	=	total number of index blocks for this file.
NDBLOV	=	total number of data block overflows (which caused a block split to occur).
NIBLOV	=	total number of index block overflows (which caused a block split to occur).
NREAD	=	total number of key/data entries read.

NWRITE	=	total number of key/data entries written.
NRECDS	=	total number of data entries within the file.
NHIGH	=	the number of the highest level index block.
NLEVEL	=	the number of index levels supporting this file.
LASTIBL	=	the number of the last accessed index block which resides in the scratch index buffer. (18 bits)
LASTDBL	=	the number of the last accessed data block which resides in the data block buffer. (18 bits)
HIBUFF	=	address of the highest level index block buffer. (18 bits)
SCRBUFF	=	address of the scratch index block buffer. (18 bits)
DATBUFF	=	address of the data block buffer. (18 bits)
IL	=	"initial load" flag, a 1 bit field when set indicates this file is being initially loaded by a unique user.
ILSZE	=	computed "initial load" size in words. This value is dependent on DBLSZE. The standard load factor percentage is defined by LODFAC at assembly time. Then ILSZE = DBLSZE*LODFAC. (12 bits)
ILKYADR	=	relative address of the last key entered in the data block buffer. This field is used when checking for sequentially ascending keys at initial load time. (12 bits)
ILRUNID	=	the generated "runid" of the user who initiated the "initial load" of this file. All other users are prohibited from performing any access to this file during an "initial load."

Table B-3 is an expansion of the layout of the RECORD LOCK AREA within an "entry" of the PFAPFCT.

LKENTRY	=	the sequential position number within the data block of the key that is locked. Also known as the entry number. For a role code file, the key lock extends across all data entries stored against the key. (12 bits)
LKBLCK	=	the number of the data block in which the locked key occurs. (18 bits)
MAXLOCKS	=	maximum number of records that may be simultaneously locked by any user on any one file. This is a variable defined at assembly time and extends across all files opened to the access package.
MAXUSERS	=	maximum number of concurrent users that can be supported by the access package. This is also a variable to be defined at assembly time.

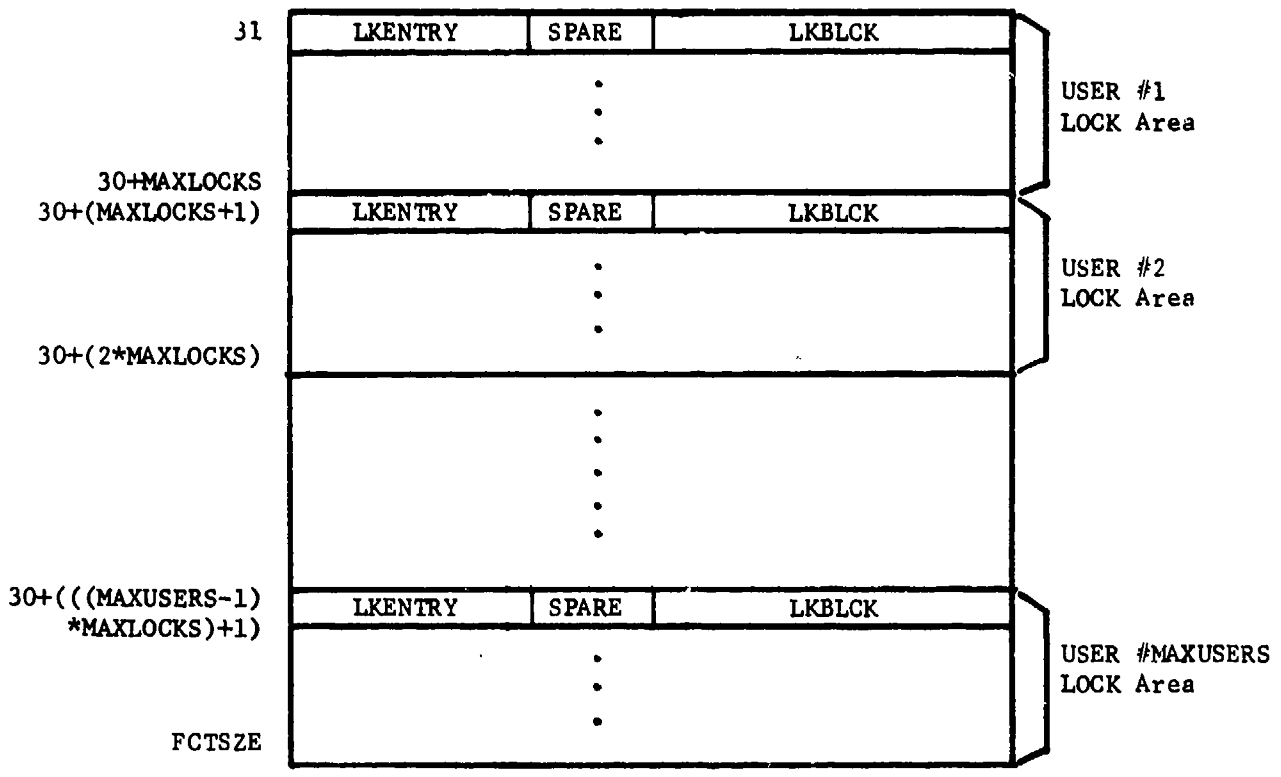


TABLE B-3

Lock Area within an entry of the PFAPFCT

FCTSZE	=	total size of an entry in the PFAPFCT. A variable defined at assembly time as FCTSZE = 30 + (MAXLOCKS*MAXUSERS)

An entry in the PFAPFCT is considered to be available if the Data Filename words contain binary zeros. The "next available entry" is considered to be the first entry encountered which has its Data Filename words set as binary zeros.

If the index blocks are embedded in the data file, then the Index Filename words will contain the same name as the Data Filename words. The I/O accesses will still be controlled through the Index File packet as if the index filename were unique.

Access to each "entry" in the PFAPFCT is controlled by a "lock" word (a Test and Set "queue" type word). Thus there are two levels of control into the PFAPFCT; one at the table level to prevent conflicts when deleting and inserting whole "entries" and one at the "entry" level to prevent conflicts between concurrent users on the same file.

2. THE RUNID TABLE

The RUNID Table is a list of the generated user "runid" names obtained from each user's PCT. Each user who performs an OPENAP command has an entry made into the next available slot in the table. Each user is removed from this table when he performs a CLOSEAP.

For commands involving testing and/or setting of record locks (such as RDRANLK, DELETE, etc.), the user's RUNID is obtained from his PCT, found in the RUNID table, and the resulting relative table location used as an index into the "record lock area" of the PFAPFCT entry for the file being accessed.

Access to the RUNID Table is controlled by a "lock" word (a Test and Set "queue" type word). It is used to prevent conflicts when users are added or deleted from the table.

Table B-4 is a layout of the RUNID Table.

0	RUNID TABLE LOCK WORD
1	GENERATED RUNID
2	GENERATED RUNID
•	•
•	•
•	•
MAXUSERS	GENERATED RUNID

TABLE B-4

The RUNID Table

3. THE COMMON BUFFER TABLE OR CBT

The CBT functions as a contiguous buffer pool out of which each file that is currently opened has assigned to it an area large enough to support 2 index block buffers and 1 data block buffer

Table B-5 shows the layout of the CBT.

LASTCBT	=	size in words of the CBT defined at assembly time. It should be computed as LASTCBT = MAXFCT*3*BUFMEAN where MAXFCT is the maximum number of file entries in the PFAPFCT and BUFMEAN is the probable mean size of the data and index blocks to be used within all files.

4. THE AVAILABLE BUFFER TABLE

The AVLBT is a table of addresses and word lengths defining those areas in the CBT which are available for use. When a file is closed and no longer needed for any other user, the buffer space must be returned to the pool. Since the buffer space was allocated on a first in, first out basis, the CBT is likely to become fragmented if files are closed (and not needed by other users) on a random basis. Initial buffer allocation is controlled by a variable called NEXTCBT which contains the next available relative CBT address (initially 0). If the difference between LASTCBT and NEXTCBT is large enough to hold the requested buffer needs, then the absolute CBT address is generated and returned to the caller. NEXTCBT is then increased by the number of words just allocated. However when space is returned NEXTCBT cannot easily be modified, so the AVLBT is used to point to the fragmented free space. If the difference between LASTCBT and NEXTCBT is too small to satisfy a buffer request, then the AVLBT is searched to find an entry whose "# of words" field is large enough to satisfy the caller's request The appropriate address is passed to the caller and adjustments made in the AVLBT entry to compensate for the size needed.

When space is returned to the AVLBT, an attempt is made to find an entry whose "next available CBT address" field is contiguous to the "ending buffer address" of the space being returned. If such an entry is found then the AVLBT entry is modified to include the new space being returned.

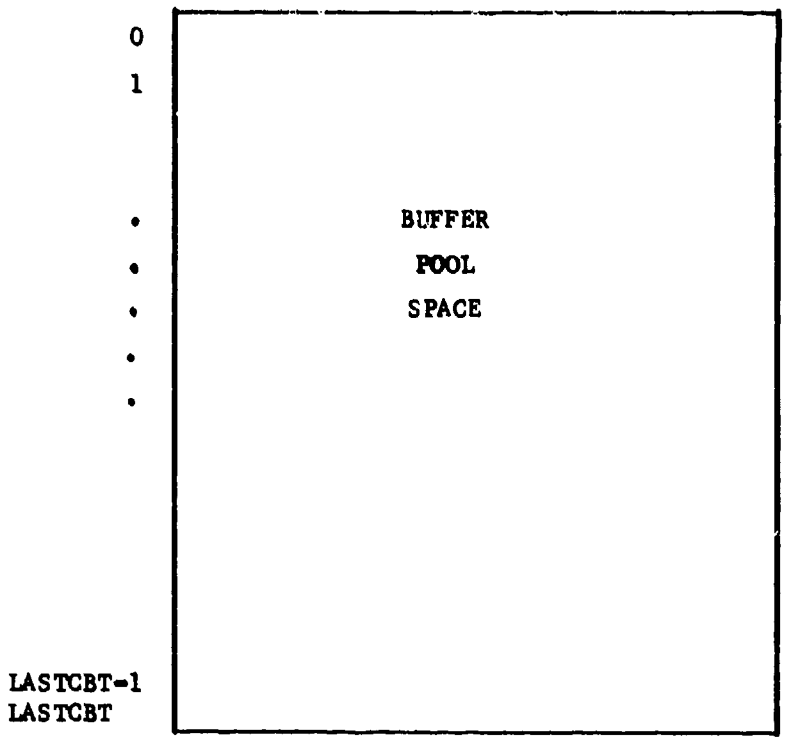


TABLE B-5

The Common Buffer Table

Access to the AVLBT and in general to allocation and release of space in the CBT is controlled through a "lock" word (a Test and Set "queue" type word).

Table B-6 is a layout of the AVLBT.

Definitions for variable names shown in the AVLBT layout are as follows:

AVWRDS	=	number of available words in the CBT fragment pointed to by NXAVCBT. (18 bits)
NXAVCBT	=	relative address in the CBT of the next available fragment. (18 bits)
AVLBTSZE	=	maximum number of entries in the AVLBT. This variable is defined at assembly time.

5. THE OVERFLOW BUFFER TABLE OR ØBT

The OBT is used as a working buffer area whenever a data block or index block split occurs due to insertion of data causing an overflow situation. The OBT is also used as a I/O packet area for DEFNFILE commands.

Access to the OBT is controlled through a "lock" word (a Test and Set "queue" type word). The lock word is generally used to queue more than 1 block split occuring concurrently. The size of the OBT is defined at assembly time as OBTSZE = 2*BUFMAX where BUFMAX is the maximum size in words of any index or data block likely to be defined for any file.

Table B-7 is a layout of the OBT.

6. THE OBT OVERFLOW AREA

The OBT has an overflow area which again is used during the block split procedure. This area is used to hold the Record Control Word(s) (RCW) and key(s) which need be inserted into the next successively higher index block. At the data block level, a maximum of 2 keys may be entered into this area while at the index block level, only 1 key may be entered due to the size restrictions placed on key and data lengths.

Table B-8 shows a layout of the OBT OVERFLOW AREA.

0	AVLBT LOCK WORD	
1	AVWRDS	NXAVCBT
2	AVWRDS	NXAVCBT
⋮	⋮	
AVLBTSZE	AVWRDS	NXAVCBT

TABLE B-6

The Available Buffer Table

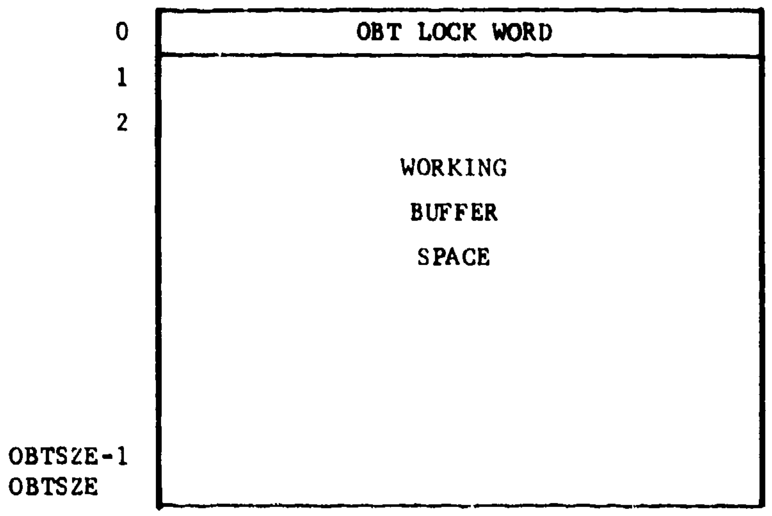


TABLE B-7

The Overflow Buffer Table

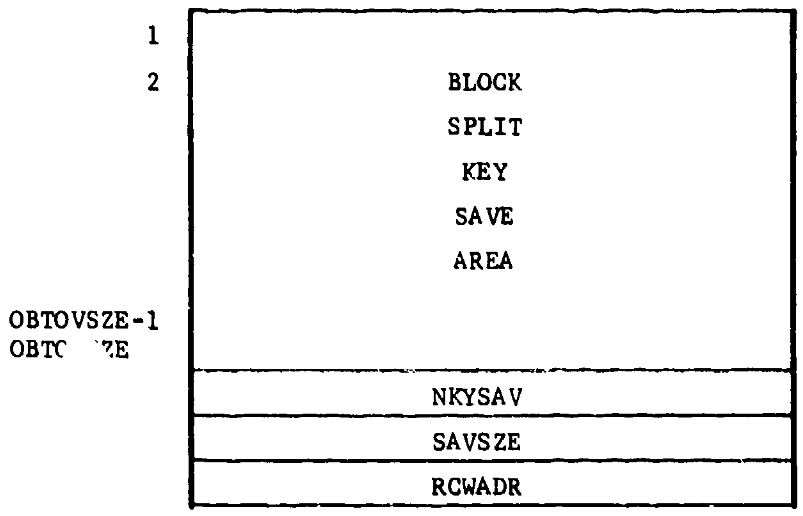


TABLE B-8

The OBT Overflow Area

Definitions for variable names are as follows:

NKYSAN	=	number of keys being saved in the area while a block split is being performed.
SAVSZE	=	total number of words (both key(s) and RCW(s)) being saved during a block split.
RCWADR	=	relative address of the RCW in an index block after whose associated key the "saved key" should be inserted.
OBTOVSZE	=	total size in words of the scratch save area. This is defined at assembly time and must be twice the size of the longest key that is likely to occur in any file.

7. THE USER'S FILE CONTROL TABLE OR FCT

Each user who is manipulating a file via the access package needs an FCT within his own core area. It is used to hold the Filename, the relative statistics pertaining to the usage of this file, and certain internal control bits.

Table 3-9 shows a layout of the USER FCT.

Definitions of the variables shown in Table 8 are as follows:

RSVD	=		the reserved control bits by which the access package can determine what the state of the file is (6 bits)
	=	0	implies file has not been "opened" at all, therefore first call must be OPEN or DEFNFILE.
	=	1	implies file has been "opened".
	=	2	implies file has been "opened" and accessed via any read/write request.
	=	77_8	implies file has been "closed" and can be subsequently "reopened".
R/W	=		the read/write bit. If 0, the file may be read or written. If 1, the file is considered to be a "read-only" file.

Words 3 through 9 are the statistics cells and have the same definitions as those in the PFAPFCT.

8. THE FILE INFORMATION BLOCK OR FIB

Each data file has a control record called the FIB residing in the first Fastrand sector of the file on mass storage. It completely defines the organization and status of the file.

Table B-10 shows the layout of the FIB.

The DEFNFILE command is used to initially define the subfields in the FIB. An initial FIB is written into the file at that time. The OPEN command causes the FIB to be read into the PFAPFCT entry assigned to this file (assuming

0	DATA		
1	FILE NAME		
2	RSVD	SPARE	R/W
3	NBLKS		
4	NIBLKS		
5	NDBLOV		
6	NIBLOV		
7	NREAD		
8	NWRITE		
9	NRECDS		

TABLE B-9
The USER FCT

0	SPARE	FT	DBLSZE	IBLSZE
1	RC	SPARE	KLEN	DLEN
2	OFNXKY		DBLSEC	IBLSEC
3	NBLKS			
4	NIBLKS			
5	NDBLOV			
6	NIBLOV			
7	NREAD			
8	NWRITE			
9	NRECDS			
10	NHIGH			
11	NLEVEL			
12	INDEX			
13	FILE NAME			

TABLE B-10
The FILE INFORMATION BLOCK

the file has not been already "opened" by another user). As each user performs a CLOSE, a new copy of the FIB is written from the PFAPFCT entry to the file assuring an up to date status of the file. Also whenever a data block split occurs, the FIB is updated since the data, index and level counters may have changed.

Note that the FIB resides within the data file. There is no FIB within a separate index file. The relative sector number computation involving data block numbers must compensate for the existence of the FIB. Moreover, since the index blocks may be embedded within the data file, the sector computaton must also compensate for the FIB. Separate index files therefore have a Fastrand sector allocated to a dummy FIB (which is never read or written) to eliminate having the index block I/O routines having to check whether the blocks are embedded or not.

The definitions of the variables shown in Table B-10 are exactly as described under the PFAPFCT.

9. ALLOCATION OF STORAGE WITHIN MODULE PFAP

The one non-reentrant module PFAP acts as a buffer between the original caller and the reentrant portion of the access package. Its main function is to save the caller's registers, then pass on to the reentrant modules the address of the caller's parameter list and the address of the register save and temporary scratch save area. This scratch save area can be used by the reentrant modules to store counters, flags, etc. of a temporary file dependent nature. The maximum size of this area is defined at assembly time but shold not exceed 100 words.

Table B-11 shows a layout of the storage area.

The OPENAPFLAG contains a 1 bit field which if 0 indicates no OPENAP has been performed for this user; if 1, indicates an OPENAP has been done.

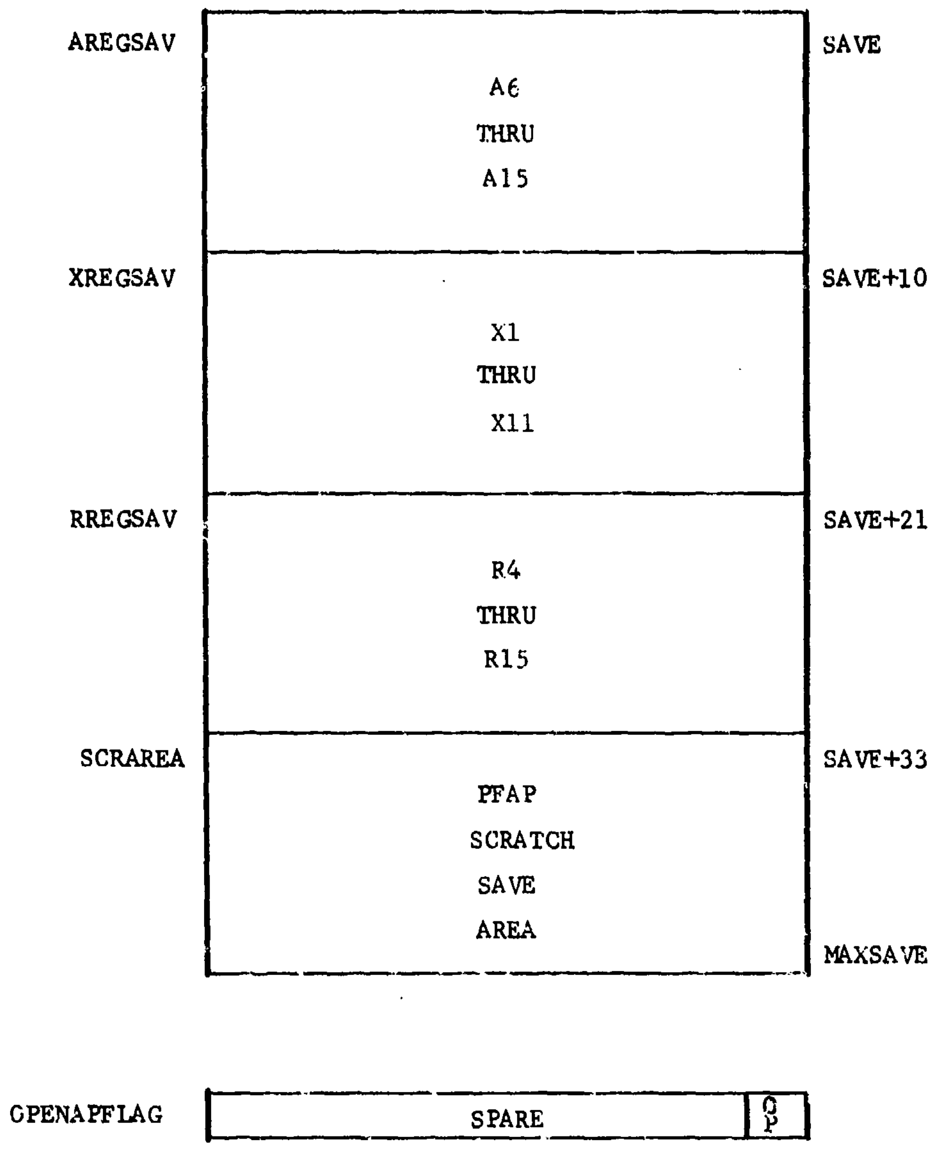


TABLE B-11

Storage Area within PFAP non-reentrant module

APPENDIX C. STIS DIRECTORY FUNCTIONAL SPECIFICATION

STIS DIRECTORY FUNCTIONAL SPECIFICATION

1. DIRECTORY CAPABILITIES

Directory services are designed to allow the encoding and decoding of all system words and terms (of one or more words) used in STIS as entity names, attributes names, attribute values, commands, etc. These system words and terms are listed in integrated ordered lists (the Word and Term Encoding Tables) so that an input language scanner can recognize and determine the role played by every word apt to be input (including the attribute associated with a value term), and also determine when a value term or entity name is not unique.

1.1 DIRECTORY TABLES

The Directory Tables are shown in Figure C-1 and listed in Table 1. They are described below.

1.1.1 Word Encoding Table (WET)

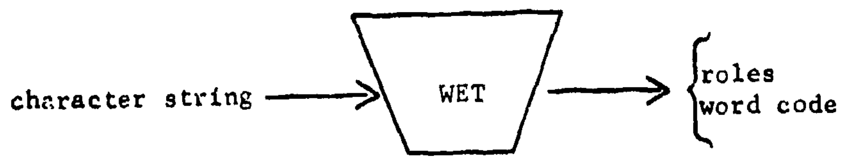
The WET entries are three 36-bit cells each.

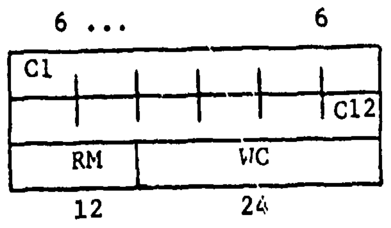
The input argument is a word or word fragment of one to twelve 6-bit characters. If the word or word fragment is longer than 12 characters, then C12 is a hyphen "-" designating a continued word fragment. Words of fewer than 12 characters are left justified and filled with blanks (word space) to C12. Entries are ordered by input argument and words of arbitrary length can be accommodated as a series of word fragments.

The Role Map is a 12-bit field which is used only if the word is 12 characters or less and represents a one-word term. If the word is a fragment of a term, then all 12 bits are "0". The interpretation of the Word Map for a term is discussed below in connection with the Term Encoding Table.

The Word Code is a 24-bit field, assigned in a sequence as words (and terms) are entered into the system.

(a) <u>WORD ENCODING TABLE</u> =



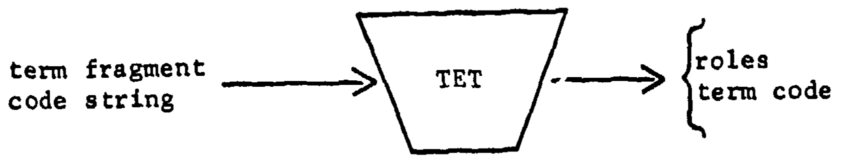


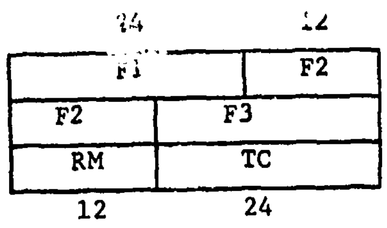
C1,...,C12 = character string

RM = Role Map

WC = Word (or word fragment) code

(b) <u>TERM ENCODING TABLE</u> =



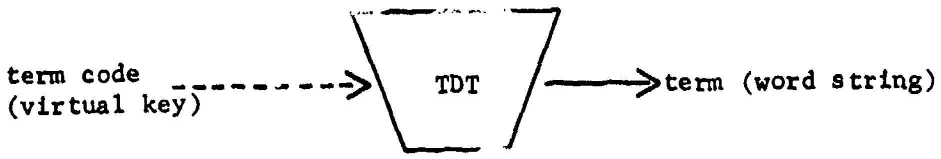


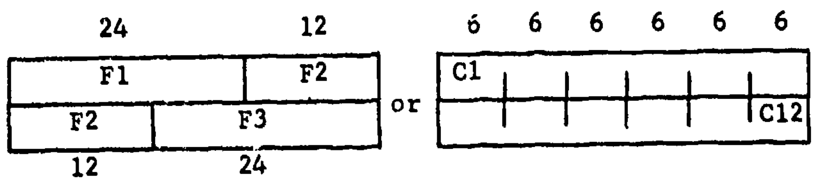
F1, F2, F3 = term fragment, word, word fragment codes

RM = Role Map

TC = Term or term fragment code

(c) <u>TERM DECODING TABLE</u> =





Term Fragment Format   Character Format

Figure C-1 Directory Tables

TABLE 1

DIRECTORY TABLES

Word Encoding Table (WET)
Term Encoding Table (TET)
Term Decoding Table (TDT)

TABLE 2

ROLE MAP INDICATORS

i	Role
1	Noise
2	System Command
3	User Command
4	Attribute
5	Attribute Value
6-12	Unused ("0")

1.1.2 Term Encoding Table (TET)

The TET entry is three 36-bit cells. The input argument is a string of three 24-bit fields, F1, F2, and F3 in which bit one is always "0" and bit two is a Word Code Indicator (WCI) which is set "1" for a word or word fragment code and "0" for a term fragment code. The TC is a term or term fragment code and is 24 bits long. (TC corresponds to WC in the WET entry.)

The Role Map is a 12-bit field which is all "0" if the input argument represents a term fragment. If the input argument is a complete term then RM represents the roles of the term. If bit i (i = 1,...,12) is "1" then the term role is role i. For a term, one or more of bits 1 through 12 will be set "1". The role interpretations are listed in Table 2.

1.1.3 Term Decoding Table (TDT)

The TDT entry is two 36-bit cells in one of two formats, either three 24-bit fields or twelve 6-bit characters. The argument is a term (term fragment) code or a word (word fragment) code. Since these will form a dense set (assigned in sequence by the system) they will be interpreted as an entry number and will not be stored in the entry (i.e., the code is a virtual key). If the Word Code Indicator (WCI) in the key is "0" then the entry is in Term Fragment Format, otherwise, it is interpreted as a character string.

In decoding each term fragment field the WCI (bit two) is examined to determine whether the corresponding TDT entry should be interpreted in Term Fragment or Character format. In this way terms made up of arbitrarily long strings of arbitrarily long words may be decoded.

1.2 DIRECTORY SUBSYSTEM COMMANDS

Each high level Subsystem command is generally decomposed into a series of lower level commands which act on words, word fragments, or term fragments. These lower level commands then either search or make entries in the various tables like the Term Encoding Table, Word Encoding Table, etc. This modular nucleus of word and fragment oriented commands will allow construction of other high level term oriented commands when needed in the future.

1.2.1 Term Oriented Commands

A term is defined to be any string of English language words each of arbitrary length and separated by blanks or any other designated delimiter(s). The end of term is signaled by an end of term sentinel following an optional last end of word delimiter. Information will be requested at the term level as a result generally of a query by an intelligence analyst. The following commands are defined to act upon term data.

- Retrieve Term Code and Role (term) = [term code, (role 1 <, role 2...>)]

  This command will find the numeric term code associated with the given English language term if the term code exists. The function will also supply a list of roles that define the various ways the term is being used. If the term code exists then at least one role will be returned.

  Each word in the term is encoded into its appropriate word code then the lower level command 'Find Term Code and Role' is used to fetch the appropriate term code. If the term consists of a single word then the word code found in the initial word encoding process is the desired term code.

- Retrieve Term Code (term, role) = term code

  This command will find and return the numeric term code associated with the given English language term provided the term exists for the specified role in the Term Encoding Table. The role acts as a qualifier for the term. The command provides the ability to answer questions such as "...is this term used as an attribute...?"

  The function 'Retrieve Term Code and Role' is called to find all the roles for the given term. A match for the desired role is then sought in the role list found.

- Find Term Code and Role (word code <, word code, ... >) = [term code, (role 1 <, role 2... >)]

  This command will find the numeric term code associated with the given string of numeric word codes if the term code exists. The function will also supply a list of roles that define the various ways in which the term is being used. If the term code exists, then at least one role will be returned.

This command performs the same functions as 'Retrieve Term Code and Role' but at a lower level of input. Here it is assumed that the English term has been encoded into its word components.

The word codes are reencoded recursively via the 'Reencode Fragments' function into the term code desired.

- Find Term Code [(word code <, word code... >), role] = term code

  This command will find and return the numeric term code associated with the given string of numeric word codes provided the term code exists for the specified role in the Term Encoding Table.

  Again, this command performs the same functions as 'Retrieve Term Code' but at a lower level of input.

  The function 'Find Term Code and Role' is called to find all the roles for the given term. A match for the desired role is then sought in the role list found.

  This command and 'Find Term Code and Role' could be used by a language processor which scans and encodes a term a word at a time into a string of word codes and subsequently desires the final term code.

- Insert Term (term, role) = term code

  This command will effectively add the English term qualified by role to the Term Encoding Table. The term code which is assigned to it by the system will be returned. If the term already exists, then an additional role is being defined in which case the pre-existing term code is returned. If both the term and role pre-exist then an error status is returned.

  The term is encoded into its string of word codes. Appropriate entries are made into the dictionary for any new words encountered. The string of word codes are further encoded into term fragments and subsequently into a single unique numeric term code. The appropriate role is set and the entry made in the Term Encoding Table. Entries are also made in the Term Decoding Table during the process to permit proper retranslation into English.

- Equate Term (new term, old term, old role) = term code

This command will find the numeric term code for the given old English language term. It will encode the new term to the point of assignment of a unique term code. At this point, the old term's code number will be assigned as well as the old role type indicated. The "new" entry with the appropriate role set is made in the Term Encoding Table. Entries involving term fragments are made in the Term Decoding Table; however, no final entry involving the "new" term code is made since the old term decoding is considered to be the "reference" term.

The old numeric term code will be returned. However if the old term code or old role does not exist, then an error status is returned.

- Decode Term (term code) = term

  This command decodes the given term code into the English language text of the term. The Term Decoding Table is used in a recursive look-up process until the final text is completely generated.

  The term code is entered as an initial entry in a push down table. A loop is then begun in which the first entry in the push down table is examined. If the entry has the Word Code Indicator bit set, then the entry is decoded into English text via 'Decode Word' function. The text is then concatenated with any previously decoded text. A 'blank' character is added to the text if the word is not marked as 'continued'. This entry is then completely processed and the push down stack is then popped. If the entry does not have the WCI bit set, then the decoding entries are retrieved and entered in the push down stack in place of the first entry. Another iteration in the loop is then begun. When the push down stack is empty the generated text is returned.

1.2.2 Word Oriented Commands

A word is defined as an English language word of arbitrary character length and followed by a blank or other designated delimiter(s). Words are partitioned on twelve character boundaries if they exceed twelve characters. The following commands are defined to act upon words.

- Encode Word (word) = word code

  This command will attempt to find the associated numeric word code for the given English word. The word code is returned if found or an error status is set if not found.

If the word is partitioned due to its size, a fragment code list is generated and the 'Reencode Fragments' function is used to find the unique desired word code.

- Insert Word (word) = word code

  This command will add the English word to the Word Encoding Table. The numeric word code assigned to it by the system will be returned. If the word is partitioned due to its size, then the appropriate fragment entries are also made in the Word Encoding Table. Entries are also made in the Word Decoding Table (which is integrated with the Term Decoding Table) for proper retranslation to English. The Word Code Indicator bit is set for those entries made in the Word Encoding Table. However for fragments which need further encoded entries in the Term Encoding Table, the WCI bit is not set.

- Equate Word (new word, old word)

  This command allows the user to synonymize a new word not in the dictionary with a pre-existing word. The final encoding of the new word is equated to the word code for the old word. This entry is then made in the Word Encoding Table for new words that are not partitioned or in the Term Encoding Table for words which are partitioned. Appropriate entries are made in the Decoding Table for proper retranslation into English. If the old word does not exist, an error status is returned.

- Find Word Code (word fragment) = word code

  This command is used to find a numeric word code associated with a given fragment of an English word. If a single word, as mentioned earlier, contains more than the maximum number of characters representable in the Word Encoding Table, it is partitioned into fragments. Each fragment then is assigned a word code which is retrievable via this command.

  Any word of size less than the maximum number of characters before partitioning may be given as an argument to this command.

  If the word code is not found in the Word Encoding Table, an error status is returned.

- Decode Word (word code) = word

  The character string represented by the given word code is retrieved from the Decoding Table and returned. This function would generally be called by the command 'Decode Term' when it has determined that it has encountered a word code with the Word Code Indicator bit set.

1.2.3 Commands For Words and/or Terms

The following commands accept fragment codes as parameters. These fragment codes may be interpreted as either word fragment codes or term fragment codes depending on the calling command.

- Re-encode Fragments (frag code,...) = Word/Term code

  This command accepts a string of fragment codes and attempts to further encode them into a single code via the Term Encoding Table. The code found in the encoding process is returned. If a code is not found, an error status is returned.

- Insert Fragments {(frag code,...), Word/Term code, role, equate flag}

  This command forms entries for the Term Encoding Table using the fragment code list supplied. Each packet entered into the TET is assigned a new code number. The last entry into the TET is given the Word/Term code number supplied in the parameter list. The role supplied is also given to this final entry.

  As each entry is made into the TET, an appropriate entry is also made in the Term Decoding Table. However no entry is made in the TDT for the last TET entry if the 'equate' flag is set. The 'equate' flag is set whenever 'Equate Term' or 'Equate Word' is the calling function.

1.2.4 Miscellaneous Functions

Several low level modules are defined in the support of the commands outlined in 1.2.1, 1.2.2, and 1.2.3. These modules are shown in the accompanying flowcharts but not formally defined here.

APPENDIX D. BASIC NODE FUNCTIONS

BASIC NODE FUNCTIONS

1. BASIC DATA OPERATIONS

Data services will be provided for the creation, maintenance, and retrieval of nodal information based on the implementation of the new node structure. These services are intended to provide all information services supplied by the current STIS.

A set of functions are specified which operate on a node residing either in the Semantic or Entity Net areas of the STIS Concept Net. The functions can be grouped into the following broad categories:

(a)	Create	(new nodes assigned and created)
(b)	Maintenance	(existing nodes expanded, changed, deleted, or displayed)
(c)	Retrieve	(existing nodes, and/or data located or accessed)

These functions are defined at the node processor level and are accessible by the STIS intermediate language processor and possibly by user written applications programs.

The special symbols used in delimiting parameters in the following functional illustrations are defined as follows:

(1)	< arg >	implies some value of "arg" must be coded where "arg" can take one of several values (i.e., < arg > ::v_1| v_2|...|vn)
	[arg]	implies this parameter is optional

1.1 CREATE FUNCTIONS

The "Create" category of functions includes the following:

- Create Entity Node (<AOR>) = Node ID

- Create Attribute Node (<AOR>) = Node ID
- Create Value Node (<AOR>) = Node ID
- Create Term Node (<AOR>) = Node ID
- Create Word Node (<AOR>) = Node ID
- Create Node (<node type>, <AOR>) = ID

```
     where node type :: entity|attribute|value|term|word
          AOR = Area of Responsibility identifier
          Node ID = The integer number assigned to
                    this node (and henceforth by
                    which it will be accessed) by
                    the storage allocator.
```

These functions will add a node of the specified type to the concept net. The space is allocated in the node data base and the identifier returned to the caller. The node is initially created in the user's work space. (Subsequent windup operations for a fact store will generally cause the node to actually be written to mass storage.)

- Create Subnode (<Node ID>) = Sub-Node ID

The parent node is brought into the user's workspace if it does not already exist there. The next available sub-node number is fetched from the parent node and the sub-node ID is formed and returned to the caller.

1.2 MAINTENANCE FUNCTIONS

The "Maintenance" category includes the largest number of functions. It includes the following:

- Store Fact (value, attr ID, < Fact Control >,
  [< mode >, < Fact Seq # >],
  [< Footnote type >, text)

```
     where value = actual numeric, string, list, etc.
                   value of the attribute

          attr ID = node ID of the attribute

          Fact Control = one or more of the following
                         qualifiers: Area of Responsi-
                         bility, Classification, sensor,
                         credibility, etc.
```

mode :: Insert|Modify

Fact Seq# = Fact sequence number in a multi-fact situation

Footnote Type :: comment|warning|text

Text = alphanumeric textual data of the footnote

This function stores the actual value against the specified attribute for the "current" node previously located or created by the user. The fact control information and optional footnote text is also stored with the value. If the attribute does not exist in the current node, then the attribute/value pair is stored. If the attribute pre-exists and a value with the exact Fact Control parameters is found, then the first old value is overwritten by the new value if the "mode" and Fact Sequence # fields were omitted. The "mode" and Fact Sequence # fields determine the action to be taken in the case of pre-existing facts. Facts are implicitly numbered 1 through n. The "mode" argument allows the insertion of new facts and the modification of old facts. Insertions occur after the fact number specified in the Fact Sequence number field. If it is necessary to insert a new value before the first pre-existing value, then the Fact Sequence # field = 0. An old value may be modified by specifying its Fact Sequence number. (Note a "modify" of Fact Sequence #0 is not permitted.)

- Store Footnote (attr ID, <Fact Control>, [<mode>, <Fact Seq. #>, <Foot Seq. #>], <footnote type>, text)

  where attr ID = node ID of the attribute

  Fact Control = one or more of the following qualifiers: Classification, sensor, credibility, etc.

  mode :: Insert|Modify

  Fact Seq # = Fact Sequence Number

  Foot Seq # = Footnote Sequence Number

  footnote type :: comment|warning|text

  text = alphanumeric textual data of the footnote

The function will store a footnote for a value qualified by the given Fact Control under the given attribute. A search is made for the given attribute under the current node. If it is not found, an error status is returned. If the attribute is found, a search is made for the correct Fact Control qualified value. If not found, an error status is returned.

If found, the footnote is stored. The text may replace a previous footnote of the same type in which case the "mode" and Footnote Sequence # fields are examined. The Footnote Sequence # field [illegible] which specific footnote is to be modified or after [illegible] the new footnote will follow. Multi-fact situations are processed using the Fact Seq. # field as described in Store Fact.

- Delete value (attr ID, < Fact Control >, [<Fact Sequence # >])

  where attr ID = node ID of the attribute

  Fact Control = one or more of the following qualifiers: Area of Responsibility, classification, sensor, credibility, etc.

  Fact Sequence # = Fact Sequence Number

The function deletes the specific Fact Controlled value (including attached text) for the given attribute for the "current" node previously specified by the user. Other existing values are not affected. The Fact Sequence Number specifies the unique fact in the multi-fact case. If the specified value cannot be found, an error status is returned.

- Delete Attribute (attr ID)

  where attr ID = node ID of the attribute

This function will delete the specified attribute under the current node. All values and associated text for this attribute are likewise deleted. If the specified attribute does not exist within the current node, an error status is returned.

- Delete Footnote (attr ID, <Fact Control>, [<Fact Seq. #>, <Foot Seq #>], <footnote type>)

  where attr ID = node ID of the attribute

  Fact Control = one or more of the following qualifiers: Area of Responsibility, classification, sensor, credibility, etc.

  Fact Seq # = Fact Sequence Number

  Foot Seq # = Footnote Sequence Number

  footnote type :: comment|warning|text

The function deletes the text of the specified footnote for the given Fact Controlled value. Other text is not affected as are other Fact Controlled values not affected. An error status is returned on any 'no find' condition. The Fact Sequence Number specifies the unique fact in the multi-fact case. The Footnote Sequence Number specifies the unique footnote in the multi-footnote case.

- Display Node (node ID)

  This function lists the configuration of the node in the Concept Net specified by the node ID.

- Display Attributes (node ID)

  This function lists all the attributes stored in the specified node.

1.3 RETRIEVE FUNCTIONS

The "Retrieve" category of functions includes the following:

- Retrieve Node (node ID)

  The function locates the node specified by node ID in the concept net and logically loads it into the user's workspace. It becomes the "current" node.

- Retrieve Fact (attr ID, < Fact Control >, [< Fact Seq # >] = value(s)

  where attr ID = node ID of the attribute

  Fact Control = one or more of the following qualifiers: Area of Responsibility, classification, sensor, credibility, etc.

  Fact Seq # = Fact Sequence Number

This function returns a value or value list to the caller for the attribute specified for the "current" node. If the attribute is not found or the proper Fact Control qualifiers cannot be found, an error status is returned. The Fact Sequence Number field specifies the unique fact in the multi-fact case. If a warning is attached to the fact, this indication is also returned with the value(s). The value may be a single data item or an array of data items. The value(s) may also have been generated by an external program called as a part of the internal node processing function.

- **Retrieve Footnote** (attr ID, <Fact Control>,
  [<Fact Seq. #>, <Foot Seq #>],
  <footnote type>) = text

  where attr ID = node ID of the attribute

  Fact Control = one or more of the following qualifiers. Area of Responsibility, classification, sensor, credibility, etc.

  Fact Seq # = Fact Sequence Number

  Foot Seq # = Footnote Sequence Number

  footnote type :: comment|warning|text

The function returns the alphanumeric text of the footnote type specified to the caller. The attribute is sought, then, the proper Fact Controlled value, then the specified footnote. If any search condition results in a 'no find', an error status is returned. The Fact Sequence Number specifies the unique fact in the multi-fact case. The Footnote Sequence Number specifies the unique footnote in the multi-footnote case.