TECHNICAL REPORT SERIES

No. 227 January, 1987

Database techniques for text modelling : the document architecture of British statutes.

M.A. Heather and B.N. Rossiter

Abstract

Text like most real world data of any complexity and volume requires the use of database technology and for economy of implementation the application of generalised Database Management Systems (DBMS). Of the three primary DBMS models, the network, hierarchical and relational, the hierarchical has so far been the most extensively used for text but the relational which has many general advantages shows great promise for textual applications but still lacks proper software. Often text is treated as free text, i.e. independent of its content, but it is necessary to classify text further into data types such as loose-text, short-text, speech-text and whole-text as categories to be recognised on the road from "machine-readable" form to "machine-understandable" form.

A fully worked example of the various levels of modelling needed and their implementations are given for British statutes using the techniques provided by the processors of the generalized database management system SPIRES.

Series Editor: M.J. Elphick

Bibliographical details

HEATHER, Michael A.

Database techniques for text modelling: the document architecture of British statutes. [By] M.A. Heather and B.N. Rossiter.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1987.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 227)

Added entries
ROSSITER, Brian Nicholas.
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 227.

Abstract

Text like most real world data of any complexity and volume requires the use of database technology and for economy of implementation the application of generalised Database Management Systems (DBMS). Of the three primary DBMS models, the network, hierarchical and relational, the hierarchical has so far been the most extensively used for text but the relational which has many general advantages shows great promise for textual applications but still lacks proper software. Often text is treated as free text, i.e. independent of its content, but it is necessary to classify text further into data types such as loose-text, short-text, speech-text and whole text as categories to be recognised on the road from "machine-readable" form to "machine-understandable" form.

A fully worked example of the various levels of modelling needed and their implementations are given for British statutes using the techniques provided by the processors of the generalized database mamagement system SPIRES.

About the author

Michael Heather is senior lecturer in law at Newcastle Polytechnic where he has been responsible for computers and law since 1979.

Nick Rossiter is programming adviser in the Computing Laboratory with particular interests in databases and information retrieval.

Suggested keywords
DATABASE SYSTEMS
ELECTRONIC PUBLISHING
INFORMATION RETRIEVAL
RELATIONAL SYSTEMS
SPIRES
TEXT DATA MANAGEMENT

Suggested classmarks (primary classmark underlined)

Dewey (18th):	348.022	001.6442
U.D.C.	340.13	681.322.06

CONTENTS

1 INTRODUCTION

1.1 THE IMPORTANCE OF TEXT

A prime concern for software in advanced information processing is how to handle text. For natural language text is at the core of human communications and information storage. Words form the elementary particles of human understanding. Words have to be handled in a meaningful way by machines in any automatic process intended to interact efficiently with humans. Text is the natural form of expression for knowledge: data held at the level of knowledge in other forms still normally require words for interpretation [Heather & Rossiter 1986] and often for organisation. Thus computer programs require extensive structured comment statements, graphical and tabular representations need captions etc., digital versions of audio or visual materials such as speech, music, pictures, designs, etc usually need some language textual characterisation for storage and retrieval. Even the symbols in mathematics need to be defined in natural language at some stage. Currently the programming languages used in knowledge engineering (such as those of the LISP and PROLOG families) are based on atoms of text as primitives.

1.2 TEXTUAL INFORMATION

Human knowledge is stored and communicated in documents mostly in the form of full text natural language. However textual information systems are difficult to categorize. Classifications made to date tend to relate to a particular task in hand. Thus Sparck Jones and Kay [1973] distinguish fact retrieval from information retrieval; Radecki [1983] emphasizes the distinction between discrete and continuous retrieval systems in developing a fuzzy-set theoretic approach to the subject; Macleod and Crawford [1983] stress the organizational differences between database management systems, document retrieval systems and electronic filing systems.

It is true that text is only one form of knowledge representation, and cannot be isolated from other work on knowledge-based systems. Much of the work around the world today is concentrated on the development of formal methods to represent knowledge. Current work on knowledge formalism can be found elsewhere [IKBS Group 1984]. However most of the world's knowledge is stored in documents in full text natural language form. Any practical knowledge-based or expert system involving any quantity of real world data needs to be able to interact by automatic methods with documents already in existence. This report is concerned with modelling the somewhat amorphous structures to be found in full text for automatic machine handling of such documents.

2 CLASSIFICATION OF TEXT

2.1 TEXTUAL DATA TYPES

As a first attempt to deal with this amorphous nature of text, it is possible to make a classification into various textual data types. There are the two main families of formal language and natural language. These may be further subdivided as in figure 1. While formal non-numeric language is of recent origin, the sub-categories of formal language in figure 1 are well developed in modern computer science. Natural language on the other hand is of much more ancient origin but is still far from being understood by the machine. The sub-classification of natural language in figure 1 into data types is an obvious step towards machine comprehension. For there are differences in the level of knowledge represented by different types of text such as 'loose-text', 'short-text' and 'whole-text'.

Without direct human involvement, computers can only deal with the deeper contents of a discourse through the surface structures to be found in the words of the text [Rossiter & Heather 1986]. In database technology the employment of data-types is normally an operation at the semantic level of the information residing in the data in contrast to data-types in programming which are mainly at the syntactical level and initially were more for the benefit of the compiler/interpreter. Databases can exploit the nature of textual data-types to capture more of the information contained in the discourse. Data-types are the elements of a data model. The data model provides an extra layer of information about the data. With text data it is highly desirable therefore to make explicit in the data model the types of text used. It may be convenient to distinguish the classes of text before describing techniques to model them.

2.2 TYPES OF FREE AND FORCE TEXT

The terminology has not been standardized and 'full text' and 'free text' are used by some writers interchangeably. 'Free text' has no obvious or settled meaning, it may be used to describe the situation where a writer has no other constraints than those of natural language. Other than in this generic sense, it seems better not to encourage the use of 'free text' except perhaps when it is necessary to distinguish it from 'force-text'.

Force-text results from the use of language that is controlled in some manner. This control may be logical as in controlled vocabulary [1]. Or the control may be physical as in fixed length fields.

----

[1] The thesaurus and different types of assigned keywords are examples of the well established use of controlled text in computer-based learning packages and in information retrieval. Important distinctions between assigned controlled vocabulary and assigned natural language terms have long been recognized in information retrieval. For instance, see

```
                          TEXT
                           |
                           |
              |----------------------------------|
              |                                  |
      FORMAL LANGUAGE                     NATURAL LANGUAGE
              |                                  |
              |                                  |
      |------------|                  |------------------|
      |            |                  |                  |
   CODES      LANGUAGE            FORCE-TEXT        FREE-TEXT
                   |                  |               |
                   |                  |               |
   |--------|---------|         |------|          |--------|
   |        |         |         |      |          |        |
SYMBOLIC  MACHINE   HIGH  CONTROLLED CONTROLLED  LOOSE  FULL-TEXT
                    LEVEL  PHYSICAL   LOGICAL    -TEXT     |
                                                           |
                                   |-------------|---------------|
                                   |             |               |
                              SHORT-TEXT    SPEECH-TEXT    WHOLE-TEXT
```

Figure 1. A Classification of Textual Data-types

As machine data processing increases, the proforma (such as application forms for employment, insurance, etc.) are making more use of text which is controlled both physically and logically. Other than in the very special case of poetry, such combined physical and logical control of words was previously only limited to oddities like crossword puzzles. Now the demand for text characters to be entered on coding forms is widespread in business and commerce.

The constraints to be found in force-text can be utilised in a textual database both to describe the content of the words and also for the technical construction of the database. More detail will be given in the implementation for statutes in the second part of this report but a simple example is in the treatment of dates. A CONTROLLED PHYSICAL date might be 23/4/80 which illustrates the point that CONTROLLED PHYSICAL FORCE-TEXT is normally in canonical form. For this example there are of course two possible canonical forms, the English and the American. This degeneracy has to be handled by the database system. It may be possible to resolve any ambiguity

----
[1](cont'd)[Stevens 1965].

automatically but always at least one further constraint has to be known e.g. that the format is consistent. As a rule of thumb, CONTROLLED PHYSICAL FORCE-TEXT can be found by an exact match with a one pass (mixed-case) string scan. The first move towards CONTROLLED LOGICAL is when ordering is relaxed. So if both the forms 23rd April 1980 and April 23rd 1980 are permitted, it is probably still better to treat these as CONTROLLED PHYSICAL for either form could be found on a one pass substring scan with local contexting. CONTROLLED LOGICAL FORCE-TEXT on the other hand is more a concentration on 'what is said' rather than 'how it is said'. Examples of CONTROLLED LOGICAL representation of dates are 'Easter Monday', 'last Thursday', 'the Queen's sixtieth birthday[1]', or for a range of dates like 'during the reign of Henry VIII'.

Loose-text consists of fragments of full text often to be found as extra words of explanation in a database that is not primarily in full text form. Thus, a personnel or bibliographic database in addition to the text fields of NAME, ADDRESS, AUTHOR, TITLE, etc. may have notes in phrases or clauses which are not in full grammatical sentences. Explanations ancillary to graphs, diagrams and figures often consist of loose-text.

In figure 1, free-text is divided into loose-text and full-text. Full text is a generic term for textual expression at the ordinary knowledge level of human discourse. It comprises the categories short-text, speech-text and whole-text. This is not an exhaustive list and it will be seen later that these categories themselves can be further subdivided.

Short-text consists of an abbreviated form of full text. It is normally achieved by relaxing the strict rules of natural language. The form of communication is not expressly complete. It may take the form merely of suppressing some of the redundancy of language normally provided as a check. Short-text usually relies on knowledge common to the writer and the reader. Syntactical shortening is possible because knowledge of normal forms of expression are common to the average reader of a given language. Semantic curtailment is more complicated however as it assumes a common knowledge of the subject matter. In terms of generative grammars, the semantic short-text lies in the very context sensitive categories. The theoretical foundations of these are not yet very well explored. This makes the processing of short-text at the semantic level more or less impossible at the present time.

In passing it might be noted that the syntactical short-text relies on a common knowledge between the parties of the communication at the semantic level while semantic short-text implies pragmatic

----

[1] 'The Queen's birthday' is an interesting example of the way natural language can carry together both a constant and a variable to be distingushed as necessary by context whether 'official' or 'personal' and if personal for which sovereign.

co-knowledge. This context-sharing between the communicator and the receiver is beyond the classical information theory of Shannon [1949] which relies on long term ergodic processes to provide the overall independence needed for the application of statistical methods. Short-text needs some meta-signal theory such as proposed by Heine [1984]. That the context of data is part of the data is now recognized as an important principle in software engineering [Hutchinson 1986]. Database techniques can help to provide some "machine awareness" of context. Discussion of this in relation to the present work is described elsewhere [Rossiter 1986b].

However these distinctions in the information-density spectrum of the various types of text are not merely of theoretical interest for they have some quite important consequences in practice. Thus in the UK the use of the teletex system PRESTEL has been smaller than was anticipated for general information but it has become very popular for specialized users. The reason for this divergence may well lie in the characteristics of short-text. Public teletex and videotex have been able to step into a ready made telematic network market place by using domestic television sets as terminal devices but at the price of accepting the constraints. Communication from the user may even be limited to simple numeric keypads and communication to the user is normally restricted by the band widths and the physical format of the domestic television receiver 'window'. With the majority of sets currently available there is a maximum of about 150 words that can be displayed on one electronic page. The information providers on these systems have therefore tended to resort to short-text. For the specialised user a considerable quantity of information can be conveyed in this way particularly where the information is of a differential nature relating to changes, up-dates etc. For the general user however the information in short-text form may be quite unintelligible.

Whole-text on the other hand is the ordinary full and free expression of language in written form and conforms to the accepted rules of such communication. The important difference between whole-texts and loose or short-text is at the semantic and pragmatic levels. Loose-text is used for isolated scraps of information, short-text either communicates information at a shallow level or acts as a mere pointer to a deeper level but whole-text tells the whole story. This wholesomeness comes not from completeness, for it may not be complete but from its precision and level of reliability. Whole-text is information at the level of knowledge. It is a matter of the level of specificity.

It is important to note that it is not just that short-text is short and whole-text is long. The difference relates more to the nature of the contents of a document. An abridgement or abstract may be in either short or whole-text form. Brevity and conciseness are not necessarily synonomous. Often a short-text form may be used because there is neither the time nor intellectual energy available to produce an equivalent succinct whole-text. On the other hand, speech-text is usually at the level of specificity of whole-text but short-text in form.

The power of human communication increases from left to right in figure 1 across the different types of text while the power of machine communication is in the reverse direction. Thus the present state of human-machine communication has reached about half-way on this spectrum with high-level languages. The aim now is to bring the machine across the divide into natural language comprehension. One of the first steps is to handle these different classes of text. Here we are concerned only with written categories. These need to be integrated with other types such as speech-text and graphics. Indeed other work carried out so far suggests that speech-text is perhaps as broad a category as the more familiar 'high-level' group[1].

2.3 FULL TEXT PROCESSING

There are four principal phases to be considered in the handling of full text in an electronic medium. There is a transitional phase in converting the information from hard-copy documents into machine-readable form. Eventually, most documents may be written specifically for the electronic medium, and this will reduce to some extent logical problems on input. The second phase concerns the structuring of the textual data for systematic storage. The third phase relates to the interrogation and retrieval of information. The last phase involves the composition of the text in its final output form. The state of the art in electronic publishing is probably well represented by TEX [Knuth 1979]. TEX can achieve sophisticated textual structures through defining elementary boxes such as single characters or rectangles which can be amalgamated into lists of boxes representing larger units such as a single page. TEX is orientated towards physical representation of text. Markup languages such as SGML (Standard Generalized Markup Language) provide device independence within the physical constraints of traditional document layout [Smith 1986].

SGML follows closely the algorithmic model and notation of TEXTFORM [TEXTFORM User Guide 1981], a powerful text processing package which was devised in the late seventies at the University of Alberta and which has been supported for a number of years as the main text processor at MTS sites including NUMAC. TEXTFORM has been utilized in the practical electronic publishing of this report although certain of the features in this report could not be achieved by automatic methods using TEXTFORM alone.

"Desk-top" electronic publishing should mean that quite ordinary documents can now be produced in a high quality form previously only to be expected from a good professional publisher. This same quality

----

[1] Details have been omitted from figure 1 for these large family groups and other specialised text views such as plain-text, clear-text, cipher-text, etc to be found in cryptography. See [Kahn 1963].

is available from the very first draft. Furthermore, effects for different classes of readers can be provided at little extra cost. For example the machine can provide double referencing. The footnote reference on the page can for the sequential reader be achieved without the skill of a human typesetter; while at the same time without the employment of a professional indexer, a full list of references sorted in some lexical order can for the casual reader be separately appended with pointers to the citation in the body of the text. Only the latter is provided in this report and we had to resort to semi-automatic methods to accomplish this. A database approach as advocated in this report could achieve electronic publishing by fully automated means. Also by having in the machine some awareness of the document architecture it is easier to apply meta-rules to deal with problems that TEXTFORM finds difficult to resolve. For instance there is the occasional conflict between where at the end of a page to break the text and where to break the footnote. An example of this can be found at the break of the second page of this report. The TEXTFORM solution to be found at the foot of page 3 would clearly not have been adopted by the human typesetter. It seems that any mark-up language will need database features if it is to avoid the general limitations of an algorithmic model.

In contrast to text processing, present methods in Information Retrieval are mainly limited to considering documents as a collection of unstructured free text. Here we are concerned with the modelling that is required to handle the natural structure that is found in text. While it is necessary to draw upon the methods of Information Retrieval the major emphasis is directed towards the techniques of modern database technology. Electronic Publishing today is concerned with integrating text with other types of data such as numeric, graphic, etc. Such integration requires a structured approach to be applied to the text. The nature of electronic text and methods of handling it will be considered in the next section.

3 ELECTRONIC TEXT

3.1 APPLICATIONS

Initially, computers were used solely for numeric applications by scientists and engineers. Thus, the early versions of languages such as FORTRAN were not designed to handle character strings at all. However, because linguistic forms of expression come more naturally to the human mind than numeric ones, it became necessary for the computer to deal with comment statements, captions, etc., and other forms of loose-text. In business applications, text is more important and COBOL was developed with text fields for use by application programmers. As applications developed for handling natural languages for purposes such as computational linguistics and automatic translation, there was a need for the more powerful tools as in SNOBOL. All mainstream computer languages now incorporate basic string handling functions. Even comment statements are now recognized as an essential part of good structured software engineering. However, these are examples of loose-text, or at the most short-text. The limited extent to which these forms could represent human thoughts was acceptable because of the limitations of hardware. With the advent of more powerful machines and larger, cheaper storage devices, it has become possible to handle whole-text by automatic means. There is then an immediate I/O problem because of the bulk and the complex structure of whole-text, so that it cannot be held easily in fixed fields or as simple program data. A high level language can only cope with a small amount of text held in main memory. Even on a mainframe more than 0.5 Mbyte at a time soon becomes excessive.

Some versions of BASIC have quite good interfaces for access to permanent data files from within programs but the better programming languages have tended to neglect efficient management of permanent data. More recent extensions proposed for Pascal such as Pascal/R [Schmidt 1977] and RAPP [Connolly 1986] have permanent relational data types indicating a move towards the use of database methods.

Flat file systems alone can handle large amounts of permanent data if the structure is simple. Thus a telephone directory for the 500 million telephone subscribers in the world containing loose-text in fields of name, number, address, etc is well within present technology. On the other hand whole-text data of a much smaller quantity cannot be addressed in data fields in the same way and present commercial full text databases are held and handled as free text with little regard to any structure.

Therefore this combination of complexity and size require database techniques to model the structure and to provide reasonable performance for realistic quantities of text in applications of electronic publishing and in the electronic office. The approach adopted here therefore is from the stand point of data base technology. The emphasis is placed on the means of storage and the two other phases of input and output developed within that framework. The fundamental features of database technology can be applied to text as data. It is necessary therefore to examine the possible data models (including those that may not yet be generally implemented) to assess their applicability for text.

3.2 DATA MODELS

The simplest computer model for textual data is the free text model which penetrates no deeper than the level of 'FREE-TEXT' in figure 1. The free text model treats a collection of text as a continuous byte-stream without internal structure. Searching can be by sequential access or through indexes. Sequential methods are slow. Tests carried out on the Amdahl 5860 (a 12.5 MIPS machine) at NUMAC using the MTS context editor show that sequential searching of text can be carried out at the rate of 4Mb per CPU second. Thus the search of a large text file of 100Mb would require at least 45 minutes elapsed time with 100 users connected to such a multiple-access system.

Searching by means of indexes is much faster but the use of calculated indexes (by hashing for example) requires a clear prespecification of key values that is not always appropriate for text. Physical indexes are expensive to build. An alternative approach to word indexing in the conventional manner by inverted files is to use associative disks or database machines. Parallel methods now give faster access without the need to go through indexes but require the purchase of specialised hardware [Agosti 1983-4] which can handle typically up to 50Mb. Thus, one solution is to apply the ICL CAFS (Content-addressable File Storage) associative disk [Haworth 1985] together with suitable software such as IDMS/R, a network system with a relational interface. Recent developments in CAFS have increased the storage capacity but secondary sparse indexes may still be needed as well to achieve reasonable performance for files larger than 20Mb [Report on CAFS Usage at Oxford 1985; ICL Computer Users Association 1985]. Even 10Mb requires a 10 second response time and text files of that size without indexes are only recommended when the enquiry load is light [Carmichael 1984]. A better response time is now available with the faster disks of the ICL Series 39 range [Carmichael 1986]. However, CAFS is not available to non-ICL installations and other manufacturers such as IBM do not seem to be following this path. Date [1983] points out that for relational systems, associative disks may well be useful for 'library search' problems, that is complex intra-record queries of the type that are needed for text, but there is only a marginal gain from associative disks in other types of query. Date also states that the hope in database machines is that the gains from parallel execution will outweigh the losses from host/backend communication and warns:

> "For offloading to be cost-effective, the amount of work offloaded should be an order of magnitude greater than the amount of work involved in doing the offloading."

This may only be realisable for some types of query. On the other hand, the Japanese Fifth Generation Project does involve the proposed construction of a relational database machine [Moto-oka 1982].

3.3 SPECIALISED TEXT SYSTEMS

In the 1960's retrieval systems for text began to appear [Teskey 1982]. These typically were a suite of programs providing facilities to build indexes using inverted files. These indexes enabled retrieval using Boolean operators on all words and substrings in a text (excluding common words) or by means of a thesaurus. These systems offered various display features such as keyword in context (KWIC). Examples of these systems are STAIRS (IBM), STATUS (Harwell), SPIRIT (France), MISTRAL (Honeywell-Bull), GOLEM (Siemens), and QUOBIRD (Belfast). In the early 1970's more specialised text retrieval systems became available such as DIALOG (Lockheed's Bibliographic database), MEDLARS (medical literature retrieval system of the US National Library of Medicine), Medusa (Newcastle University retrieval system on MEDLARS data), and LEXIS (full text law retrieval). These systems began to incorporate some modern database management system techniques such as data fields and structures, and data directories and dictionaries.

3.4 GENERALIZED DBMS FOR TEXT

The prime objective of a database management system (DBMS) is to make application programs independent of the physical structure of the data. To achieve this objective, the ANSI/SPARC architecture may be used [ANSI/X3/SPARC Report 1978]. A conceptual schema or model is defined as a global logical definition of the data structure. This schema relates to the internal (physical) definition by a mapping from the logical level to the physical level. The schema is protected from changes at the physical level by adjusting this mapping. Each user has his own view (external schema) of the database which may be restricted. This is achieved by a series of mappings which provide security, logical data independence, etc. [Rossiter 1984]. The schemas are often classified into three main types: hierarchical, network, and relational [Date 1981].

Hierarchical systems became popular as soon as magnetic tapes came into use as data could be easily organised physically in the form of nested trees. With the advent of magnetic disks, more sophisticated physical structures based on hierarchical data structuring have become possible and the ability to design complex systems has increased (e.g. IMS). The specialised text systems referred to above employ a hierarchical form of structure with relatively easy implementation. Hierarchical systems are inadequate for complex commercial data processing because they only allow the modelling of one-to-many relationships.

The network model is more general than the hierarchical model. It allows the modelling of many-to-many relationships. The CODASYL model is a well known example which enables many-to-many relationships to be achieved in a number of ways. Many implementations of it exist (e.g. IDMS), and it has proved applicable to many administrative and commercial areas. To implement, however, a successful application of the network model needs the involvement of professional programming staff even for record retrieval because of the intimate knowledge that is required of the physical structure. In particular network query languages are not suited to iterative searching and this may be the chief reason why the network model has not been used extensively for text. This is not to say that a network text programming initiative is impossible but it would be a fairly major undertaking. Any user interface would always be of very complicated design and there would be the usual network constraint of having to define statically all the relationships. This same conclusion with regard to the network model seems to have been reached at other text sites. For example it is reported by the Oxford University Computing Service:

> "AT OUCS, however, the only true database management system available is IDMS. Constraining textual data to fit this, although feasible, is not an appealing prospect." [Burnard 1985].

The relational model [Codd 1970] on the other hand offers the power of the network model but with a simple and elegant method of data manipulation. In particular, relational systems offer advantages in the ease with which keys can be manipulated and in flexibility in both searching and in dynamically varying the unit of retrieval as users' needs change. However, problems arise with the relational systems that are in use today in handling the free text of the headings and content. A general survey of relational database systems was carried out in 1983 by Schmidt and Brodie [1983] to assess how closely the systems were adhering to the principles laid down by Codd and to compare their capabilities. While the use of text was not of primary interest in their report, it is possible by examining carefully the capabilities of the systems surveyed to extract information on their suitability for textual applications. The results are summarised in the table of figure 2 together with known advances made since the report of Schmidt and Brodie was published.

In relational systems, variable length fields imply variable length records and vice versa for first normal form requires one occurrence of each field per record. Variable length records are possible in other models even with only fixed length fields when it is possible to vary the number of the fixed fields. With text, the extreme variation of record size means that setting the maximum field length at the largest can result in very poor storage utilization. The bulk nature of text therefore means that this wastage can be quite critical and most of the system storage available is occupied with blank space. Systems such as INGRES and ORACLE have introduced facilities such as data compression to deal with the limitation of fixed length records. The compression is usually achieved by the use of a blocking code to represent unused space in the fixed length fields.

The first two columns of figure 2 indicate how well data of markedly varying length can be stored. The first column shows whether variable-length format is available with the standard character field-type which with system overheads can typically be up to 256 bytes in length. The second column shows the same for longer text values typically held in special field-types declared as CHAR(LONG) or MEMO. Column two shows that five systems NOMAD, ORACLE, REVELATION, SQL/DS and SYSTEM R would provide efficient storage for the average textual record size but DBASE3, INGRES and QBE for small records only. The others would require a fixed maximum length to be specified for each occurrence of data, the unused portion of which would be wasted. Column three shows the maximum

Product	Variable-length fields		Max Len Field (bytes)	Field Indexes	Word Indexes	Substring Search Facility	
	short strings	long strings				short strings	long strings
DBASE3	no	yes	4000	yes	no	yes	no
INFO	no	no		no	{ only with a non-relational extension }		
INGRES	no*	no*	2000	yes	no	yes	yes
NOMAD	yes			yes	no	yes	
ORACLE	no*	no*	64k	yes	no	yes	no
PRTV	no			yes	no	yes	
QBE	yes	yes	3200	yes	no	yes (also patterns)	yes
RAPPORT	no	no	1000 (at NUMAC)	yes	no	yes (with RASQL)	yes
REVELATION	yes	yes		no	no	yes	yes
SQL/DS	no	yes		yes	no	yes	no
SYSTEM R	no	yes	32k	yes	no	yes	no

* however, data compression available

Figure 2. Analysis of Relational Systems for Text

length permitted in the system for any character field.

The fourth column indicates the ease with which data can be located in large files, and nearly all systems provide an indexing facility to assist in this. However, as the fifth column shows, none allow index construction to be programmed by the database designer to the extent that indexes can be built on words contained within the text. Word indexes are essential for most text applications.

Relational systems are in fact not too suited to indexing in the traditional manner using physical addresses for complex data. There is no real concept of a 'global' index in relational databases. A single index cannot readily be constructed for the various domains in the different relations across the database. On the other hand, the relational approach offers the possibility of logical indexing with the facilities for searching at various levels dynamically. Stonebraker [1986] considers some of the extra facilities that are required by relational systems to be effective in document processing. These include a BREAK operator for creating a new relation to hold the words comprising a document and their position within it and a CONCAT operator to rebuild documents from word indexes. The relation holding the words would act as a logical index to the main text.

The last two columns indicate the ability to search for substrings contained within a textual string, and nearly all systems have this ability for short text fields of up to about 256 characters. However the long text fields of type LONG or MEMO can usually not be searched at all for substrings even in sequential mode. Thus some relational systems have provided variable-length long fields to store text but have not provided adequate word indexing and substring search facilities for these fields. Lorie [1981] of IBM Research San Jose summed up the situation thus:

> "Finally, the extensions of data types to support non-formatted data completes the list of enhancements that could convert a classical relational system supporting data processing transactions into a system supporting a much wider class of applications".

Lorie says of the IBM prototype System R:

> "the normal maximum size of a field is 254 bytes, but there is a special field type CHAR (LONG) VARYING which allows up to 32767 bytes, but which gives intolerable restrictions and performance penalties in its use".

The same could be said of DBASE3, ORACLE and SQL/DS and for the same reason. Except for the limit of 2000 bytes per text field, INGRES is probably the most promising candidate for providing adequate facilities for text. Not only does it offer the best facilities at present but RTI who provide the product are conscious of the additional facilities required to implement effective full-text relational databases.

The three models of network, hierarchical and relational may be considered as the basic models in use today for practical purposes. However, other models have been proposed usually in order to incorporate more semantic features than can be included in the basic models in an attempt to model more closely the real world. These include RM/T [Codd 1979], the Role Model [Bachman & Day 1978; Bachman 1980], the Basic Semantic Model [Schmid & Swanson 1975], the Chen Entity-Relationship Model [Chen 1976], and the Borkin Semantic Model [Borkin 1979].

General implementations of these models are not available in commercial packages but they are useful for providing a more complete

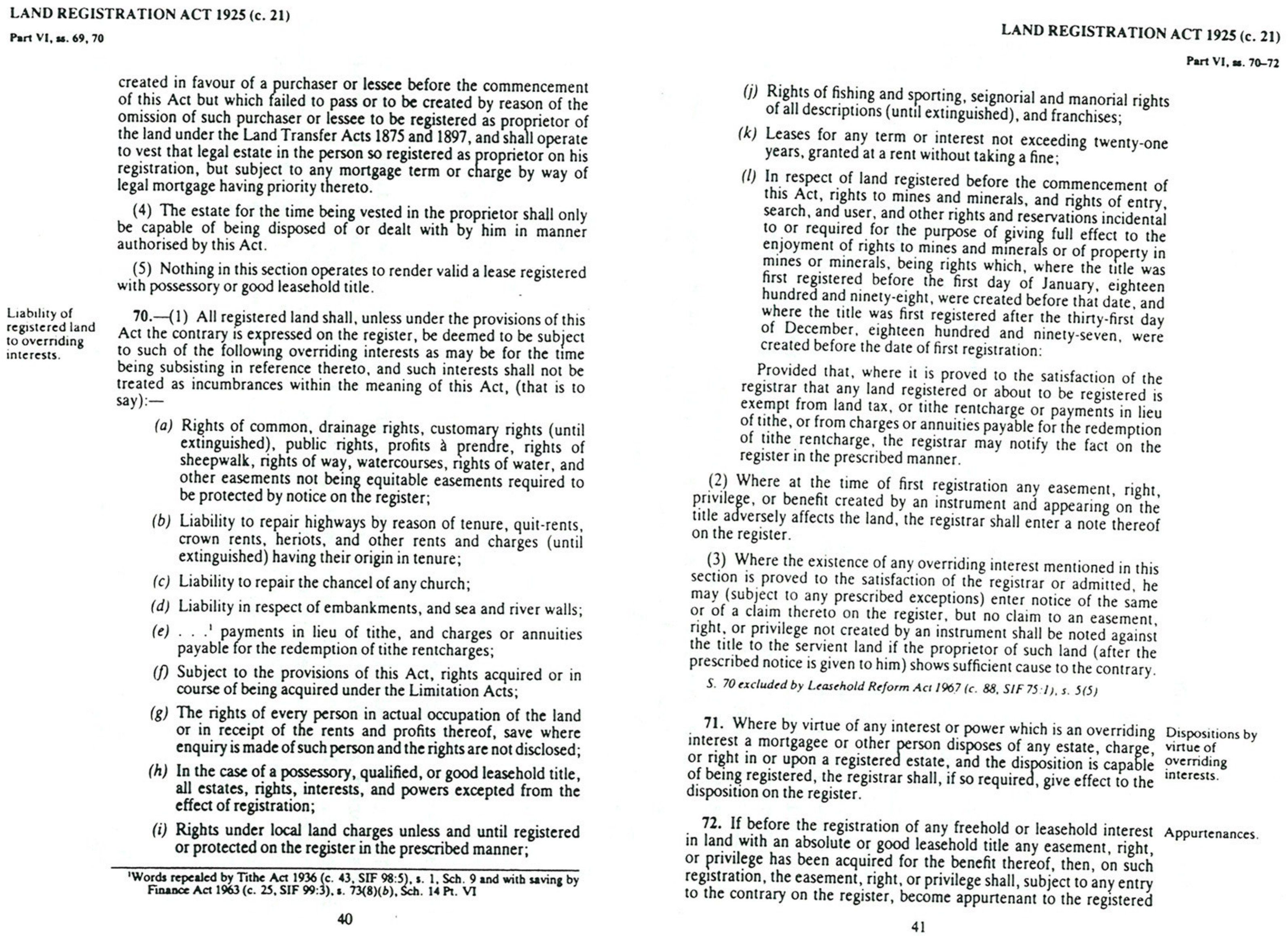


Figure 3. An extract from the printed version of UK Statutes reproduced with permission of the controller of HMSO

LAND REGISTRATION ACT 1925 (c. 21)

Part VI, ss. 69, 70

created in favour of a purchaser or lessee before the commencement of this Act but which failed to pass or to be created by reason of the omission of such purchaser or lessee to be registered as proprietor of the land under the Land Transfer Acts 1875 and 1897, and shall operate to vest that legal estate in the person so registered as proprietor on his registration, but subject to any mortgage term or charge by way of legal mortgage having priority thereto.

(4) The estate for the time being vested in the proprietor shall only be capable of being disposed of or dealt with by him in manner authorised by this Act.

(5) Nothing in this section operates to render valid a lease registered with possessory or good leasehold title.

Liability of registered land to overriding interests.

70.—(1) All registered land shall, unless under the provisions of this Act the contrary is expressed on the register, be deemed to be subject to such of the following overriding interests as may be for the time being subsisting in reference thereto, and such interests shall not be treated as incumbrances within the meaning of this Act, (that is to say):—

(a) Rights of common, drainage rights, customary rights (until extinguished), public rights, profits à prendre, rights of sheepwalk, rights of way, watercourses, rights of water, and other easements not being equitable easements required to be protected by notice on the register;

(b) Liability to repair highways by reason of tenure, quit-rents, crown rents, heriots, and other rents and charges (until extinguished) having their origin in tenure;

(c) Liability to repair the chancel of any church;

(d) Liability in respect of embankments, and sea and river walls;

(e) . . .[1] payments in lieu of tithe, and charges or annuities payable for the redemption of tithe rentcharges;

(f) Subject to the provisions of this Act, rights acquired or in course of being acquired under the Limitation Acts;

(g) The rights of every person in actual occupation of the land or in receipt of the rents and profits thereof, save where enquiry is made of such person and the rights are not disclosed;

(h) In the case of a possessory, qualified, or good leasehold title, all estates, rights, interests, and powers excepted from the effect of registration;

(i) Rights under local land charges unless and until registered or protected on the register in the prescribed manner;

[1]Words repealed by Tithe Act 1936 (c. 43, SIF 98:5), s. 1, Sch. 9 and with saving by Finance Act 1963 (c. 25, SIF 99:3), s. 73(8)(*b*), Sch. 14 Pt. VI

40

LAND REGISTRATION ACT 1925 (c. 21)

Part VI, ss. 70–72

(j) Rights of fishing and sporting, seignorial and manorial rights of all descriptions (until extinguished), and franchises;

(k) Leases for any term or interest not exceeding twenty-one years, granted at a rent without taking a fine;

(l) In respect of land registered before the commencement of this Act, rights to mines and minerals, and rights of entry, search, and user, and other rights and reservations incidental to or required for the purpose of giving full effect to the enjoyment of rights to mines and minerals or of property in mines or minerals, being rights which, where the title was first registered before the first day of January, eighteen hundred and ninety-eight, were created before that date, and where the title was first registered after the thirty-first day of December, eighteen hundred and ninety-seven, were created before the date of first registration:

Provided that, where it is proved to the satisfaction of the registrar that any land registered or about to be registered is exempt from land tax, or tithe rentcharge or payments in lieu of tithe, or from charges or annuities payable for the redemption of tithe rentcharge, the registrar may notify the fact on the register in the prescribed manner.

(2) Where at the time of first registration any easement, right, privilege, or benefit created by an instrument and appearing on the title adversely affects the land, the registrar shall enter a note thereof on the register.

(3) Where the existence of any overriding interest mentioned in this section is proved to the satisfaction of the registrar or admitted, he may (subject to any prescribed exceptions) enter notice of the same or of a claim thereto on the register, but no claim to an easement, right, or privilege not created by an instrument shall be noted against the title to the servient land if the proprietor of such land (after the prescribed notice is given to him) shows sufficient cause to the contrary.

S. 70 excluded by Leasehold Reform Act 1967 (c. 88, SIF 75:1), s. 5(5)

71. Where by virtue of any interest or power which is an overriding interest a mortgagee or other person disposes of any estate, charge, or right in or upon a registered estate, and the disposition is capable of being registered, the registrar shall, if so required, give effect to the disposition on the register.

Dispositions by virtue of overriding interests.

72. If before the registration of any freehold or leasehold interest in land with an absolute or good leasehold title any easement, right, or privilege has been acquired for the benefit thereof, then, on such registration, the easement, right, or privilege shall, subject to any entry to the contrary on the register, become appurtenant to the registered

Appurtenances.

41

specification of a subject than can be obtained using the three basic models. So if a more advanced model is employed it is still necessary to map it on to one of the basic models while aiming to retain as much of the semantic detail as possible. The effectiveness of the E-R and Semantic models has been investigated as part of this work and the models of Chen and Borkin[1] will be contrasted later in this report in the application of modelling UK statutes.

In attempting to implement the complex textual structure to be found in an act of parliament, it was found necessary at present to use a hierarchical package and this implementation will now be described in some detail as an experiment in textual data modelling. The standard DBMS employed at MTS sites[2] is the Stanford Public Information REtrieval System (SPIRES) which provides the word indexing which is not yet available in the relational implementations. Some explanation of the way that SPIRES handles text should first now be given.

3.5 SPIRES AND TEXT

SPIRES is a generalised DBMS employing basically the hierarchical model for the structuring of data, but with facilities also to construct network structures of some complexity and provide relational views. It was developed at Stanford University [Schnoeder, Kiefer, Guertin & Berman 1976] and is employed in the RLIN project for storing the bibliographic holdings of several major American universities. Although it is designed to be truly generalised, SPIRES was initially developed for textual applications and thus has most of the facilities that are required in this area. These include variable length records and fields, generous allowances for the maximum lengths of records and fields, string data-type, and versatile index construction which can be controlled or programmed to give indexes on individual words excluding if required certain common words. Searching facilities include matching on complete field values or on partial values. The physical organisation employed in data record structure and in the indexes is B-trees. A review of this access method, which was developed initially by Bayer, and its variants has been made by Comer [Comer 1979]. In SPIRES, long textual strings which would quickly fill the B-tree are removed to a residual data set, leaving just the key and a pointer to the residual set in the B-tree. The effectiveness of such an access method is illustrated by a quoted maximum of five disk accesses for location of a particular record in a file containing a million records [Spires File Definition Manual 1982]. Index records are held completely within a B-tree structure because of their short length.

----
[1] See figure 5 for the Chen E-R model and figure 6 for the Borkin Semantic Model.
[2] For an earlier description of SPIRES and its use at NUMAC, see [Rossiter 1980].

STATUTORY SUBSTRUCTURE	HEX CODE
Act Key No	#0F91
Title of Act	#02
List of footnotes,	#3F
Other headings	#03
Preamble	#F0
Arrangement of sections	#61
Arrangement of schedules	#F3
Centre headings	#2D+
Main text	#5A+
Schedule[s n]	#37
Centre headings	#2E
Explanation of omitted provisions	#F1

Figure 4. Paradigm of UK Statute Structure.

The SPIRES system comprises some six main separate language processors:

- the file definition language used to define the logical structure of a file, its external schemas, its physical design, and its physical indexes;
- the search and display language used to locate and print information, either in sequential mode or via direct access or inverted files;
- the updating language used to change the content of a database;
- the formats language used to read in data to a database which is not in the default SPIRES input format, or to write output in a user-defined output;
- the protocols language used to provide host language control of the session of a user;
- the partial processing language used to provide navigation through the hierarchical structures.

The use of certain facilities which rely on physical relationships to be found in some custom-built text packages are not encouraged in SPIRES because of the nature of its logico-semantic design. Thus distance operators between word positions are not permitted in the primary search language because they are physically rather than logically oriented, and as they can only be applied somewhat arbitrarily they offend against the principle of inclusive processing. Thus the way to search for records with words within a particular proximity of each other is to search for all combined occurences and then to apply the partial processing language under the control of the protocols language to the result stack. This provides a more general method of specifying constraints in a search. This may involve associations between complex entities as may be determined at run time and those dependent on context sensitive conditions, or involve simpler relationships between words such as those within the same sentence or same paragraph. If the cost of updating and storage overheads is critical, this approach is also much more efficient: word indexes then only need store the record address where a particular word is to be found.

The complexity of text is such that it may be necessary to use extensively any of these features and processors in an implementation on SPIRES. An act of parliament is an excellent example of a complicated structured document and a procedure to implement UK statutes on a computer will now be given in some detail as an illustration of text modelling in a SPIRES data base.

4 UK STATUTES

4.1 STATUTORY TEXT

Statutes promulgated by governments are an important primary legal source. In order to discover the law on a particular topic, the citizen or lawyer needs to be able to find the relevant part of the legal sources. An extract from the printed version of a UK statute is given in figure 3.

A person seeking the answer to some specific point of a legal problem cannot be satisfied with a bibliographic reference. If recourse to an act of parliament becomes necessary, it is in the whole-text that any solution is to be found. This whole-text is divided into numbered sections in the body of the act, or in numbered paragraphs in one or more schedule. These sections and paragraphs may be further divided without limit, and sub-sub-subparagraphs are not uncommon. On the other hand sections and paragraphs may be grouped into higher level parts. Some logical arrangement is usually chosen to determine the ordering and the division between the main body and the schedules. In fact other alternative orderings would often be possible because there are many intra-relationships within the subject matter [Heather 1982]. Thus for example a word or phrase in a section may be defined in some other section in another part of the act or even in another act altogether. Any interdependency may be explicit or by implication. In the extract in figure 3, the effect of section 70 is that the meaning of the word "incumbrance" found elsewhere in the act is qualified negatively by this lengthy description. In order to aid the reader in comprehending any portion of an act, the statutory draftsman provides a complex array of loose-text, short-text, and whole-text employed in the long title, the short title, the preamble, in cross headings, subheadings, cross notes, side notes, marginal notes, etc. [1]. The whole-text is

----

[1] This use of text to operate on other text is sometimes referred to as meta-text: see [Zunde 1984].

further qualified or refined in other ways, for instance, in the footnotes[1] or by reference to parallel statutes, or to relevant case law. It is extremely rare for cases to be refered to explicitly in statutes but these may be essential for a full understanding of the effect of any statutory provision. Thus in 1981 [William & Glyn's Bank Ltd v Boland 1981] there was a decision on the effect of section 70(1)(g) in the extract shown from this act of 1925 which has important consequences for occupiers of mortgaged property. This same sub-sub-section is still receiving judicial attention today [Winkworth v Edward Baron Development Co Ltd 1986]. The structured form and the requirement to consult a number of related legal texts makes the use of database technology highly attractive for processing statutes automatically.

4.2 STATUTORY PARADIGM

Parliament enacts about eighty UK statutes every year. There is considerable variation in size and content, but they all adhere to the same basic but complex form. A basic paradigm for an English statute is presented in figure 4. The common form of it has remained the same for several centuries although it has become more elaborate in recent years. The substructure is composed of a set of entity types arranged in a definite pattern for the purposes described above. The printed version uses a variety of type faces and sizes to convey to the reader in an effortless fashion the different effect of these various entities. Figure 4 lists the hexadecimal code corresponding to the type face used for each entity. The plus sign '+' indicates the head of a family of codes. Thus 'Main Text' commencing with code #5A includes about 30 additional codes[2] representing different formatting features.

The aim of this work has been to retain all the information on the printed page presented to the reader, visible in the example in figure 3. The assumption is that the use of different styles and size of type-fonts is necessary for full comprehension of the text. The detail of the formatting forms a study in itself and will not be dealt with in this report. The aim of retaining all the information is achieved by mapping it on to a database structure.

----

[1] in the extract shown footnote 6 provides information about the wording in section 70(1)(e).
[2] The hexadecimal codes in figure 4 map directly onto the standard typesetting styles used by HMSO [HMSO Technical Services 1981].

5 STATUTORY DATA MODELS

5.1 SEMANTIC MODELS

For a full specification of the structure of data, access to the semantic level is required. If the structure is simple, a small range of data types is usually sufficient. Text however because of its amorphous nature usually requires full semantic models to capture completely its structure. Examples are those of Chen and Borkin already mentioned above.

The viewpoint of Chen is that database design is concerned primarily with the occurrence of entities and the relationship between entities. Further analysis of the entities of a statute as given in figure 4 suggest that the following constraints need to be applied:

- An act must contain at least one section[1], and may contain one or more parts.
- If a part occurs in an act, it must contain at least one section.
- A section may contain one or more subsections, which are described by the same type of attributes as the section.
- An act may contain one or more schedules.
- A schedule must contain at least one paragraph, and may contain one or more subschedules.
- If a subschedule occurs in a schedule, it must contain at least one paragraph.
- A paragraph may contain one or more subparagraphs, which are described by the same type of attributes as the paragraph.

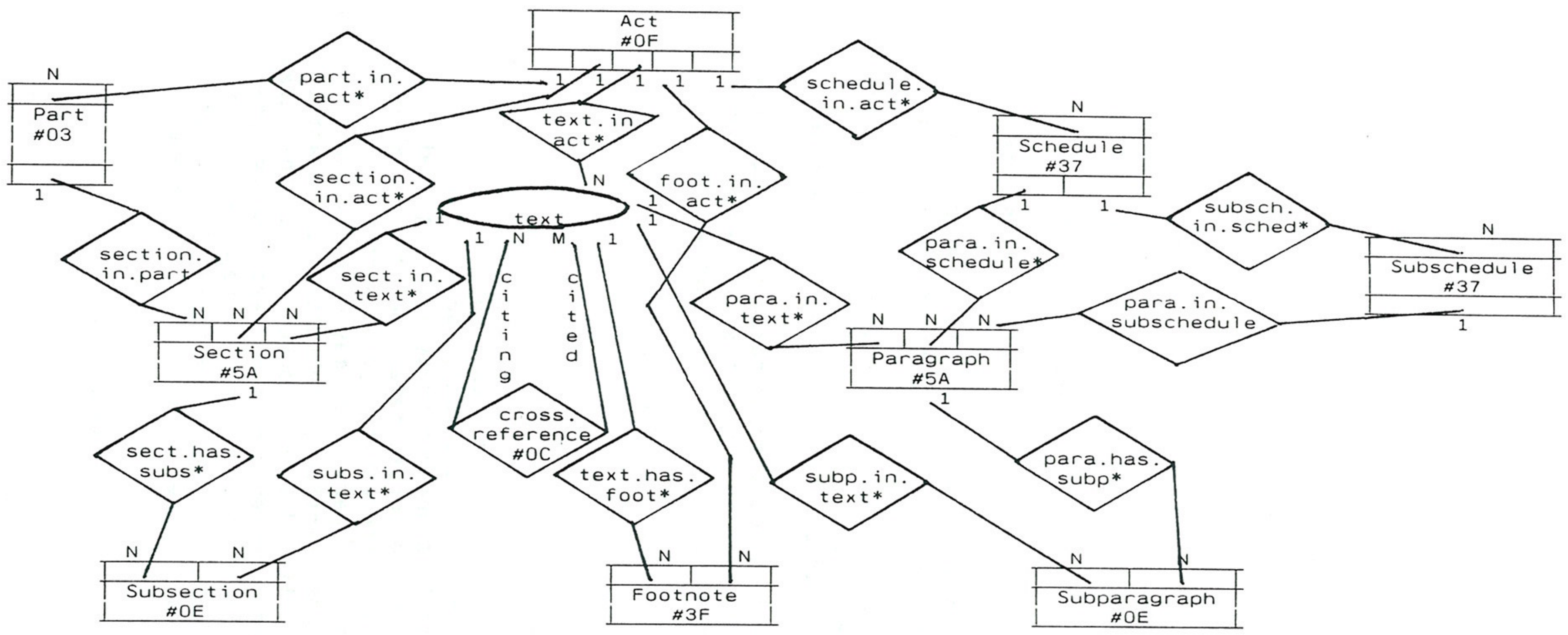
Therefore on applying the principles of Chen, an entity-relationship diagram of a UK statute would be represented in the form of figure 5(a). The particular representation here follows the style enhanced by Howe [1983], together with the further enhancements to handle generalisations advanced by Sakai [1983]. The hexadecimal identifier at each node in the tree structure corresponds to that of the relevant entity in figure 4.

The generic structure defined here is 'text' which is used to represent the specific entity-types 'section', 'subsection', 'paragraph' and 'subparagraph' as one entity-type if required. Its creation enables for instance the 'cross.reference' relationship where one of the occurrences of the entity-types 'section', 'subsection', 'paragraph' and 'subparagraph' may cite any other occurrence of the entity-types 'section', 'subsection', 'paragraph' and 'subparagraph' to be simplified into an occurrence of the 'text' entity-type may cite another occurrence of the 'text' entity-type. Cross-citations are of considerable importance in legal statutory text, about one in five of the 'text' entities cites another 'text' entity in our sample of the law. The average number of references made by each citing 'text' entity is almost two.

----

[1] or un-numbered equivalent main text in old statutes.

Figure 5. The Chen Entity-Relationship Model of Statutes:
(a) Diagrammatic Representation after Howe and Sakai



Notes: 1) Each rectangle names an entity-type, each diamond-shape names a relationship, and the oval names a generalisation.
2) For each one-to-many (1:N) association, i.e. all except 'cross.reference' which is many-to-many (N:M):
a) a dot within the lower segment of the parent entity (flagged by '1') indicates that the relationship is mandatory and that a child entity (flagged by 'N') must occur.
b) a dot outside the lower segment of the parent entity indicates that the child entity may not occur, so the relationship is optional.
c) a dot within the upper segment of the child entity indicates that if the child entity occurs, the relationship is mandatory.
d) a dot outside the upper segment of the child entity indicates that the relationship is optional, even if the child entity does occur.
3) The N:M association is recursive. The role played on each side of the relationship is shown by the annotations 'citing' and 'cited'. All N:M relationships need to be represented by table-types in the E-R model.
4) A relationship with an alphanumeric name but additionally flagged '*' is mandatory in that if the child entity occurs, it must be linked through the relationship to the parent entity.
5) A relationship not additionally flagged '*' is optional in that even if the child entity occurs, it may not be linked through the relationship to the parent entity.
6) Only optional 1:N relationships need to be represented by table-types in the entity-relationship model.
7) The value following # for each entity-type and the 'cross.reference' relationship is its hexadecimal code on the HMSO tape.

Figure 5. The Chen Entity-Relationship Model of Statutes:
(b) Partially-normalized Table-types

Act (year, chapter, title, date, preamble, arrangements,
crossnotes, + 14 text.formatting attributes)

Part (part.no, year, chapter, part.headings, part.subheadings,
crossnotes, footnotes.to.old.statutes, + 5
text.formatting.attributes)

Section (section.no, year, chapter)

Section.in.Part (section.no, part.no, year, chapter)

Subsection (subsection.no, section.no, year, chapter)

Schedule (schedule.no, year, chapter, schedule.headings,
crossnotes, omissions, footnotes.to.old.statutes, + 29
text.formatting attributes)

Subschedule (subschedule.no, schedule.no, year, chapter,
subschedule.headings, crossnotes, omissions,
footnotes.to.old.statutes, + 29 text.formatting attributes)

Paragraph (paragraph.no, schedule.no, year, chapter)

Para.in.subschedule (paragraph.no, subschedule.no, schedule.no, year,
chapter)

Subparagraph (subparagraph.no, paragraph.no, schedule.no,
year, chapter)

Footnote (footnote.no, year, chapter, footnote.text)

Text {section, subsection, paragraph, subparagraph}

Text (text.id, footnote.no, marginal.note.other, crossnotes, omissions,
footnotes.to.old.statutes, + 20 text.formatting attributes)

Cross.reference (citing.text.id, cited.text.id)

Notes: 1) The underlined attributes comprise the identifier for each
table type.
2) The brackets {} define the scope of a generic entity-type.

The complete E-R model requires more detail than the diagram of figure 5(a), in particular information on the attributes of each entity-type and which of these comprise the identifier. Figure 5(b) shows this information for the statute law but it should be noted that many of the tables are not fully normalised. Thus many of the non-identifying attributes such as the text formatting ones can have multiple values for each identifier value. To remove such dependencies through further normalisation would require the creation of more entity-types which would complicate the model considerably. The choice of identifiers indicates the numbering system for the statutes, thus section numbers continue to increment across parts whilst paragraph numbers are reset to unity in each schedule but continue to increment across subschedules.

It will be seen from figure 5 that this model shows an entity is determined to some extent by its environment. Thus, main text with identifier #5A in figure 4 has to be mapped onto two distinct entities depending on whether it is to be found in the main body of the act or to be found in a schedule. Because the numbering of subdivisions within schedules may depend on the nature of the subdivision, it is necessary in this model to introduce notional entities to represent these subdivisions. Further implications of the use of the E-R model for statutes are discussed elsewhere [Rossiter 1986a].

The graphical approach of the Chen entity-relationship model can be compared with the tabular nature of Codd's original ideas which have been extended by Borkin to include more semantic features. The purpose of Borkin's approach was to design a super model which could be used for comparing network and relational implementations. It therefore provides greater freedom to incorporate network aspects within a relational model which can be put to some effect in modelling statutes. The corresponding version of the paradigm of figure 4 using the Borkin approach is to be found in figure 6. This form of representation amplifies the rich semantic features to be found in the internal structure of a statute which require models of this capability. The notes at the end of the figure explain how the model is to be interpreted. Again constraints on the necessary occurrence of entities (which are important for database construction) are determined from the model applying the convention adopted by Borkin of case significance for predicate pairs (AGENT, OBJECT; agent, object). The advantage of this model is that it provides facilities for expressing semantic information by typing the class of a given attribute using what Borkin refers to as a 'characteristic' in row 3 of each relation. An extract from the Land Registration Act 1925 is shown in figure 7 marked with the various types of text employed as explained on page 2.

A hierarchical version of the statutory paradigm of figure 4 is presented elsewhere [Heather 1985] together with a discussion of the semiotic significance of text as surface structure representing deeper levels of knowledge.

5.2 IMPLEMENTING A SPIRES MODEL OF THE STATUTES

Having established a semantic, hierarchical or relational model for the statutes, the next stage is to design in terms of the software to be used the conceptual schema as discussed earlier[1]. In designing any schema choices have to be made about fundamental quantities. With statutes it is necessary to take decisions at the outset on the following:

a. the basic record unit; and
b. the method of identifying each unit.

5.2.1 Choice of Unit

With any of the models, it is necessary to decide in advance how to partition the textual data into basic units. Atomisation is a characteristic weakness [Heather 1986] implicit in the use of most formal methods. This is in contrast to natural language representations where the choice of unit can be dynamic and any decision may usually be postponed until the full circumstances and context of the use are known. The choice is usually governed by the storage capability of the system (e.g. track and block sizes, buffer restrictions, spanning facilities, etc), the human capacity for searching, retrieving and comprehending the information [Blair & Maron 1985] and the character of the document which will often be determined by traditional printing techniques. Thus, the size of a statute is controlled by parliamentary business and other political factors. The eighty or so acts passed by the UK parliament each year range from those with only one or two sections up to those such as the recent Companies Act [Companies Act 1985] with 747 sections and 25 schedules. The natural unit of a whole act would often be too large (up to half a megabyte for one of the typical larger statutes[2]) as a unit of retrieval but within an act there is no single sub-unit. There are two major subdivisions of main text and schedules together with preliminary materials and footnotes. The main text is divided into sections and the schedules into paragraphs. Sections may be grouped in parts and the schedules given various subdivisions. The solution adopted here is to define a non-homogeneous basic unit of record. Sections are used for main text, paragraphs for schedules, footnotes are taken together in one unit and other extraneous information in units of their own. The use of the section/paragraph has also been adopted by commercial legal full-text retrieval systems as their basic unit but they have in contrast selected a whole legal case rather than a single paragraph as the unit of record for their case report libraries. The reason for this seems to be that statutes are much more concentrated in their content than a case report, the whole of which may contain only

----

[1] Section 3.4, page 10.
[2] e.g. the Law of Property Act 1925 (c.20) consists of about 532,000 bytes.

Figure 6. A linearised representation of the internal structure of UK statutes using Borkin's Semantic Data Model

Act

be act:OBJECT					
act					
FT/P		FT/L	loose-text	short-text	short-text
year*	chapter*	date	title	preamble	crossnote

Part

in.act:AGENT		in.act:OBJECT be part:OBJECT	
act		part	
FT/P		FT/L	loose-text
year*	chapter*	number*	part-headings

Section

in.act:AGENT contains.part: agent		contains.part:object within.part:agent	in.act:OBJECT within.part:object be section:OBJECT	
act		part	section	
FT/P		FT/L	FT/P	whole-text
year*	chapter*	number	number*	section-text

Subsection

in.act:AGENT contains.part: agent		contains.part: object within.part: agent	in.act:OBJECT within.part: object in.section: AGENT	in.section:OBJECT be subsection:OBJECT	
act		part	section	subsection	
FT/P		FT/L	FT/L	FT/L	whole-text
year*	chapter*	number	number*	number*	subsection-text

Schedule

in.act:AGENT		in.act:OBJECT be schedule:OBJECT	
act		schedule	
FT/P		FT/L	loose-text
year*	chapter*	number*	schedule-headings

Subschedule

in.act:AGENT		in.act:OBJECT in.schedule:AGENT	in.schedule:OBJECT be subschedule:OBJECT	
act		schedule	subschedule	
FT/P		FT/L	FT/L	loose-text
year*	chapter*	number*	number*	subschedule-headings

Paragraph

in.act:AGENT		in.act:OBJECT in.sched:AGENT contains.sub- schedule:agent	contains.sub- schedule:object within.sub- schedule.agent	in.sched:OBJECT within.subschedule: object be paragraph:OBJECT	
act		schedule	subschedule	paragraph	
FT/P		FT/L	FT/L	FT/P	whole-text
year*	chapter*	number*	number	number*	paragraph-text

Subparagraph

in.act:AGENT		in.act:OBJECT in.sched:AGENT contains.sub- schedule:agent	contains.sub- schedule: object within.sub- schedule:agent	in.sched:OBJECT within.subschedule: object in.paragraph:AGENT	in.para- graph:OBJECT be subpara- graph:OBJECT	
act		schedule	subschedule	paragraph	subparagraph	
FT/P		FT/L	FT/L	FT/P	FT/L	whole-text
year *	chapter *	number*	number	number*	number *	sub- para- graph text

Notes for each relation :

row 1 - predicate:case pairs (for a pair, if 'AGENT,OBJECT' then the relationship is applicable to every row of data in the relation, if 'agent,object' then the relationship need not hold over every row)

AGENT	- agent of an association
OBJECT	- object of an association
BE	- existence of an entity

2 - entity type

3 - characteristic (whole-text fields are further resolved into a set of formatting subfields not included here). FT/P is force-text with physical control, FT/L is force-text with logical control.

4 - domain (* indicates fields which would comprise the primary key in the relational approach)

This represents only the principal features of the data structure. There are other entity-types involved such as footnotes, cross references, etc. For example the footnote entity-type is defined in an analogous manner to the entity-type "part" above.

one or two germane points.

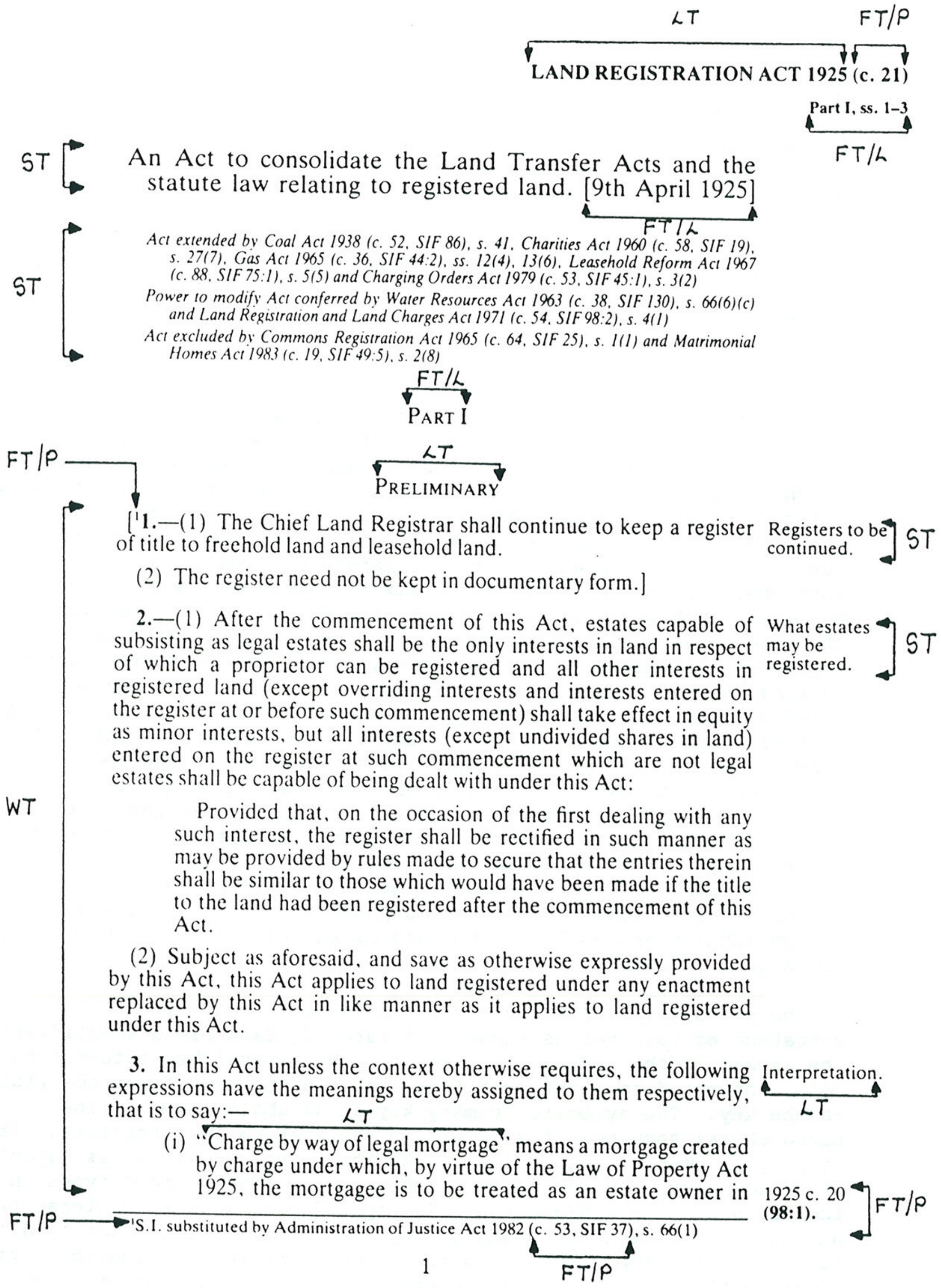
5.2.2 Choice of Key

It seems that humans can recall information in a variety of ways (e.g. recognition by signs, sounds, smells, words, etc) but on machines it is always necessary to have some primary address which maps to a physical storage area through an access method (e.g. hashing, B-trees, etc.). This is the function of the primary key. The simplest form of key is a numerical index (slot key) which is convenient for the system but which bears no direct relationship to the contents of the data. Other possible forms of the key are the ordinary hard-copy identification of the unit (natural key) and a semantic description of the content (symbolic key).

In this study, all three forms of the primary key have been tried. The slot organisation is the simplest to implement but provides no semantic information on the content of a record and machine look-up on the description of an item is not possible using the primary key. Use of a natural primary key in the standard form used by lawyers (e.g. 1976 (c.42) s.122) does in principle overcome these problems but it requires complex routines to parse it to ensure that interpretation by the computer is always possible. Examples of problems which arise in interpreting the natural key as derived from the HMSO tapes result from the use of mixed-case, non-printable characters and extraneous punctuation, as well as differing numbering systems such as arabic (e.g. 2), roman (II), letter (B) and word (Second). Further, the order of the sorted natural keys is not the logical order of the units within an act. The parsing routines to convert the natural key to a symbolic key were thus developed but their complexity makes it inefficient for them to be applied each time the database is searched. Consequently, the parsing routines were used once only during the build of the database to convert the natural key to a symbolic key more amenable to machine processing. This symbolic key proved to be the best candidate as a primary key and was finally adopted for this purpose in preference to the other two possibilities, although the natural key was retained for display of content to users.

The symbolic primary key comprises 14 data field-types concatenated together as shown in figure 8. Each field identifies one aspect of the content of a record, the natural key information being converted to right-justified arabic numbers at fixed positions in the key. The symbolic primary key in database terms is the hierarchical sequence of a record in the statute law structure. It is purely numeric, null values being stored as zeros, and is capable of rapid addressing on its component values. Three field-types in the key need some explanation: 'SI' specifies the class of legal text and has the value 1 for statute law; 'header' is set for act headings to distinguish the record-type assigned to the information consisting of act title, preamble, arrangement of sections and footnotes, which are all included together in one record following the structure to be

Figure 7. Extract from Land Registration Act 1925 showing Different Characteristics of Text


LT
FT/P
FT/L
ST
FT/L
ST
FT/L
LT
FT/P
ST
ST
WT
LT
LT
FT/P
FT/P
FT/P

LAND REGISTRATION ACT 1925 (c. 21)

Part I, ss. 1–3

An Act to consolidate the Land Transfer Acts and the statute law relating to registered land. [9th April 1925]

Act extended by Coal Act 1938 (c. 52, SIF 86), s. 41, Charities Act 1960 (c. 58, SIF 19), s. 27(7), Gas Act 1965 (c. 36, SIF 44:2), ss. 12(4), 13(6), Leasehold Reform Act 1967 (c. 88, SIF 75:1), s. 5(5) and Charging Orders Act 1979 (c. 53, SIF 45:1), s. 3(2)

Power to modify Act conferred by Water Resources Act 1963 (c. 38, SIF 130), s. 66(6)(c) and Land Registration and Land Charges Act 1971 (c. 54, SIF 98:2), s. 4(1)

Act excluded by Commons Registration Act 1965 (c. 64, SIF 25), s. 1(1) and Matrimonial Homes Act 1983 (c. 19, SIF 49:5), s. 2(8)

PART I

PRELIMINARY

[[1]**1.**—(1) The Chief Land Registrar shall continue to keep a register of title to freehold land and leasehold land.

Registers to be continued.

(2) The register need not be kept in documentary form.]

2.—(1) After the commencement of this Act, estates capable of subsisting as legal estates shall be the only interests in land in respect of which a proprietor can be registered and all other interests in registered land (except overriding interests and interests entered on the register at or before such commencement) shall take effect in equity as minor interests, but all interests (except undivided shares in land) entered on the register at such commencement which are not legal estates shall be capable of being dealt with under this Act:

What estates may be registered.

Provided that, on the occasion of the first dealing with any such interest, the register shall be rectified in such manner as may be provided by rules made to secure that the entries therein shall be similar to those which would have been made if the title to the land had been registered after the commencement of this Act.

(2) Subject as aforesaid, and save as otherwise expressly provided by this Act, this Act applies to land registered under any enactment replaced by this Act in like manner as it applies to land registered under this Act.

3. In this Act unless the context otherwise requires, the following expressions have the meanings hereby assigned to them respectively, that is to say:—

Interpretation.

(i) "Charge by way of legal mortgage" means a mortgage created by charge under which, by virtue of the Law of Property Act 1925, the mortgagee is to be treated as an estate owner in

1925 c. 20 **(98:1).**

[1] S.I. substituted by Administration of Justice Act 1982 (c. 53, SIF 37), s. 66(1)

1

FT/L is force-text with logical control, LT is loose-text,
FT/P is force-text with physical control, ST is short-text,
WT is whole-text.

found on the HMSO data tape. The field 'duplication no.' was only assigned a value when in the build of the database a key had been generated which was identical with one already occurring in the database. The value of 'duplication no.' was increased by one each time a potential duplicate occurred thus ensuring that every record could be added to the database. Its function is thus to allow the integrity constraint that the hierarchical sequence of each record should be unique to be violated, but to record when this violation had been allowed. The violation was only necessary in 29 of the 5978 records created. This low proportion of 0.5% indicates the suitability of the paradigm of the data. By providing this extra default field, it was possible to use fully automated techniques for building the database as all records would be added and exceptions to the basic statutory paradigm duly noted. The conceptual model could then be changed, the SPIRES file definition modified and another build made.

Another feature of the primary key is the need in some circumstances to record a range for one of the component fields such as 'section no.start' and 'section no.high'. This range is a further example of the non-homogeneous record unit in the situation when a number of units (such as a range of sections) need to be taken together in an ad-hoc group which is not a general group in the model such as 'parts'. This can be used for null records. Thus repealed sections are grouped together on the HMSO data tapes as e.g. 'Sections 27-32'. By treating them in this way it means that if a run-time search is made for section 29, the system can find it in this range and return section 29 as a null section being a member of the repealed group. At first sight the more fundamental approach would seem to be to unwrap the range at build-time and store separate null sections as ordinary sections, but this would result in loss of information which is preserved by providing maximum and minimum values.

Section and paragraph numbers have been allowed to use the same fields as it is always possible to determine by context whether a section or paragraph is involved. A paragraph is only to be found in a schedule and so reference may always be made to ascertain whether 'schedule no. start' has a non-zero value. However, it may be that the shortening of the primary key obtained through this device is not justified and that a full representation of the symbolic key with separate fields for section and paragraph numbers would be better. Subsequent work on manipulating the text records suggests that for efficiency, context has to be dealt with by direct field-type attribution and not by indirect look-up.

5.2.3 Constructing the SPIRES File Definition

The file definition language, the first of the SPIRES language processors mentioned above, is employed to construct a meta-data record. SPIRES holds meta-data as data and this record is kept in the subfile FILEDEF. FILEDEF can be handled like any other SPIRES

```
---------------------------------------------------
| SI | year | chapter | header | part no.| part no.|
|    |      |   no.   |        |  start  |  high   | ........
---------------------------------------------------

--------------------------------------------------
| schedule | schedule | subschedule | subschedule |
| no.start | no.high  | no.start    | no.high     |  .......
--------------------------------------------------

-----------------------------------------------------------
| section or para- | section or para- | subsection or sub- |
| graph no.start   | graph no.high    | paragraph no.      |  ...
-----------------------------------------------------------

--------------------
| duplication No.  |
--------------------
```

Figure 8. Symbolic Key for Statute Law

subfile but it also performs the role of the system's data dictionary as each of its records provides a full specification of the structure of the data for every application at that SPIRES site.

Appendix I contains the full file definition for British statutes. Reference may be made to the SPIRES manual [SPIRES File Definition Manual 1982] for a complete specification of the file definition language. Comments have been added at appropriate points to explain particular details. The overall structure of the file definition itself is outlined in figure 9. The symbolic key shown in figure 8 is put together from the components needed for unique specification of a primary key. These components are obtained by parsing the natural key as explained previously. The procedure BUILDKEY is invoked on the left-hand column of page 40 to accomplish this, and is itself defined on page 46. Before addition, a look-up is made in the records so far completed for the provisional key constructed in this way. Once the database is constructed, it is then possible to recover any of the components such as CHAPTER, SCHEDULE by the use of the procedure V which is defined on page 50 and invoked by the respective virtual element as on page 41.

The characteristics of the type of text corresponding to row 3 in the Borkin model of figure 6 use the descriptive terminology of figure 1. The SPIRES model however implements these in a variety of ways as appropriate to the software. Thus whole text has such a complex fine structure which it is necessary to preserve in order to provide the proper output formats for human comprehension that it has to be categorised and mapped on to separate elements, i.e SPIRES fields. These can be found on pages 42-45 of the SPIRES FILEDEF of Appendix I. FORCE-TEXT in the statutes (as in the examples of figure 7) on the other hand can be better dealt with in the SPIRES model by more prescriptive methods. Textual data type control, whether physical or logical, is related to database validation [Rossiter 1984] and can be implemented through the use of SPIRES Actions. These are processing rules which may be user defined but a large number of pre-defined actions are already provided in SPIRES to examine, validate, modify input and output values and convert to some form for efficient internal storage. There are about two hundred SPIRES actions including some to deal with text strings relating to items such as telephone numbers, dates, personal names, money, time, etc. The rules can have qualifiers attached to them and may be combined together. Figure 9 shows the place of actions within a SPIRES file definition structure.

Simple examples of FORCE-TEXT (control physical) are to be found in the way that integers are treated by $INT and unpredictable distributions of blanks within text by $SQU in the file definition on page 40 of Appendix I. FORCE-TEXT (control logical) is more complicated, but $DATE is an example of a SPIRES action that can take care of dates in various forms. For fuller details of these actions reference should be made to the SPIRES File Definition manual [SPIRES File Definition Manual 1982]. It is the existence of these features that makes it possible to construct a SPIRES model that can cope automatically both on input and output with the amorphous nature of text.

As indicated earlier, the indexing for full text is very important and this is achieved by the use of a further record-type REC02, the structure of which is defined on page 50. The structure itself is quite simple consisting only of a value and one or more pointers. The linkage defined on page 54 is in two parts. The PASSPROC statement controls the building of the index. Full-text values are passed from 82 elements to this one index, all punctuation is converted to spaces and all values are broken on spaces to form a word index. Words shorter than three characters long or on the 'stop list' are excluded from the index. The SRCPROC statement controls the searching of the index and enables queries to be transformed or mapped on to the index under the control of the specified actions.

In textual documents, it is vital to retain the order of the lines in the data structure. This is achieved through creating the structure DETAIL (page 41) which is fully defined on pages 42-45. The structure can occur any number of times, each occurrence comprising an integer held in the element SEQ and a text string held by one of the 81 optional elements, each element-type indicating a different formatting characteristic. The structure occurrences are sorted in ascending order on the value for SEQ which is effectively the line number of each line in the record. In Stonebraker's terms [Stonebraker 1986], SEQ is a form of LID (line identifier) which is used for ordering relations in INGRES as a first step towards use of

```
File ---> Record ---->  Element   -----> Element ----  type
 |          /           category         name    ----  occurrence
 |         /           (- fixed                  ----  length
 V        V             - required               ----> comment
  Subfile               - optional               ----> alias
                        - virtual)               ----> actions
                                                       (- inproc
                                                        - outproc)


 where ----- represents 1:1 association,
       ---->            1:N
```

Figure 9. An outline of the Structure of the SPIRES File Definition

the system for document processing.

5.2.4 Use of the File Definer

The construction of the above native file definition is a relatively procedural method of defining the conceptual schema. However, SPIRES also provides a direct non-procedural method of implementing a conceptual schema by the use of its FILE DEFINER[1] which produces the native file definition language as object code. The input to the file definer of the conceptual schema for the statutory structure is reproduced in figure 10. The native file definition produced by the File Definer is given in Appendix II. This differs in some details from that of Appendix I which is the file definition actually employed for the data base structure. This is because the full range of definition facilities is only available with the native language, so that it is necessary to modify the output from the File Definer to satisfy all the requirements of the project. In particular, use of the native file definition language introduces the capability for a programmer to create his own action rules and to optimise index searching and updating. Normally this extra refinement would be added to a file definition produced by the File Definer. In this instance the file definition of Appendix I was programmed directly before the File Definer was available from Stanford. Appendices I and II are reproduced here in full for comparison between the use of third- and fourth-generation programming tools.

----

[1] The file definer is the type of software currently referred to as 'a fourth-generation productivity tool' and full details are to be found elsewhere [SPIRES File Definer Manual 1981].

5.2.5 The Project in ANSI/SPARC terminology

Rationalizing the architecture of SPIRES in terms of the ANSI/SPARC architecture described earlier is not an easy task. An attempt is shown in figure 11 in which the file definer input is equated to the conceptual schema and the native file definition language to an internal schema. This correspondence though is only approximate. The file definer input contains information on which fields are indexed, which should be present only in the internal schema. The native file definition language contains the action rules to enforce integrity which should be in the conceptual schema.

Figure 11 also shows that the semantic models employed are not part of the ANSI/SPARC architecture. In general, the use of semantic modelling features in the initial analysis provides a structure which cannot completely satisfy the boundary conditions if automatic mapping into a conceptual data model is attempted. Probably for this reason, ANSI/SPARC have yet to address the problem of integrating semantic modelling into a database architecture.

5.2.6 The Construction of the Database

From the point of view of modern knowledge engineering, the SPIRES File Definition consisting of entities, relationships, protocols and actions is a knowledge base containing a machine representation of the sum of human knowledge about the internal architecture of English statutes as they exist today. Such a knowledge base should be able to take printed versions of statutes and from the output of an optical scanner, extract the structural information from the nature of the formatting and typesetting. We are concerned with the data modelling and were able to avoid errors from the character-reading apparatus by using "pure data", namely the computer type-setting code that drives the printer producing the ordinary printed version and which is related to the printed version by a unitary transformation.

The data tapes containing the official versions of Acts of Parliament were kindly supplied by the Controller of Her Majesty's Stationery Office. HMSO publish statutes in groups arranged according to subject matter. Two groups provided on separate magnetic tapes consisting of a range of statutes from the thirteenth century until the present day were used. The aim was to establish a legal information retrieval system for teaching and research of sufficient size to provide a realistic sample of real-world conditions. A full set of UK statutes amount to about 200Mb. A 2Mb corpus of case law was already available and it was considered that a data base of the order of 10Mb was the minumum necessary to simulate for teaching and research the open-endedness and unpredictable search result characteristics of the real world. The first tape of about 2 Mb containing the Landlord & Tenant Group was used for the design of the file definition and to build the first subfile. The second tape of about 5 Mb containing the Property & Conveyancing Statutes was then used as raw data for the original file definition and a second

Figure 10. The Conceptual Schema for Statute Law in File Definer Input Form

```
File STATLT.DEF
Goal STATLT
Subfile STATLT
Fixed
ID/ key/ length 36, exact
RECNO/ integer/ occurrence 1/ index
Required
ACCESS-NO, SN, AN/ length 252, maximum/+
  occurrence 1/ squeeze
ACT-TITLE, AT, TITLE/ occurrence 1/ +
  squeeze/ index/ word
KIND/ occurrence 1/ index
DATE/ date/ occurrence 1/ index
TAPE/ integer/ occurrence 1
Optional
MARG-N-OTHER, MARG, MN, XOE-4T/ +
  index/ word
BROWSE, BROW/ index/ word
INFO, WORD, W/ index/ word
SCHEDULE-TEXT, X37-3T/ index
SECT-MAIN-TEXT, X5A-2T/ index info
TEXT-IN-COL, ENGLISH, X7A-4T/ index info
TEXT, X7F-3T/ index info
SINGLE-INDENT2, X7B-2T/ index info
MARG-N-REF-ACT, XOC-4T/ index info
NOT-IND-SUB-SIN, XF7-2T/ index info
PART-TEXT, X03-5T/ index info
NOTE-EMB-SIN-IND, X5B-2T/ index info
SINGLE-INDENT-SM, XF2-2T/ index info
MARG-N-ACT-RUFF, XF6-2T/ index info
SECTUNK12, XF5-2T/ index info
SECTUNK11, X2E-2T/ index info
GEN-BOLD-TYPE, X02-2T/ index info
SCHSOURCE/ occurrence 1
DETAIL/ structure/ +
  closeout $sort(ascend.err)
  Fixed
  SEQ/ key/ integer 2
  Optional
  GEN-BOLD-TYPE, X02-1 / index info
  UPPER-CASE, X03-1/ index/ word
  ARRANGE, XF3-1/ index info
  PREAMBLE, XF0-1/ index info
  MINOR-COL-HEAD, X16-1/ index browse
  TWO-COLUMNS, X61-1, ARRANGE-SECTIONS/+
    index browse
  GEN-SUBHEADING, X2D-1/ index browse
  SMALL-REPRO-TXT, XF2-1/ index info
  REPR-LONG-TITLE, XF5-1/ index info
  GENUNK3, XF7-1/ index info
  SMALL-REPRO-MRG, XF8-1/ index info
  SMALL-REPRO-COL, X18-1/ index info
  MARG-N-REF-ACT, XOC-4/ index info
  CROSSNOTE, X4B-4/ index info
  FOOTNOTE, X3F-4/ index info
```

```
  FOOTN-OLD-STAT, XF9-4/ index info
  OMISS, XF1-4/ index info
  OLD-STAT-BANNER, X7F-4/ index info
  TEXT-IN-COL, ENGLISH, X7A-4/index info
  GEN-UNID, LATIN/ index info
  SCHEDULE-TEXT, X37-3/ index info
  CENTRE-HEAD, X2E-3/ index info
  NUMB-PARA, X5A-3/ index info
  TEXT, X7F-3/ index info
  SINGLE-IND-SUB, X0B-3/ index info
  SINGLE-IND-SUB2, X6C-3/ index info
  DOUBLE-IND-SM, X5B-3/ index info
  DOUBLE-IND-SM2, X5D-3, NIL3/index info
  DOUBLE-IND-SM3, X7D-3/ index info
  DOUBLE-IND-SM4, X4D-3/ index info
  TRIPLE-IND, X4E-3/ index info
  TRIPLE-IND2, X10-3/ index info
  COL-PR-HEAD, X11-3/ index info
  COL-PR-HEAD2, X3D-3/ index info
  COL-PR-DOUBLE, X12-3/ index info
  COL-PR-DOUBLE2, X3C-3/ index info
  COL-PR-TEXT, X13-3/ index info
  PRINT-TABULAR, XF0-3/ index info
  STARRED-NOTE-SM, XF2-3/ index info
  COL-PR-HEAD-UNRL, XF3-3, +
    ARRANGE-SCHEDULE/ index browse
  COL-PR-HEAD-REPR, XF6-3/ index info
  COL-PR-HEAD-MED, X61-3/ index browse
  SIN-IND-SUB-NOTE, X7B-3/ index info
  COL-PRINT, XTOF00-3/ index info
  FORM-TIT-BOLD-UC, X2F-3/ index info
  INSTRUCTION, XF7-3/ index info
  CARRIAGE-CONTR, X0D-3/ index info
  COL-PR-HEAD3, X25-3/ index info
  COL-PR-HEAD-SM, XF4-3/ index info
  DOC-TIT-CNT-ITAL, X2D-3/ index browse
  DBL-IND-SUB-NIL2, X50-3/ index info
  PART-TEXT-IN-SCH, X03-3/ index browse
  SCHED-UNID/ index info
  SECT-MAIN-TEXT, X5A-2/ index info
  CONT-TEXT, X7F-2/ index info
  INDENT-TEXT-SUB, X25-2,NIL6/index info
  SINGLE-INDENT-SM, XF2-2/ index info
  SINGLE-INDENT, X6C-2/ index info
  HALF-I-SIN-I-SUB, X7B-2/ index info
  DBL-INDENT-LIST, X7D-2/ index info
  DBL-IND-SUB-NIL, X50-2/ index info
  NOT-IND-SUB-SIN, XF7-2/ index info
  REPR-MARG-NOT-SM, XF8-2/ index info
  NOTE-EMB-SIN-IND, X5B-2/ index info
  DOUBLE-INDENT, X5C-2/ index info
  DOUBLE-INDENT2, X5D-2/ index info
  DBL-IND-FST-MORE, X4D-2/ index info
  TRIPLE-IND-LIST, X60-2/ index info
  TRIPLE-IND-LIST2, X4E-2/ index info
```

```
  MARG-N-ACT-RUFF, XF6-2/ index info
  SECTUNK11, X2E-2/ index info
  SECTUNK12, XF5-2/ index info
  SECT-UNID/ index info
  PART-TEXT, X03-5/ index browse
  SINGLE-IND-SM-P, XF2-5/ index info
  REP-MARG-NOT-SMP, XF8-5/ index info
  PART-SUBHEADING, X2D-5/ index browse
  CONT-TEXT-PART, X7F-5/ index info
  NOT-IND-SUB-SINP, XF7-5/ index info
  INDENT-TEXT-SUBP, X25-5/ index info
  PART-UNID/ index info
  end
KEY/ occurrence 1
KEYHEAD/ occurrence 1
TYPESCH/ occurrence 1/ index
TYPESUBSCH/ occurrence 1/ index
SO
OUTNO
LARGACT/ occurrence 1
PARTLOW.SCT/ integer/ occurrence 1
PARTHIGH.SCT/ integer/ occurrence 1
CHAPTER.SCT/ integer/ occurrence 1
SOW
OUTNOW
Virtual
SI/ integer/ index
YEAR/ integer/ index
CHAP/ integer/ index
HEAD/ integer/ index
SCHED/ integer/ index
SCHEDHIGH/ integer/ index
SCHEDM/ integer/ index
SUBHIGH/ integer/ index
SCHEDN/ integer/ index
SECT/ integer/ index
SECTHIGH/ integer/ index
MISC/ integer/ index
CON/ integer/ index
DUPL/ integer/ index
PARTLOW/ integer/ index
PARTHIGH/ integer/ index
SUBPART/ integer/ index
Goal REC40
Subfile SECTION.IN.PART
Fixed
YEARCHAPSECT/ key/ length 9, exact
PARTSPEC/ length 6, exact
Goal ZSCH01
Subfile PARA.IN.SUBSCHEDULE
Fixed
YEARCHAPSCHPARA/ key/ length 12, exact
SUBSCHSPEC/ length 4, exact
end
```

subfile build. The success of the second build proved the adequacy of the original file definition. It so happened that HMSO had slightly changed the format of its tapes before supplying the second tape. This particular change was only to the form and not the structure and the appropriate way to deal with this would be to modify the SPIRES input format[1]. Normally it is not good practice to alter the raw data as this may unwittingly lead to the loss of information. On this occasion, we wanted to test the SPIRES model for reproducibilty on independent data. This second HMSO data tape was modified manually by applying global commands to the whole tape in the MTS context editor thus restoring it to HMSO's original format. The second tape was then compiled automatically without any difficulties although it later emerged that it contained structure not found on the first tape.

6 CONCLUSIONS

This implementation of the internal architecture of English statutes has been fully implemented in SPIRES and has provided a full-text legal-information retrieval-system for teaching the art of legal problem solving in the electronic medium as well as for further research [Heather 1983]. The real significance of this success is that it has been achieved using totally automatic means. Once the schema had been devised and the file definition programmed, SPIRES was able to accept any statute. The rationality of the design of the SPIRES system was even able to cope with the rationality of English Statutes in ways that were not planned when the first file definition was written. The footnotes are an example of this. Statutory draftsmen have made greater use of footnotes over the last century and by process of natural development they have adopted rational methods of dealing with them which are close to good database principles. The result was that SPIRES took care of the footnotes without explicit design. To capture automatically the whole statutory structure by programming in an ordinary high-level language would have been quite beyond our capabilities and resources. Indeed it has been possible to accomplish a major project without assistance and as a spare-time activity.

From a conceptual point of view, this implementation shows the advantages of using database technology for textual documents. Although not easy, the selection of an appropriate model based on a rigorous theoretical view enables the whole of a document to be represented in internal machine form. One important characteristic of digital systems is that a copy can be as good as the original and

----

[1] The Format Definition for the statutes written in the SPIRES format language controls both input and output formats and is itself of comparable size and complexity to the File Definition of Appendix I. It is not possible to give full treatment to the Format Definition here and the intention is to develop a separate report on it.

```
     |---------------------|                  |---------------------------|
     | Semantic Modelling  |                  | Entity-Relationship Model |
     |---------------------|                  | Borkin Semantic Model     |
                     .                        |---------------------------|
                     .                             .
  ANSI/SPARC         .                             .             ANSI/SPARC
 ==============================     =====================================
 |                   .         |    |      .                             |
 |  ---------------  .         |    |      .    ---------------------    |
 |  |External Data|  .         |    |      .    | SPIRES Subfile    |    |
 |  |   Models    |  .         |    |      .    |   Definitions     |    |
 |  ---------------  .         |    |      .    |(Access/Privileges)|    |
 |          \        .         |    |      .    ---------------------    |
 |           \       .         |    |      .   /                         |
 |        -----------------------   |  ----------------------------      |
 |        |Conceptual Data Model|   |  |SPIRES File Definer Source|       |
 |        -----------------------   |  ----------------------------      |
 |                   |         |    |              |                     |
 |        ---------------------     |  --------------------------------  |
 |        |Internal Data Model|     |  |SPIRES Native File Definition|   |
 |        ---------------------     |  |         Language            |   |
 |                   |         |    |  --------------------------------  |
 |                   |         |    |              |                     |
 |        ---------------------     |   ---------------------------      |
 |        |Physical Data Model|     |   |Compiled SPIRES Database|       |
 |        ---------------------     |   |       Definition       |       |
 |                             |    |   ---------------------------      |
 ==============================     =====================================
```

Figure 11. Levels of SPIRES Definitions and the E-R and Borkin Models in Conventional Database Terminology

this work has shown that this principle holds true for the digital modelling of complex text. The compiled SPIRES database so far as we are aware, contains every jot of information contained on the printed page. It would be possible to drive a typesetting program to reproduce the original printed version but more important it is possible for any user to re-organize the data logically to reproduce the same information in a totally different form without the loss of any information. It is this move from the physical to the logical form which can be achieved using database techniques.

ACKNOWLEDGEMENTS

We are grateful to Newcastle University Computing Laboratory and Newcastle Polytechnic Computer Unit for NUMAC facilities with the use of their IBM 370/168 and Amdahl 5860 computers; to Stanford University Center for Information Technology for the use of SPIRES which has full-text capabilities not otherwise available to us; to Her Majesty's Stationery Office who provided us with their typesetting data tapes containing full statutory structures.

To handle large complex data sets requires powerful hardware, sophisticated software as well as good quality data. This project could not have been carried out without access to a combination of all three.

BIBLIOGRAPHIC REFERENCES

page

Agosti, M, (1983-4), Specialised Hardware for Database Management and Information Retrieval: a classified bibliography, ACM SIGIR Forum **18** 13-40. 9

ANSI/X3/SPARC DBMS Framework (1978). Report of the Study Group on Data Base Management Systems, Information Systems **3**. 10

Bachman, C W, (1980), The Role Data Model Approach to Data Structures, Proceedings, International Conference on Data Bases, University of Aberdeen. 13

Bachman, C W, and Day, M, (1978), The Role Concept in Data Models, Proceedings, 3rd International Conference on Very Large Data Bases, Tokyo. 13

Blair, David C, and Maron, M E, (1985), An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System, Communications of the ACM, **28** 289-299. 23

Borkin, S A, (1979), Equivalence properties of semantic data bases for database systems, Technical Report of Laboratory for Computing Science, MIT, TR-206. 13

Burnard, L, (1985), CAFS and Text: the View from Academia, ICL Technical Journal **4** (4) 468-482. 11

Carmichael, J W S, (April 1986), CAFS and Text - an Update, BCS 8th Research Colloquium on Information Retrieval, University of Strathclyde. 9

Carmichael, J W S, (November 1984), The application of ICL's CAFS to Text Storage and Retrieval, PROTEXT I, Dublin. 9

Chen, P P-S, (1976), The Entity-Relationship Model - towards a unified view of data, ACM Trans Database Systems **1** (1) 9-36. 13

Codd, E F, (1970), A Relational Model of Data for Large Shared Data Banks. Communications of ACM **13** (6) 377-387. 11

Codd, E F, (1979), Extending the Database Relational Model to capture more meaning, ACM Trans Database Systems **4** 397-434. 13

Comer, D, (1979), The Ubiquitous B-tree, ACM Computing Surveys **11** 121-137. 15

Companies Act 1985 (c.6) HMSO London. 23

Connolly, M, (1986), High level language constructs for relational database design, Ph. D. thesis, Queen's University of Belfast. 8

Date, C J, (1981), An Introduction to Database Systems, third edition, Systems Programming Series, Addison-Wesley **1**. 10

Date, C J, (1983), Introduction to Database Systems, Addison-Wesley **2** 341-361. 9

Haworth, G McC, (1985), The CAFS System today and tomorrow, ICL Technical Journal **4** 365-392. 9

Heather, M A, (1982), Legal Structures for Law Machines: in C. Ciampi, Artificial Intelligence and Legal Information Systems, North Holland, Amsterdam **1** 117-127. 17

Heather, M A, (1983), Teaching and Researching Legal Information Systems: the Newcastle Electronic Law Library, Informatica e Diritto **IX** part 3, 175-181. 33

Heather, M A, (1985), Semiotic Orders in Law, Second Annual Conference on Law and Technology: Legal Language, Computational Linguistics and Artificial Intelligence, Houston. 22

Heather, M A, (1986), Future Generation Systems in the Service of the Law, in: Automated Analysis of Legal Texts, eds. Martino, A A, Socci Natali, F, North-Holland, Amsterdam 643-660. 23

Heather, M A, and Rossiter, B N, (1986), The Textual Environment and Database Management Systems, Third Symposium on Empirical Foundations of Information and Software Science 1985, Risoe National Library, Denmark (to be published by Plenum). 1

Heine, M, (1984), The flow of control in a communication process, Cybernetica **27** 57-64. 5

HMSO Technical Services, (1981), General Specification of HMSO Data Tapes for Information Retrieval, HMSO London. 18

Howe, D R, (1983), Data Analysis for Data Base Design, Edward Arnold, London 126-167. 19

Hutchinson, A, (1986), Some Practical Principles for Design of Maintainable Systems, Computer Journal **29** (1) 20-23. 5

ICL Computer Users Association (UK), (November 1985), CAFS in Action, Working Party Report. 9

IKBS Special Interest Group Reports (1984), Proceedings of Workshops no.1 and no.2 on Architectures for Large Knowledged Based Systems, Manchester University 1984, IEE. 1

Kahn, David, (1963), Plaintext in the New Unabridged: An Examination of the Definitions on Cryptology In Webster's Third New International Dictionary, Crypto Press, New York. 6

Knuth, Donald E, (1979), TEX and METAFONT, New directions in Typesetting, Digital Press, 6

Massachusetts.
Law of Property Act 1925 (c.20) HMSO London. 23
Lorie, R A, (1981), Issues in databases for design application, IBM Research Report RJ3176. 13
Macleod, I A, and Crawford, R G, (1983), Document Retrieval as a Database Application, Inf Technol Res & Dev **2** (1) 43-60. 1
Moto-oka, T, et al, (1982), Challenge for Knowledge Information Processing Systems, in Moto-oka, T, (editor), Fifth Generation Computing Systems, North-Holland. 9
Radecki, T, (1983), A Theoretical Background for Applying Fuzzy Set Theory in Information Retrieval. Fuzzy Sets and Syst. **10** (2) 169-183. 1
Report on CAFS Usage at Oxford, (1985), Annual Report of Oxford Computing Service 1984-85, Oxford. 9
Rossiter, B N, (1980), The SPIRES Database Management System, Proceedings SEAS Spring Technical Meeting at Durham, March 1980, 255-261. 15
Rossiter, B N, (1984), Introduction to Database Management Systems, Computer Physics Communications **33** 5-12. 10, 29
Rossiter, B N, (1986), Full Text Data Base Management Systems: A Model and Implementation for Law, in: Automated Analysis of Legal Texts, eds. Martino, A A, Socci Natali, F, North-Holland, Amsterdam 899-916. 22
Rossiter, B N, (1986), Machine Awareness in Database Technology, Proceedings Symposium VI, Meta-intelligence and the Cybernetics of Consciousness. XI International Congress of Cybernetics, Namur, September 1986. 5
Rossiter, B N, and Heather, M A, (1986), Data Models and Legal Text, Thirteenth International Conference of the Association for Literary and Linguistic Computing, April 1986, University of East Anglia, Norwich. 2
Sakai, H, (1983), Entity-Relationship Approach to Logical Database Design, published as part of Entity-Relationship Approach to Software Engineering, North-Holland 155-187. 19
Schmid, H A, and Swanson, J R, (1975), Proceedings ACM SIGMOD, International Conference on Management of Data. 13
Schmidt, J W, (1977), Some high level constructs for data of type relation, ACM Trans Database Systems **2** 247-261. 8
Schmidt, J W, and Brodie, M L, (1983), Relational database systems: Analysis and Comparison, Springer-Verlag. 11
Schnoeder, J R, Kiefer, W C, Guertin, R L, Berman, W J, (1976), Stanford's Generalized Database System, 15

Proceedings Very Large Data Base Conference.

Shannon, C, and Weaver, W, (1949), The Mathematical theory of communication, Illinois. 5

Smith, Joan M, (June 1986), The Implications of SGML for the Preparation of Scientific Publications, Computer Journal, **29** (3) 193-200. 6

Sparck Jones, K, and Kay, M, (1973), Linguistics and Information Science. Academic Press, New York. 1

SPIRES File Definer Manual, (1981), Center for Information Technology, Stanford University. 30

SPIRES File Definition Manual, (April 1982), Center for Information Technology, Stanford University. 15,28,29

Stevens, M E, (1965), Automatic Indexing: A State of the Art Report, Natl. Bur. Std. (US) Monograph 91. 3

Stonebraker, M, (1986), Document Processing in a Relational Database System, The INGRES Papers, Addison-Wesley 357-375. 13, 29

Teskey, F N, (1982), Principles of Text Processing, Ellis Horwood. 9

TEXTFORM User Guide (May 1981), Computing Services, University of Alberta. 6

William & Glyn's Bank Ltd v Boland [1981] AC 487. 18

Winkworth v Edward Baron Development Co Ltd and Others 1986 LSG 83(3) 199. 18

Zunde, Pranas, (1984), Empirical Laws and Theories of Information and Software Science. Information Processing Management **20** (1-2) 5-18. 17

Appendix I: SPIRES FILEDEF for Statute Law

```
 FILE = LAU2.STATLT;
 COMMENTS = Definition for physical file holding the
logical (sub)files PROPERTY LAW STATUTES (called STATLT),
CASES, YCSP, and SUBSCHINFO, together with indexes to
PROPERTY LAW STATUTES and CASES;
 AUTHOR = Nick Rossiter, Computing Laboratory, Newcastle
University, & Michael Heather, Department of Law, Newcastle
Polytechnic;
 DEFDATE = Fri. June 6, 1980;
 MODDATE = Tues. June 24, 1986;
 MODTIME = 13:21:16;
 MAXVAL = 16384;
 STATISTICS = 2;
 RECORD-NAME = REC01;
   COMMENTS = defines record-type for PROPERTY LAW
STATUTES, provides physical structure of B-Trees with keyed
and residual data sets;
   REMOVED;
   FIXED;
     KEY = ID;
       OCC = 1;
       LEN = 36;
       INPROC = $CALL(IDSCH)/  $CALL(TRANSKEY)/
$CALL(FETCHSUBSCH)/  $CALL(TRANSKEY)/ $CALL(STORESUBSCH)/
$CALL(IDSECT)/  $CALL(IDPARA)/ $CALL (CHECKPARA)/
  $CALL(BUILDKEY)/ $LOOKUP(REPLACE,REC01,KIND,D)/
$CALL(TESTREC)/ $CALL(INITIALISE,INCLOSE) / $MAX.OCC(1,
KEEP.LAST);
       COMMENTS = the symbolic key, built by the process
defined in the INPROC from the data values derived in the
build for its components;
     ELEM = RECNO;
       OCC = 1;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
       COMMENTS = a slot number, unique for each record;
   REQUIRED;
     ELEM = ACCESS-NO;
       OCC = 1;
       INPROC = $SQU /$TRIM / $MAX.LEN(252,KEEP.FIRST,W)/
$CALL(TRANSHEX);
       ALIASES = AN, SN;
       COMMENTS = the natural language form of the symbolic
key e.g. 1974 c.19 Sch.I para.5;
     ELEM = ACT-TITLE;
       OCC = 1;
       INPROC = $SQU /$TRIM / $CALL(TRANSHEX);
       ALIASES = AT, TITLE;
       COMMENTS = the title of an act;
     ELEM = KIND;
       OCC = 1;
COMMENTS = characterises the type of data the record is
holding by its value as follows: 1 - act headings, 2 -
section, 3 - schedule, 4 - act headings (continuation), 10 -
part.  6 - 9, 11 - 16 indicate record starts with marginal
note, 1st element after marginal note being: 6 -
text-in-col, 7 - text, 8 - single-indent2, 9 -
not-ind-sub-sin, 11 - note-emb-sin-ind, 12 -
single-indent-sm, 13 - marg-note-act-ruff, 14 - sectunk12,
15 - sectunk11, 16 - gen-bold-type;
     ELEM = DATE;
       OCC = 1;
       LEN = 4;
       INPROC = $DATE(CONVERT,,W);
       OUTPROC = $DATE.OUT(1);
       COMMENTS = either the complete date if one can be
extracted from the preamble or the year of the act otherwise
;
     ELEM = TAPE;
       OCC = 1;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
       COMMENTS = indicates which tape the record was
derived from - 1 = LAU1PS, 2 = LAU1LT;
   OPTIONAL;
     ELEM = MARG-N-OTHER;
       INPROC = $CALL(TRANSHEX);
       ALIASES = MARG, MN, X0E-4T;
COMMENTS = marginal note provides comment on content of the
subsequent component of the law, this is the first of a
series of 15 optional record-level elements to hold the
header line of each record e.g.  the heading of a schedule
or of a part, the first line of a section or paragraph, the
marginal note which starts a subsection, or several rarer
forms.  These header lines are unsequenced but may be
thought of conceptually as sequence number 0 in a record;
     ELEM = SCHEDULE-TEXT;
       INPROC = $CALL(TRANSHEX);
       ALIASES = X37-3T;
     ELEM = SECT-MAIN-TEXT;
       INPROC = $CALL(TRANSHEX);
       ALIASES = X5A-2T;
     ELEM = TEXT-IN-COL;
       INPROC = $CALL(TRANSHEX);
       ALIASES = ENGLISH, X7A-4T;
     ELEM = TEXT;
       INPROC = $CALL(TRANSHEX);
       ALIASES = X7F-3T;
     ELEM = SINGLE-INDENT2;
       INPROC = $CALL(TRANSHEX);
       ALIASES = X7B-2T;
     ELEM = MARG-N-REF-ACT;
       INPROC = $CALL(TRANSHEX);
       ALIASES = X0C-4T;
       COMMENTS = marginal note providing act and chapter
of cross-reference to another act;
     ELEM = NOT-IND-SUB-SIN;
       INPROC = $CALL(TRANSHEX);
       ALIASES = SECTUNK, XF7-2T;
     ELEM = PART-TEXT;
```

```
          INPROC = $CALL(TRANSHEX);
          ALIASES = X03-5T;
        ELEM = NOTE-EMB-SIN-IND;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK3, X5B-2T;
        ELEM = SINGLE-INDENT-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = XF2-2T;
        ELEM = MARG-N-ACT-RUFF;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK9, XF6-2T;
          COMMENTS = marginal note providing a cross-reference
to another act in Ruffhead form;
        ELEM = SECTUNK12;
          INPROC = $CALL(TRANSHEX);
          ALIASES = XF5-2T;
        ELEM = SECTUNK11;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X2E-2T;
        ELEM = GEN-BOLD-TYPE;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X02-2T;
        ELEM = SCHSOURCE;
          OCC = 1;
COMMENTS = default value is SCHEDULE, but can also be PART,
FORM, TABL (for Table), SPEC (for Specimen), CASE, NUMB (for
NUMBER), DTHY (for dot.hyphen), CHAP (for Chapter).  This
field indicates the content of the X37 line, which can be
e.g.  Schedule 1, Specimen no.  3, .-6;
        ELEM = DETAIL;
          TYPE = STR;
          INPROC = $SORT(ASCEND.ERR,S);
        ELEM = KEY;
          OCC = 1;
          INPROC = $CALL(DELETE,INCLOSE);
          COMMENTS = provisional symbolic key passed from
input format and then deleted;
        ELEM = KEYHEAD;
          OCC = 1;
          INPROC = $CALL(DELETE,INCLOSE);
          COMMENTS = value for HEAD passed from input format
and then deleted;
        ELEM = KEYDUPLSTART;
          OCC = 1;
          COMMENTS = initial value for DUPL passed from input
format;
        ELEM = TYPESCH;
          OCC = 1;
          COMMENTS = manner in which number of a schedule is
presented: ROMAN, LETTER, or WORD. Default for arabic
numerals is '';
        ELEM = TYPESUBSCH;
          OCC = 1;
          COMMENTS = manner in which number of a subschedule
is presented: see TYPESCH;
        ELEM = SO;
```

```
          COMMENTS = given values in first record from A-Z,
AA-ZZ;
        ELEM = OUTNO;
          COMMENTS = given values in first record from 1-52;
        ELEM = LARGACT;
          OCC = 1;
          COMMENTS = has the value ON for large acts with
arrangements of sections, and OFF for small acts with no
arrangements of sections;
        ELEM = PARTLOW.SCT;
          OCC = 1;
          LEN = 4;
          INPROC = $INT/ $CALL(LOWIN,INCLOSE)/
$MAX.OCC(1,KEEP.LAST);
          OUTPROC = $INT.OUT;
          COMMENTS = holds part number (start);
        ELEM = PARTHIGH.SCT;
          OCC = 1;
          LEN = 4;
          INPROC = $INT/ $CALL(HIGHIN,INCLOSE)/
$MAX.OCC(1,KEEP.LAST);
          OUTPROC = $INT.OUT;
          COMMENTS = holds part number (finish);
        ELEM = CHAPTER.SCT;
          OCC = 1;
          LEN = 4;
          INPROC = $INT/ $CALL(CHAPIN,INCLOSE)/
$MAX.OCC(1,KEEP.LAST);
          OUTPROC = $INT.OUT;
          COMMENTS = holds chapter of part number;
        ELEM = SOW;
          COMMENTS = given values in first record from FIRST -
TWENTIETH;
        ELEM = OUTNOW;
          COMMENTS = given values in first record from 1 - 20;
    VIRTUAL;
        ELEM = SI;
          INPROC = $INT(4,W);
          OUTPROC = $CALL(V);
          COMMENTS = each component of the symbolic key can be
derived as a virtual element by the appropriate substring
operation on the key value;
        ELEM = YEAR;
          INPROC = $INT(4,W);
          OUTPROC = $CALL(V);
        ELEM = CHAP;
          INPROC = $INT(4,W);
          OUTPROC = $CALL(V);
        ELEM = HEAD;
          INPROC = $INT(4,W);
          OUTPROC = $CALL(V);
        ELEM = SCHED;
          INPROC = $INT(4,W);
          OUTPROC = $CALL(V);
        ELEM = SCHEDHIGH;
          INPROC = $INT(4,W);
```

```
      OUTPROC = $CALL(V);
    ELEM = SCHEDM;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = SUBHIGH;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = SCHEDN;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = SECT;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = SECTHIGH;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = MISC;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = CON;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = DUPL;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = PARTLOW;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = PARTHIGH;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
    ELEM = SUBPART;
      INPROC = $INT(4,W);
      OUTPROC = $CALL(V);
  STRUCTURE = DETAIL;
    COMMENTS = After the extraction of the header line for
the record, each subsequent line is held in the structure
DETAIL for which each occurrence comprises a sequence
number held in SEQ and a data field value for one of the
other fields belonging to this structure;
    FIXED;
      KEY = SEQ;
        OCC = 1;
        LEN = 2;
        INPROC = $INT(2,S);
        OUTPROC = $INT.OUT(4,0);
        COMMENTS = holds the sequence number which ensures
that the lines of the law held within each record are
printed out in the correct order;
    OPTIONAL;
      ELEM = GEN-BOLD-TYPE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = X02-1, 1;
COMMENTS = throughout this definition, the aliases preceded
by X are of the form Xcc-t['T'] where cc indicates the
hexadecimal code for the element on the HMSO tape, t is the
type of record in which the element occurs - 1 act headings,
2 section or paragraph, 3 schedule, 4 ubiquitous, 5 part,
and T occurs if the element is code at the record level. The
short aliases of form n[a] (n is numeric, a is letter) are
referenced in the PASSPROC in the index linkage for INFO;
      ELEM = UPPER-CASE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = X03-1, 2;
      ELEM = ARRANGE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = XF3-1, 3;
      ELEM = PREAMBLE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = XF0-1, 4;
      ELEM = MINOR-COL-HEAD;
        INPROC = $CALL(TRANSHEX);
        ALIASES = OTHER-GEN1, X16-1, 5;
      ELEM = TWO-COLUMNS;
        INPROC = $CALL(TRANSHEX);
        ALIASES = ARRANGE-SECTIONS, OTHER-GEN2, X61-1, 6;
      ELEM = GEN-SUBHEADING;
        INPROC = $CALL(TRANSHEX);
        ALIASES = OTHER-GEN3, X2D-1, 7;
      ELEM = SMALL-REPRO-TXT;
        INPROC = $CALL(TRANSHEX);
        ALIASES = GENUNK, XF2-1, 7A;
      ELEM = REPR-LONG-TITLE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = GENUNK2, XF5-1, 7B;
      ELEM = GENUNK3;
        INPROC = $CALL(TRANSHEX);
        ALIASES = NIL1, XF7-1, 7C;
      ELEM = SMALL-REPRO-MRG;
        INPROC = $CALL(TRANSHEX);
        ALIASES = GENUNK4, XF8-1, 7D;
      ELEM = SMALL-REPRO-COL;
        INPROC = $CALL(TRANSHEX);
        ALIASES = GENUNK5, X18-1, 7E;
      ELEM = MARG-N-REF-ACT;
        INPROC = $CALL(TRANSHEX);
        ALIASES = X0C-4, 8;
      ELEM = CROSSNOTE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = X4B-4, 10;
      ELEM = FOOTNOTE;
        INPROC = $CALL(TRANSHEX);
        ALIASES = X3F-4, 11;
      ELEM = FOOTN-OLD-STAT;
        INPROC = $CALL(TRANSHEX);
        ALIASES = XF9-4, 12;
      ELEM = OMISS;
        INPROC = $CALL(TRANSHEX);
        ALIASES = XF1-4, 13;
      ELEM = OLD-STAT-BANNER;
        INPROC = $CALL(TRANSHEX);
        ALIASES = OLD-STAT-NO-SECT, X7F-4, 14;
        ELEM = TEXT-IN-COL;
          INPROC = $CALL(TRANSHEX);
          ALIASES = ENGLISH, X7A-4, 15;
        ELEM = GEN-UNID;
          INPROC = $CALL(TRANSHEX);
          ALIASES = LATIN, 16;
        ELEM = SCHEDULE-TEXT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X37-3, 17;
        ELEM = CENTRE-HEAD;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X2E-3, 18;
        ELEM = NUMB-PARA;
          INPROC = $CALL(TRANSHEX);
          ALIASES = NIL2, X5A-3, 19;
        ELEM = TEXT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X7F-3, 20;
        ELEM = SINGLE-IND-SUB;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X0B-3, 21;
        ELEM = SINGLE-IND-SUB2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X6C-3, 22;
        ELEM = DOUBLE-IND-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X5B-3, 23;
        ELEM = DOUBLE-IND-SM2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = NIL3, X5D-3, 24;
        ELEM = DOUBLE-IND-SM3;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X7D-3, 25;
        ELEM = DOUBLE-IND-SM4;
          INPROC = $CALL(TRANSHEX);
          ALIASES = NIL4, X4D-3, 26;
        ELEM = TRIPLE-IND;
          INPROC = $CALL(TRANSHEX);
          ALIASES = NIL5, X4E-3, 27;
        ELEM = TRIPLE-IND2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X10-3, 28;
        ELEM = COL-PR-HEAD;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X11-3, 29;
        ELEM = COL-PR-HEAD2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X3D-3, 30;
        ELEM = COL-PR-DOUBLE;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X12-3, 31;
        ELEM = COL-PR-DOUBLE2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X3C-3, 32;
        ELEM = COL-PR-TEXT;
          INPROC = $CALL(TRANSHEX);
```

```
          ALIASES = X13-3, 33;
        ELEM = PRINT-TABULAR;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK, XF0-3, 33A;
          COMMENTS = print in tabular form;
        ELEM = STARRED-NOTE-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK2, XF2-3, 33B;
          COMMENTS = starred note - small type (indicated by
*);
        ELEM = COL-PR-HEAD-UNRL;
          INPROC = $CALL(TRANSHEX);
          ALIASES = ARRANGE-SCHEDULE, SCHEDUNK3, XF3-3, 33C;
          COMMENTS = unruled column headings;
        ELEM = COL-PR-HEAD-REPR;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK4, XF6-3, 33D;
          COMMENTS = heading for reproduced schedule;
        ELEM = COL-PR-HEAD-MED;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK5, X61-3, 33E;
          COMMENTS = column headings in table - larger type;
        ELEM = SIN-IND-SUB-NOTE;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK6, X7B-3, 33F;
          COMMENTS = note - single indent subsequent lines;
        ELEM = COL-PRINT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK7, XT0F00-3, 33G;
        ELEM = FORM-TIT-BOLD-UC;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK8, X2F-3, 33H;
          COMMENTS = form title - bold upper case title style
;
        ELEM = INSTRUCTION;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK9, XF7-3, 33I;
          COMMENTS = instruction (comment);
        ELEM = CARRIAGE-CONTR;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK10, X0D-3, 33J;
        ELEM = COL-PR-HEAD3;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK11, X25-3, 33K;
          COMMENTS = column headings in table (separated by
hex 27A7);
        ELEM = COL-PR-HEAD-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK12, XF4-3, 33L;
          COMMENTS = column headings in table, small text;
        ELEM = DOC-TIT-CNT-ITAL;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK13, X2D-3, 33M;
          COMMENTS = document title - centre italics;
        ELEM = DBL-IND-SUB-NIL2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK14, X50-3, 33N;
          COMMENTS = double indent, going to no indent;
        ELEM = PART-TEXT-IN-SCH;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SCHEDUNK15, X03-3, 330;
          COMMENTS = part text heading within schedule
(error);
        ELEM = SCHED-UNID;
          INPROC = $CALL(TRANSHEX);
          ALIASES = 34;
        ELEM = SECT-MAIN-TEXT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X5A-2, 35;
        ELEM = CONT-TEXT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X7F-2, 36;
        ELEM = INDENT-TEXT-SUB;
          INPROC = $CALL(TRANSHEX);
          ALIASES = NIL6, X25-2, 37;
        ELEM = SINGLE-INDENT-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = XF2-2, 38;
        ELEM = SINGLE-INDENT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X6C-2, 39;
        ELEM = HALF-I-SIN-I-SUB;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SINGLE-INDENT2, X7B-2, 40;
          COMMENTS = first line half indented, subsequent
lines single indent;
        ELEM = DBL-INDENT-LIST;
          INPROC = $CALL(TRANSHEX);
          ALIASES = OTHER-SECTION1, X7D-2, 41;
          COMMENTS = double indented list, proud letter or
number;
        ELEM = DBL-IND-SUB-NIL;
          INPROC = $CALL(TRANSHEX);
          ALIASES = OTHER-SECTION2, X50-2, 42;
        ELEM = NOT-IND-SUB-SIN;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK, XF7-2, 43;
          COMMENTS = first line not indented - subsequent
lines single indent;
        ELEM = REPR-MARG-NOT-SM;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK2, XF8-2, 44;
          COMMENTS = reproduced marginal note in old statute
- small type;
        ELEM = NOTE-EMB-SIN-IND;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK3, X5B-2, 45;
          COMMENTS = embedded note or quote - single indent;
        ELEM = DOUBLE-INDENT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK4, X5C-2, 46;
          COMMENTS = double indent;
        ELEM = DOUBLE-INDENT2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK5, X5D-2, 47;
          COMMENTS = probably double indent;
        ELEM = DBL-IND-FST-MORE;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK6, X4D-2, 48;
          COMMENTS = double indent with top line indented
slightly more;
        ELEM = TRIPLE-IND-LIST;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK7, X60-2, 49;
          COMMENTS = triple indented list;
        ELEM = TRIPLE-IND-LIST2;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK8, X4E-2, 50;
          COMMENTS = triple indented list, proud letter or
number;
        ELEM = MARG-N-ACT-RUFF;
          INPROC = $CALL(TRANSHEX);
          ALIASES = SECTUNK9, XF6-2, 51;
          COMMENTS = marginal note, an act ref to Ruffhead;
        ELEM = SECTUNK11;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X2E-2, 53;
        ELEM = SECTUNK12;
          INPROC = $CALL(TRANSHEX);
          ALIASES = XF5-2, 54;
        ELEM = SECT-UNID;
          INPROC = $CALL(TRANSHEX);
          ALIASES = 55;
        ELEM = PART-TEXT;
          INPROC = $CALL(TRANSHEX);
          ALIASES = X03-5, 56;
        ELEM = SINGLE-IND-SM-P;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK, XF2-5, 56A;
          COMMENTS = assumed same as F2 in section;
        ELEM = REP-MARG-NOT-SMP;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK2, XF8-5, 56B;
          COMMENTS = assumed same as F8 in section;
        ELEM = PART-SUBHEADING;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK3, X2D-5, 56C;
        ELEM = CONT-TEXT-PART;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK7, X7F-5, 56G;
          COMMENTS = assumed same as 7F in section;
        ELEM = NOT-IND-SUB-SINP;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK9, XF7-5, 56I;
          COMMENTS = assumed same as F7 in section;
        ELEM = INDENT-TEXT-SUBP;
          INPROC = $CALL(TRANSHEX);
          ALIASES = PARTUNK11, X25-5, 56K;
```

```
  ELEM = PART-UNID;
   INPROC = $CALL(TRANSHEX);
   ALIASES = 57;
USERDEFS;
 VARIABLE = ACCESS;
  LEN = 252;
  TYPE = STRING;
 VARIABLE = ANSAVE;
  TYPE = DYNAMIC;
 VARIABLE = ANSAVES;
  TYPE = DYNAMIC;
 VARIABLE = CHAR;
  LEN = 1;
  TYPE = STRING;
 VARIABLE = CHARINT;
  LEN = 2;
  TYPE = INT;
 VARIABLE = CHARLAST;
  LEN = 2;
  TYPE = INT;
 VARIABLE = CHARR;
  LEN = 1;
  TYPE = STRING;
 VARIABLE = ENO;
  TYPE = DYNAMIC;
 VARIABLE = HIGH;
  LEN = 3;
  TYPE = STRING;
 VARIABLE = KEY;
  TYPE = DYNAMIC;
 VARIABLE = KEYDUPL;
  TYPE = DYNAMIC;
 VARIABLE = KEYDUPLI;
  LEN = 2;
  TYPE = INT;
 VARIABLE = KEYHEAD;
  TYPE = DYNAMIC;
 VARIABLE = KEYMISC;
  TYPE = DYNAMIC;
 VARIABLE = KEYMISCI;
  LEN = 2;
  TYPE = INT;
 VARIABLE = KEYPART;
  TYPE = DYNAMIC;
 VARIABLE = KEYPARTHIGH;
  TYPE = DYNAMIC;
 VARIABLE = KEYPARTLOW;
  TYPE = DYNAMIC;
 VARIABLE = KEYSCHED;
  TYPE = DYNAMIC;
 VARIABLE = KEYSCHEDHIGH;
  TYPE = DYNAMIC;
 VARIABLE = KEYSCHEDM;
  TYPE = DYNAMIC;
 VARIABLE = KEYSCHEDMI;
  LEN = 2;
```

```
  TYPE = INT;
 VARIABLE = KEYSCHEDMSV;
  TYPE = DYNAMIC;
 VARIABLE = KEYSCHEDN;
  TYPE = DYNAMIC;
 VARIABLE = KEYSECT;
  TYPE = DYNAMIC;
 VARIABLE = KEYSECTHIGH;
  TYPE = DYNAMIC;
 VARIABLE = KEYSECTSAVE;
  TYPE = DYNAMIC;
 VARIABLE = KEYSUBHIGH;
  TYPE = DYNAMIC;
 VARIABLE = KEYSUBPART;
  TYPE = DYNAMIC;
 VARIABLE = M;
  LEN = 2;
  TYPE = INT;
 VARIABLE = MARG;
  TYPE = DYNAMIC;
 VARIABLE = MARGS;
  TYPE = DYNAMIC;
 VARIABLE = NOSECT;
  TYPE = DYNAMIC;
 VARIABLE = N1;
  TYPE = INT;
 VARIABLE = OUTNO;
  OCC = 52;
  TYPE = DYNAMIC;
 VARIABLE = OUTNOW;
  OCC = 20;
  TYPE = DYNAMIC;
 VARIABLE = RESULT;
  LEN = 80;
  TYPE = STRING;
 VARIABLE = ROMANHIGH;
  LEN = 16;
  TYPE = STRING;
 VARIABLE = ROMANLOW;
  LEN = 16;
  TYPE = STRING;
 VARIABLE = ROMSOL;
  LEN = 2;
  TYPE = INT;
 VARIABLE = S;
  LEN = 16384;
  TYPE = DYNAMIC;
 VARIABLE = SCHSOURCE;
  TYPE = DYNAMIC;
 VARIABLE = SIZE;
  LEN = 2;
  TYPE = INT;
 VARIABLE = SO;
  OCC = 52;
  TYPE = DYNAMIC;
 VARIABLE = SOW;
```

```
          OCC = 20;
          TYPE = DYNAMIC;
        VARIABLE = SUBSCH;
          TYPE = DYNAMIC;
        VARIABLE = S1;
          LEN = 16384;
          TYPE = DYNAMIC;
        VARIABLE = S2;
          LEN = 16384;
          TYPE = DYNAMIC;
        VARIABLE = S3;
          LEN = 16384;
          TYPE = DYNAMIC;
        VARIABLE = TEMP;
          LEN = 252;
          TYPE = STRING;
        VARIABLE = TEMPI;
          TYPE = INT;
        VARIABLE = TYPESCH;
          TYPE = DYNAMIC;
        VARIABLE = TYPESUBSCH;
          TYPE = DYNAMIC;
        USERPROC = BUILDKEY;
          UPROC = IF $SIZE($VAL) > 0 : IF #KEYSECT = '000' :
LET KEYSECT = $INSETR($VAL,'0',3);
          UPROC = IF $TYPETEST(#KEYSECT,INT) = '' : LET
KEYSECT = '000';
          UPROC = IF ^$ELSE : LET NOSECT = 'ON';
          UPROC = IF #NOSECT = 'ON' : IF #KEYSECT = '000' :
JUMP MISC;
          UPROC = LET KEYMISC = '00';
          UPROC = JUMP ASSIGN;
          UPROC = ++MISC;
          UPROC = LET KEYMISCI = $INT(#KEYMISC) + 1;
          UPROC = LET KEYMISC = $STRING(#KEYMISCI);
          UPROC = LET KEYMISC = $INSETR(#KEYMISC,'0',2);
          UPROC = LET KEYSECT = #KEYSECTSAVE;
          UPROC = ++ASSIGN;
          UPROC = LET KEYSECTSAVE = #KEYSECT;
          UPROC = IF $MATCH(#ANSAVES,'?Part?') = 0 : JUMP
ASSIGN2;
          UPROC = LET KEYPARTLOW = #KEYSCHEDM;
          UPROC = LET KEYPARTHIGH = #KEYSUBHIGH;
          UPROC = LET KEYSCHEDM = '00';
          UPROC = LET KEYSUBHIGH = '00';
          UPROC = ++ASSIGN2;
          UPROC = LET KEYPART = #KEYPARTLOW #KEYPARTHIGH
#KEYSUBPART;
          UPROC = SET VAL = $LEFTSTR(#KEY,7) #KEYHEAD
#KEYSCHED #KEYSCHEDHIGH #KEYSCHEDM #KEYSUBHIGH #KEYSCHEDN
#KEYPART #KEYSECT #KEYSECTHIGH #KEYMISC '0' '00';
          UPROC = LET KEY = $VAL;
          COMMENTS = builds symbolic key from its components
including the assignment of a subsection or subparagraph
number if necessary;
        USERPROC = CAP;
```

```
          UPROC = SET VAL $CAPITALIZE($VAL);
          COMMENTS = used for capitalizing values;
        USERPROC = CHAPIN;
          UPROC = IF $CURCMD ^= 'ADD' : RETURN;
          UPROC = SET VAL #KEYSUBPART;
          COMMENTS = fetches chapter of part number in build;
        USERPROC = CHECKPARA;
          UPROC = LET TEMP = $VAL;
          UPROC = IF $MATCH(#TEMP,'?-?') = 1 : LET HIGH =
$RIGHTSUB(#TEMP,'-');
          UPROC = IF ^$ELSE : LET TEMP = $LEFTSUB(#TEMP,'-');
          UPROC = IF $TYPETEST(#TEMP,INT) = '' : LET NOSECT =
'ON';
          UPROC = IF ^$ELSE : LET TEMP = '';
          UPROC = IF $SIZE(#TEMP) > 6 : LET NOSECT = 'ON';
          UPROC = IF ^$ELSE : LET TEMP = '';
          UPROC = IF #NOSECT = 'OFF' : IF $SIZE(#HIGH)  > 0 :
LET KEYSECTHIGH = $INSETR(#HIGH,'0',3);
          UPROC = SET VALUE = #TEMP;
          UPROC = IF $SIZE($VAL) > 0 : LET KEYSECT =
$INSETR($VAL,'0',3);
          UPROC = LET TEMP = $RIGHTSTR(#MARG,1);
          UPROC = "LET TEMP = $TRANSLATE(#TEMP,'`[]"" ()','')";
          UPROC = LET TEMP = $CAPITALIZE(#TEMP);
          UPROC = SET VAL = $LEFTSUB(#TEMP,'.');
          UPROC = IF $SIZE($VAL) > 9 : SET VAL '';
          UPROC = LET MARGS = 'ON';
          UPROC = XEQ USERPROC ROMAN;
          COMMENTS = investigates paragraph and section values
and massages them for inclusion in the symbolic key;
        USERPROC = DELETE;
          UPROC = SET DELETE;
          COMMENTS = used to delete some temporary element
values;
        USERPROC = FETCHSUBSCH;
          UPROC = LET TEMP = $VAL;
          UPROC = IF $MATCH($VAL,'?-?') = 1 : LET HIGH =
$RIGHTSUB($VAL,'-');
          UPROC = IF ^$ELSE : LET TEMP = $LEFTSUB($VAL,'-');
          UPROC = IF $TYPETEST(#TEMP,INT) = '' : LET TEMP = '';
          UPROC = IF $SIZE(#TEMP) > 6 : LET TEMP = '';
          UPROC = IF $SIZE(#TEMP) > 0 : IF $SIZE(#HIGH) > 0 :
IF $TYPETEST(#HIGH,INT) = 'INT' : LET KEYSCHEDHIGH =
$INSETR(#HIGH,'0',3);
          UPROC = LET KEYSCHED = $INSETR(#TEMP,'0',3);
          UPROC = IF #TEMP = '' : LET KEYSCHED = '000';
          UPROC = SET VAL #ANSAVE;
          UPROC = LET SUBSCH = 'ON';
          UPROC = LET M = $MATCH($VAL,'?Subsch?');
          UPROC = IF #M = 0 : JUMP END1;
          UPROC = LET TEMP = $RIGHTSUB($VAL,'Subsch ');
          UPROC = LET TEMP = $LEFTSUB(#TEMP,',');
          UPROC = LET TEMP = $TRANSLATE(#TEMP,'()., ','');
          UPROC = IF #SCHSOURCE = 'NOTE' : LET KEYSCHEDN = '1';
          UPROC = SET VALUE = #TEMP;
          UPROC = RETURN;
        UPROC = ++END1;
        UPROC = SET VALUE = '';
        COMMENTS = converts the processed schedule values
found into symbolic key form and identifies the subschedule
value;
      USERPROC = HIGHIN;
        UPROC = IF $CURCMD ^= 'ADD' : RETURN;
        UPROC = SET VAL #KEYPARTHIGH;
        COMMENTS = fetches the value for part number
(finish) during build;
      USERPROC = HRENT;
        UPROC = LET M = $MATCH($VAL,'?h?');
        UPROC = IF #M = 0 : RETURN;
        UPROC = LET S = $RIGHTSUB($VAL,h);
        UPROC = LET S2 = $LEFTSUB($VAL,h);
        UPROC = LET S1 = $LEFTSTR(#S,1);
        UPROC = LET M =
$MATCH(#S1,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y
,Z);
        UPROC = IF #M = 0 : RETURN;
        UPROC = LET S3 = #S2 ' ' #S;
        UPROC = SET VALUE = #S3;
        COMMENTS = used in index building for stripping the
leading extraneous h which occurs in some values for
typesetting purposes;
      USERPROC = IDPARA;
        UPROC = LET TEMP = $VAL;
        UPROC = IF $MATCH(#TEMP,'?-?') = 1 : LET HIGH =
$RIGHTSUB(#TEMP,'-');
        UPROC = IF ^$ELSE : LET TEMP = $LEFTSUB(#TEMP,'-');
        UPROC = IF $TYPETEST(#TEMP,INT) = '' : LET NOSECT =
'ON';
        UPROC = IF ^$ELSE : LET TEMP = '';
        UPROC = IF $SIZE(#TEMP) > 6 : LET NOSECT = 'ON';
        UPROC = IF ^$ELSE : LET TEMP = '';
        UPROC = IF #NOSECT = 'OFF' : IF $SIZE(#HIGH)  > 0 :
IF $TYPETEST(#HIGH,INT) = 'INT' : LET KEYSECTHIGH =
$INSETR(#HIGH,'0',3);
        UPROC = SET VALUE = #TEMP;
        UPROC = IF $VAL = '' : LET KEYSECT = '000';
        UPROC = IF $SIZE($VAL) > 0 : LET KEYSECT =
$INSETR($VAL,'0',3);
        UPROC = SET VAL #ANSAVE;
        UPROC = LET M = $MATCH($VAL,'?Para?');
        UPROC = IF #M = 0 : JUMP END3;
        UPROC = LET M = $MATCH($VAL,'?Para ,?');
        UPROC = IF #M = 1 : JUMP END2;
        UPROC = LET M = $MATCH($VAL,'?Para  ,?');
        UPROC = IF #M = 1 : JUMP END2;
        UPROC = LET TEMP = $RIGHTSUB($VAL,',Para');
        UPROC = LET TEMP = $LEFTSUB(#TEMP,',*');
        UPROC = "LET TEMP = $CHANGE(#TEMP,''D','')";
        UPROC = LET TEMP = $CHANGE(#TEMP,'I, A','I');
        UPROC = IF $MATCH(#TEMP,'?]?') = 1 : LET TEMP =
$RIGHTSUB(#TEMP,']');
        UPROC = "LET TEMP = $TRANSLATE(#TEMP,''[]"" ()','')";
        UPROC = SET VAL $CAPITALIZE(#TEMP);
        UPROC = XEQ USERPROC ROMAN;
        UPROC = RETURN;
        UPROC = ++END2;
        UPROC = SET VALUE '';
        UPROC = LET NOSECT = 'ON';
        UPROC = RETURN;
        UPROC = ++END3;
        UPROC = SET VALUE '';
        COMMENTS = identifies the paragraph values, converts
extraneous characters to null and roman values to arabic;
      USERPROC = IDRECNO;
        UPROC = LET ACCESS = $GETUVAL(AN,0,0);
        UPROC = IF #ACCESS = 0 THEN JUMP END2;
        UPROC = LET M = $MATCH(#ACCESS,'?*?');
        UPROC = IF #M = 0 : JUMP END2;
        UPROC = LET TEMP = $RIGHTSUB(#ACCESS,'*');
        UPROC = LET M = $MATCH(#TEMP,'?*?');
        UPROC = IF #M = 0 : JUMP OK;
        UPROC = LET TEMP = $RIGHTSUB(#TEMP,'*');
        UPROC = ++OK;
        UPROC = SET VALUE = #TEMP;
        UPROC = RETURN;
        UPROC = ++END2;
        UPROC = SET VALUE = '';
        COMMENTS = can be used for deriving the slot number
from the field ACCESS-NO;
      USERPROC = IDSCH;
        UPROC = IF $CURCMD ^= 'ADD' : SET LASTRULE;
        UPROC = IF ^$ELSE : RETURN;
        UPROC = IF #KEYHEAD ^= '0' : SET VAL #ANSAVE;
        UPROC = IF ^$ELSE : JUMP ON;
        UPROC = SET LASTRULE;
        UPROC = LET KEYSECTSAVE = '000';
        UPROC = LET KEYMISC = '00';
        UPROC = RETURN;
        UPROC = ++ON;
        UPROC = IF $MATCH($VAL,'?Sch ,?') = 1 : JUMP END2;
        UPROC = LET M = $MATCH($VAL,'?Sch?');
        UPROC = IF #M = 0 : JUMP END1;
        UPROC = LET TEMP = $RIGHTSUB($VAL,',Sch');
        UPROC = LET TEMP = $LEFTSUB(#TEMP,',');
        UPROC = LET TEMP = $TRANSLATE(#TEMP,'()., ','');
        UPROC = SET VALUE = #TEMP;
        UPROC = RETURN;
        UPROC = ++END1;
        UPROC = SET VALUE = '';
        UPROC = RETURN;
        UPROC = ++END2;
        UPROC = SET VAL '1';
        COMMENTS = identifies the schedule part of the
identifier;
      USERPROC = IDSECT;
        UPROC = SET VAL #ANSAVE;
        UPROC = LET M = $MATCH($VAL,'?,S ?');
        UPROC = IF #M = 0 : JUMP END1;
        UPROC = LET M = $MATCH($VAL,'?S ,?');
        UPROC = IF #M = 1 : JUMP END2;
        UPROC = LET M = $MATCH($VAL,'?S  ,?');
        UPROC = IF #M = 1 : JUMP END2;
        UPROC = LET TEMP = $RIGHTSUB($VAL,',S ');
        UPROC = LET TEMP = $LEFTSUB(#TEMP,',*');
        UPROC = "LET TEMP = $CHANGE(#TEMP,''D','')";
        UPROC = LET TEMP = $CHANGE(#TEMP,'I, A','I');
        UPROC = IF $MATCH(#TEMP,'?]?') = 1 : LET TEMP =
$RIGHTSUB(#TEMP,']');
        UPROC = "LET TEMP = $TRANSLATE(#TEMP,''[]"" ()','')";
        UPROC = SET VAL $CAPITALIZE(#TEMP);
        UPROC = XEQ USERPROC ROMAN;
        UPROC = RETURN;
        UPROC = ++END2;
        UPROC = SET VALUE = '';
        UPROC = LET NOSECT = 'ON';
        UPROC = RETURN;
        UPROC = ++END1;
        UPROC = LET M = $MATCH($VAL,'?,0?');
        UPROC = IF #M = 0 : JUMP END3;
        UPROC = SET VALUE = '0';
        UPROC = RETURN;
        UPROC = ++END3;
        UPROC = SET VALUE = '';
        COMMENTS = identifies the section component of the
identifier, converts extraneous characters to '', and roman
values to arabic;
      USERPROC = INITIALISE;
        UPROC = IF $CURCMD ^= 'ADD' : RETURN;
        UPROC = IF $GETUVAL(SO,0,'111') = '111' : JUMP GOMAIN
;
        UPROC = LET KEYDUPL = $GETUVAL(KEYDUPLSTART,0);
        UPROC = ++LOOP;
        UPROC = LET SO::#N1 = $GETUVAL(SO,#N1);
        UPROC = LET OUTNO::#N1 = $GETUVAL(OUTNO,#N1);
        UPROC = LET N1 = #N1 + 1;
        UPROC = IF #N1 > 51 : JUMP ON;
        UPROC = JUMP LOOP;
        UPROC = ++ON;
        UPROC = LET N1 = 0;
        UPROC = ++LOOP2;
        UPROC = LET SOW::#N1 = $GETUVAL(SOW,#N1);
        UPROC = LET OUTNOW::#N1 = $GETUVAL(OUTNOW,#N1);
        UPROC = LET N1 = #N1 + 1;
        UPROC = IF #N1 > 19 : JUMP GOMAIN;
        UPROC = JUMP LOOP2;
        UPROC = ++GOMAIN;
        UPROC = LET MARGS = 'OFF';
        UPROC = LET SUBSCH = 'OFF';
        UPROC = LET KEYSCHEDM = '00';
        UPROC = LET KEYSECTHIGH = '000';
        UPROC = LET KEYSCHEDHIGH = '000';
        UPROC = LET KEYSUBHIGH = '00';
        UPROC = LET KEYPARTLOW = '00';
        UPROC = LET KEYPARTHIGH = '00';
```

```
        UPROC = LET KEYSCHEDN = '0';
        UPROC = LET KEYSCHED = '';
        UPROC = LET KEYSECT = '';
        UPROC = LET KEYSUBPART = '00';
        UPROC = LET NOSECT = 'OFF';
        UPROC = LET MARG = $GETUVAL(MARG,0);
        UPROC = LET ANSAVE = $GETUVAL(AN,0);
        UPROC = * #ANSAVE;
        UPROC = LET ANSAVES = #ANSAVE;
        UPROC = LET ANSAVE =
$CHANGE(#ANSAVE,',Part',',Subsch');
        UPROC = LET KEY = $GETUVAL(KEY,0);
        UPROC = LET KEYHEAD = $GETUVAL(KEYHEAD,0);
        UPROC = LET TYPESCH = $GETUVAL(TYPESCH,0);
        UPROC = LET TYPESUBSCH = $GETUVAL(TYPESUBSCH,0);
        UPROC = LET SCHSOURCE = $GETUVAL(SCHSOURCE,0);
        UPROC = SET VAL #KEY;
        UPROC = IF #KEY = '' : * 'key is null';
        COMMENTS = if command is not ADD, no key processing
is done. Otherwise, this procedure initiates the building
of the symbolic key, and builds from the first record the
arrays to perform concversions of letters and words to
numerals;
        COMMENTS = VARIABLE defines either static variables
 whose values are transient or dynamic variables whose
values are permanent;
      USERPROC = ISARRANGE;
        UPROC = LET TEMP = $GETUVAL(ARRANGE,0,'NONE');
        UPROC = IF #TEMP ^= 'NONE' : JUMP END2;
        UPROC = RETURN;
        UPROC = ++END2;
        UPROC = SET VALUE 'THE';
      USERPROC = LOWIN;
        UPROC = IF $CURCMD ^= 'ADD' : RETURN;
        UPROC = SET VAL #KEYPARTLOW;
        COMMENTS = fetches the value for part number (start)
during build;
      USERPROC = ROMAN;
        UPROC = LET ROMSOL = 0;
        UPROC = LET TEMP = $VAL;
        UPROC = IF $MATCH(#TEMP,'?I?','?V?','?X?') > 0 : IF
$SIZE(#TEMP) < 9 : IF $STRIP(#TEMP,'IVX ') = '' : JUMP
OKROMAN;
        UPROC = IF $ELSE : RETURN;
        UPROC = ++OKROMAN;
        UPROC = LET SIZE = $SIZE(#TEMP) - 1;
        UPROC = LET CHARLAST = 0;
        UPROC = ++LOOP;
        UPROC = LET CHARR = $RIGHTSTR(#TEMP,#SIZE);
        UPROC = LET CHARR = $TRANSLATE(#CHARR,'IVX','15A');
        UPROC = LET CHARINT = $CHANGE(#CHARR,'A','10');
        UPROC = IF $SIZE(#TEMP) > 1 : IF #CHARINT <
#CHARLAST : LET CHARINT = 0 - #CHARINT;
        UPROC = LET ROMSOL = #ROMSOL + #CHARINT;
        UPROC = LET CHARLAST = #CHARINT;
        UPROC = LET SIZE = #SIZE - 1;
```

```
       UPROC = IF #SIZE > - 1 : JUMP LOOP;
       UPROC = SET VAL #ROMSOL;
       COMMENTS = converts roman numerals to arabic
numerals in the range 1 to 39;
     USERPROC = STORESUBSCH;
       UPROC = IF #KEYSCHEDN = '1' : JUMP ON;
       UPROC = IF #SCHSOURCE = 'SCHEDULE' : LET KEYSCHEDMSV
= '00';
       UPROC = LET TEMP = $VAL;
       UPROC = IF $MATCH($VAL,'?-?') = 1 : LET HIGH =
$RIGHTSUB($VAL,'-');
       UPROC = IF ^$ELSE : LET TEMP = $LEFTSUB($VAL,'-');
       UPROC = IF $TYPETEST(#TEMP,INT) = '' : LET TEMP = '';
       UPROC = IF $SIZE(#TEMP) > 6 : LET TEMP = '';
       UPROC = IF $SIZE($VAL) > 0 : IF $SIZE(#TEMP) = 0 :
IF $MATCH(#ANSAVE,'?,S ?','?,Para?') = 0 : JUMP ASSIG;
       UPROC = IF $SIZE($VAL) > 0 : IF $SIZE(#TEMP) = 0 :
IF $MATCH(#ANSAVE,'?,S ?','?,Para?') > 0 : JUMP ASSIG2;
       UPROC = JUMP C;
       UPROC = ++ASSIG;
       UPROC = LET KEYSCHEDMI = $INT(#KEYSCHEDMSV);
       UPROC = LET KEYSCHEDMI = #KEYSCHEDMI + 1;
       UPROC = LET KEYSCHEDM = $STRING(#KEYSCHEDMI);
       UPROC = LET KEYSCHEDM = $INSETR(#KEYSCHEDM,'0',2);
       UPROC = LET KEYSCHEDMSV = #KEYSCHEDM;
       UPROC = JUMP ON;
       UPROC = ++ASSIG2;
       UPROC = LET KEYSCHEDM = $INSETR(#KEYSCHEDMSV,'0',2);
       UPROC = JUMP ON;
       UPROC = ++C;
       UPROC = IF $SIZE(#TEMP) > 0 : IF $SIZE(#HIGH) > 0 :
IF $TYPETEST(#HIGH,INT) = 'INT' : LET KEYSUBHIGH =
$INSETR(#HIGH,'0',2);
       UPROC = LET KEYSCHEDM = $INSETR(#TEMP,'0',2);
       UPROC = IF #TEMP = '' : LET KEYSCHEDM = '00';
       UPROC = LET KEYSCHEDMSV = #KEYSCHEDM;
       UPROC = ++ON;
       UPROC = SET VAL #ANSAVE;
       UPROC = LET M = $MATCH($VAL,'?,Chapter ?');
       UPROC = IF #M = 0 : RETURN;
       UPROC = LET TEMP = $RIGHTSUB($VAL,',Chapter ');
       UPROC = SET VAL = $LEFTSUB(#TEMP,',');
       UPROC = "SET VAL = $TRANSLATE($VAL,'`[] ()','')";
       UPROC = XEQ USERPROC ROMAN;
       UPROC = IF $SIZE($VAL) > 0 : IF $TYPETEST($VAL,INT)
= 'INT' : LET KEYSUBPART = $INSETR($VAL,'0',2);
       COMMENTS = identifies the part, subschedule, and
chapter of part identifiers for inclusion in the primary
key, converts extraneous characters to null, and converts
roman values to arabic;
     USERPROC = TESTREC;
       UPROC = IF $SIZE($VAL) > 2 : RETURN;
       UPROC = * 'duplicated record key about to be added';
       UPROC = LET KEYDUPLI = 1 + $INT(#KEYDUPL);
       UPROC = LET KEYDUPL = $STRING(#KEYDUPLI);
       UPROC = LET KEYDUPL = $INSETR(#KEYDUPL,'0',2);
```

```
       UPROC = SET VAL $LEFTSTR(#KEY,34) #KEYDUPL;
       COMMENTS = checks whether the search for a record
with an identical key to that about to be added has
succeeded. If it has, the duplication number is increased
by one and the key thus slightly altered, so that addition
can be made;
     USERPROC = TRANSHEX;
       UPROC = "SET VAL $TRANSLATE($VAL,'
       COMMENTS = converts to null, two non-printing
characters;
     USERPROC = TRANSKEY;
       UPROC = LET TEMP = $VAL;
       UPROC = "LET TEMP = $TRANSLATE(#TEMP,'`[]"" ()','')";
       UPROC = SET VAL #TEMP;
       UPROC = IF #SUBSCH = 'ON' : LET TYPESCH = #TYPESUBSCH
;
       UPROC = IF #TYPESCH = 'WORDS' : JUMP WORDWORK;
       UPROC = IF #TYPESCH = 'ROMAN' : JUMP ROMAN;
       UPROC = IF #TYPESCH ^= 'LETTER' : RETURN;
       UPROC = JUMP LOOP;
       UPROC = ++ROMAN;
       UPROC = IF $MATCH($VAL,'?-?') = 0 : JUMP SIMPLE;
       UPROC = LET ROMANLOW = $LEFTSUB($VAL,'-');
       UPROC = LET ROMANHIGH = $RIGHTSUB($VAL,'-');
       UPROC = SET VAL #ROMANLOW;
       UPROC = XEQ USERPROC ROMAN;
       UPROC = LET ROMANLOW = $VAL;
       UPROC = SET VAL #ROMANHIGH;
       UPROC = XEQ USERPROC ROMAN;
       UPROC = LET ROMANHIGH = $VAL;
       UPROC = SET VAL #ROMANLOW '-' #ROMANHIGH;
       UPROC = * $VAL;
       UPROC = RETURN;
       UPROC = ++SIMPLE;
       UPROC = XEQ USERPROC ROMAN;
       UPROC = RETURN;
       UPROC = ++LOOP;
       UPROC = LET TEMP =
$CHANGE(#TEMP,#SO::#N1,#OUTNO::#N1);
       UPROC = LET N1 = #N1 + 1;
       UPROC = IF #N1 > 51 : JUMP ON2;
       UPROC = JUMP LOOP;
       UPROC = ++WORDWORK;
       UPROC = LET TEMP = $CHANGE(#TEMP,'TO','-');
       UPROC = LET N1 = 0;
       UPROC = ++LOOP2;
       UPROC = LET TEMP =
$CHANGE(#TEMP,#SOW::#N1,#OUTNOW::#N1);
       UPROC = LET N1 = #N1 + 1;
       UPROC = IF #N1 > 19 : JUMP ON2;
       UPROC = JUMP LOOP2;
       UPROC = ++ON2;
       UPROC = SET VAL #TEMP;
COMMENTS = takes the type of the schedule or subschedule
value (WORDS e.g.  FIRST, ROMAN e.g.  I, or LETTER e.g.  A),
and converts them to arabic numerals.  The array variables
```

```
SO and OUTNO are built at the commencement of each build to
hold the letters A-Z,AA-ZZ, and the numbers 1-52
respectively.  The array variables SOW and OUTNOW are also
built then to hold the strings FIRST - TWENTIETH and the
numbers 1-20 respectively.  If the type is '', then the
value is already an arabic numeral and no action is taken;
      USERPROC = V;
        UPROC = LET TEMP = $GETUVAL(ID,0);
        UPROC = IF $VELEM = 'SI' : SET VAL $SUBSTR(#TEMP,0,1)
;
        UPROC = IF $VELEM = 'YEAR' : SET VAL
$SUBSTR(#TEMP,1,3);
        UPROC = IF $VELEM = 'CHAP' : SET VAL
$SUBSTR(#TEMP,4,3);
        UPROC = IF $VELEM = 'HEAD' : SET VAL
$SUBSTR(#TEMP,7,1);
        UPROC = IF $VELEM = 'SCHED' : SET VAL
$SUBSTR(#TEMP,8,3);
        UPROC = IF $VELEM = 'SCHEDHIGH' : SET VAL
$SUBSTR(#TEMP,11,3);
        UPROC = IF $VELEM = 'SCHEDM' : SET VAL
$SUBSTR(#TEMP,14,2);
        UPROC = IF $VELEM = 'SUBHIGH' : SET VAL
$SUBSTR(#TEMP,16,2);
        UPROC = IF $VELEM = 'SCHEDN' : SET VAL
$SUBSTR(#TEMP,18,1);
        UPROC = IF $VELEM = 'PARTLOW' : SET VAL
$SUBSTR(#TEMP,19,2);
        UPROC = IF $VELEM = 'PARTHIGH' : SET VAL
$SUBSTR(#TEMP,21,2);
        UPROC = IF $VELEM = 'SUBPART' : SET VAL
$SUBSTR(#TEMP,23,2);
        UPROC = IF $VELEM = 'SECT' : SET VAL
$SUBSTR(#TEMP,25,3);
        UPROC = IF $VELEM = 'SECTHIGH' : SET VAL
$SUBSTR(#TEMP,28,3);
        UPROC = IF $VELEM = 'MISC' : SET VAL
$SUBSTR(#TEMP,31,2);
        UPROC = IF $VELEM = 'CON' : SET VAL
$SUBSTR(#TEMP,33,1);
        UPROC = IF $VELEM = 'DUPL' : SET VAL
$SUBSTR(#TEMP,34,2);
        COMMENTS = used to derive virtual elements by
appropriate substring processing on symbolic key;
 RECORD-NAME = REC02;
   COMBINE = REC01;
   COMMENTS = defines index-file to hold keywords for
general keyword searching;
   REQUIRED;
     KEY = INFO;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
       COMMENTS = defines POINTER element to hold
inverted-file addresses;
 RECORD-NAME = REC03;
```

```
   COMBINE = REC01;
   COMMENTS = defines index-file to hold date values;
   FIXED;
     KEY = DATE;
       LEN = 4;
       INPROC = $DATE;
       OUTPROC = $DATE.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC04;
   COMBINE = REC01;
   COMMENTS = defines index-file to hold information from
elements MINOR-COL-HEAD, TWO-COLUMNS, GEN-SUBHEADING,
UPPER-CASE, PART-TEXT, COL-PR-HEAD-UNRL, COL-PR-HEAD-MED,
DOC-TIT-CNT-ITAL, PART-TEXT-IN-SCH, PART-SUBHEADING, to
allow searches to be made on mainly headings;
   REQUIRED;
     KEY = BROWSE;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC05;
   COMBINE = REC01;
   COMMENTS = defines index-file for SCHEDULE values;
   FIXED;
     KEY = SCHEDULE;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC06;
   COMBINE = REC01;
   COMMENTS = defines index-file for PARAGRAPH values;
   FIXED;
     KEY = PARAGRAPH;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC07;
   COMBINE = REC01;
   COMMENTS = defines index-file for ACT-TITLE values;
   REQUIRED;
     KEY = ACT-TITLE;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC08;
   COMBINE = REC01;
   COMMENTS = defines index-file for RECNO (slot) values;
   FIXED;
```

```
    KEY = RECNO;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC10;
  COMBINE = REC01;
  COMMENTS = defines index-file for CHAPTER values;
  FIXED;
    KEY = CHAPTER;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC11;
  COMBINE = REC01;
  COMMENTS = defines index-file for HEAD values;
  FIXED;
    KEY = HEAD;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC12;
  COMBINE = REC01;
  COMMENTS = defines index-file for MARG-N-OTHER (marginal
note) values, held as words;
  REQUIRED;
    KEY = MARG;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC13;
  COMBINE = REC01;
  REQUIRED;
    KEY = SUBJECT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
    ELEM = NT;
      THESAURUS;
 RECORD-NAME = REC14;
  COMBINE = REC01;
  COMMENTS = defines index-file for SI values;
  FIXED;
    KEY = SI;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
```

```
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC15;
  COMBINE = REC01;
  COMMENTS = defines index-file for YEAR (year of act)
values;
  FIXED;
    KEY = YEAR;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC16;
  COMBINE = REC01;
  COMMENTS = defines index-file for SCHEDM (subschedule
start) values;
  FIXED;
    KEY = SCHEDM;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC17;
  COMBINE = REC01;
  COMMENTS = defines index-file for MISC (subsection
number) values;
  FIXED;
    KEY = MISC;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC18;
  COMBINE = REC01;
  COMMENTS = defines index-file for CON (continuation
marker) values;
  FIXED;
    KEY = CON;
      LEN = 4;
      INPROC = $INT;
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
 RECORD-NAME = REC19;
  COMBINE = REC01;
  COMMENTS = defines index-file for SCHEDHIGH (schedule
number, finish) values;
  FIXED;
    KEY = SCHEDHIGH;
```

```
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC20;
   COMBINE = REC01;
   COMMENTS = defines index-file for SUBHIGH (subschedule
number, finish) values;
   FIXED;
     KEY = SUBHIGH;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC21;
   COMBINE = REC01;
   COMMENTS = defines index-file for SECTHIGH (section or
paragraph number, finish) values;
   FIXED;
     KEY = SECTHIGH;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC22;
   COMBINE = REC01;
   COMMENTS = defines index-file for SCHEDN (schedule note
marker) values;
   FIXED;
     KEY = SCHEDN;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC23;
   COMBINE = REC01;
   COMMENTS = defines index-file for TAPE values;
   FIXED;
     KEY = TAPE;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC24;
   COMBINE = REC01;
   COMMENTS = defines index-file for SUBPART (chapter to
part number) values;
   FIXED;
     KEY = SUBPART;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC25;
   COMBINE = REC01;
   COMMENTS = defines index-file for DUPL (duplication
number) values;
   FIXED;
     KEY = DUPL;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC26;
   COMBINE = REC01;
   COMMENTS = defines index-file for PARTLOW (part number,
start) values;
   FIXED;
     KEY = PARTLOW;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC27;
   COMBINE = REC01;
   COMMENTS = defines index-file for PARTHIGH (part
number,finish) values;
   FIXED;
     KEY = PARTHIGH;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC28;
   COMBINE = REC01;
   COMMENTS = defines index-file for KIND values;
   REQUIRED;
     KEY = KIND;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC29;
   COMBINE = REC01;
   COMMENTS = defines index-file for TYPESCH values;
   REQUIRED;
     KEY = TYPESCH;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC30;
   COMBINE = REC01;
   COMMENTS = defines index-file for TYPESUBSCH values;
   REQUIRED;
     KEY = TYPESUBSCH;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = REC40;
   COMMENTS = defines record-type keyed on YCS
(year+chapter+section,start) to enable PARTSPEC
(part,start+part,finish+chapter) to be determined, given
the value for YCS;
   FIXED;
     KEY = YEARCHAPSECT;
       OCC = 1;
       LEN = 9;
       ALIASES = YCS;
     ELEM = PARTSPEC;
       OCC = 1;
       LEN = 6;
   OPTIONAL;
     ELEM = DUMMY;
 RECORD-NAME = ZCASE;
   COMMENTS = defines record-type for data on CASES, keyed
on system-generated slot number;
   REMOVED;
   SLOT;
   FIXED;
     ELEM = RECNO;
       OCC = 1;
       LEN = 4;
       INPROC = $INT;
       OUTPROC = $INT.OUT;
       COMMENTS = sequence number for CASES, starts at
100001;
   REQUIRED;
     ELEM = ACCESS-NO;
       OCC = 1;
       INPROC = AW22:1,252/$SQU /$TRIM;
       ALIASES = AN, SN;
     ELEM = CASE-TITLE;
       OCC = 1;
       INPROC = $SQU /$TRIM;
       ALIASES = AT, TITLE;
     ELEM = KIND;
       OCC = 1;
       COMMENTS = always has a value of '5' for cases;
     ELEM = DATE;
       OCC = 1;
       LEN = 4;
```

```
       INPROC = A48,'NO DATE','2100'/AW31:0/A123,'NO DATE',1
;
       OUTPROC = A76:1/A48,'2100','NO DATE';
   OPTIONAL;
     ELEM = GEN-BOLD-TYPE;
       ALIASES = X02;
     ELEM = CASE-MAIN-TEXT;
       ALIASES = X5A-2;
     ELEM = CASE-UNID;
     ELEM = GEN-UNID;
     ELEM = OMISS;
   USERDEFS;
     USERPROC = CAP;
       UPROC = SET VAL $CAPITALIZE($VAL);
 RECORD-NAME = ZCASE2;
   COMMENTS = defines index-file for INFO, holding words
contained in the texts;
   REQUIRED;
     KEY = INFO;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = ZCASE3;
   COMMENTS = defines index-file for DATE values;
   FIXED;
     KEY = DATE;
       LEN = 4;
       INPROC = $DATE;
       OUTPROC = $DATE.OUT;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
 RECORD-NAME = ZSCH01;
   COMMENTS = defines record-type keyed on YCSS
(year+chapter+schedule+paragraph) which enables SUBSCHSPEC
(subschedule,start+subschedule,finish) to be determined,
given a value for YCSS;
   FIXED;
     KEY = YEARCHAPSCHPARA;
       OCC = 1;
       LEN = 12;
       ALIASES = YCSS;
     ELEM = SUBSCHSPEC;
       OCC = 1;
       LEN = 4;
   OPTIONAL;
     ELEM = DUMMY;
 RECORD-NAME = ZTRACK;
   COMMENTS = defines a structure for holding the status of
each user in the GET protocol.  The key is the user's id
and each user (except LAU1, LAU2, CL22) can only access his
own information;
   FIXED;
     KEY = ACCOUNT;
       OCC = 1;
       LEN = 4;
```

Appendix I: SPIRES FILEDEF for Statute Law

```
      INPROC = $CAP/ $TRIM/ $SQU/ $TEST.ACCT;
  OPTIONAL;
    ELEM = FOUND.REF;
      OCC = 1;
      LEN = 36;
      ALIASES = FR, 1;
      COMMENTS = id of reference found in FIND command;
    ELEM = STRUC1;
      TYPE = STR;
    ELEM = REF0.FLAG;
      OCC = 1;
      LEN = 1;
  STRUCTURE = STRUC1;
    REQUIRED;
      KEY = REF1;
        OCC = 1;
    OPTIONAL;
      ELEM = REF1.ID;
        OCC = 1;
        LEN = 36;
        ALIASES = 2;
        COMMENTS = id of reference referred to by FOUND.REF
;
      ELEM = REF1.USER;
        OCC = 1;
        COMMENTS = id of reference sought as supplied by
user;
      ELEM = REF1.FLAG;
        OCC = 1;
        LEN = 1;
        COMMENTS = set to 1 if this is current user
position, otherwise no value as all are deleted when 1 is
assigned to any part of the record;
      ELEM = STRUC2;
        TYPE = STR;
  STRUCTURE = STRUC2;
    REQUIRED;
      KEY = REF2;
        OCC = 1;
      ELEM = REF2.ID;
        OCC = 1;
        LEN = 36;
        ALIASES = 3;
      ELEM = REF2.USER;
        OCC = 1;
      ELEM = REF2.FLAG;
        OCC = 1;
        LEN = 1;
    OPTIONAL;
      ELEM = STRUC3;
        TYPE = STR;
  STRUCTURE = STRUC3;
    OPTIONAL;
      ELEM = STRUC4;
        COMMENTS = add structure, etc;
 GOALREC-NAME = REC01;
```

```
    PTR-ELEM = POINTER;
      EXTERNAL-NAME = ITEM;
        PASSPROC = A170;
    INDEX-NAME = REC02;
      SEARCHTERMS = WORD, W,INFO;
        SRCPROC = $CAP/$SQU /$TRIM
/$BREAK/$TEST.LEN(EX.LE,2)/
$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR,AS,BE,FOR,BY,SHALL,IS,ON,
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')/$TRUNC.SRC('?');
        PASSPROC = "A167:2,SCHEDULE-TEXT,SECT-MAIN-TEXT,TEXT-
IN-COL,TEXT,
SINGLE-INDENT2,
MARG-N-REF-ACT,SECTUNK,SINGLE-INDENT-SM,SECTUNK9,SECTUNK12,1
,4,7A,7B,7C,7D,7E,
8,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,2
9,30,31,32,33,33A,
33D,33F,33G,33H,33I,33J,33K,33L,33N,34,35,36,37,38,39,40,41,
42,43,44,45,46,47,
48,49,50,51,53,54,55,56A,56B,56G,56I,57,
56K,GEN-BOLD-TYPE,SECTUNK11,SECTUNK3 / A48,'.',' ',',','
',':',' ',)',' ','(',' ',';',' ','-',' ','*',' ','""',' 
','''',' '/
$BREAK/$TEST.LEN(EX.LE,2)/$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR
,AS,BE,FOR,BY,SHALL,IS,ON,
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')";
      PTR-GROUP = POINTER;
      COMMENTS = defines linkage for INFO index, index is
made by taking values for all text elements, converting
punctuation to spaces, breaking on spaces, rejecting any
word <= 2 characters long or which is in the stop list.
Truncated searches are allowed using ? as trailing character
;
    INDEX-NAME = REC03;
      SEARCHTERMS = DATE,D;
        SRCPROC = $DATE(TRUNC,,W);
        GOALREC-ELEM = DATE;
        PASSPROC = $PASS.ELEM('DATE',BINARY);
      PTR-GROUP = POINTER;
      COMMENTS = linkage for DATE allows truncated date
searches to be made automatically, thus search for 1976
will find 28 aug 1976;
    INDEX-NAME = REC04;
      SEARCHTERMS = BROWSE,BROW;
        SRCPROC = $CAP/$SQU /$TRIM /$BREAK/
$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR,AS,BE,
FOR,BY,SHALL,IS,ON,ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT
,SUCH,AN,IF,MAY,
WHERE,A,SUBSECTION,PARAGRAPH,ABOVE,WITH,IT')/$TRUNC.SRC('?')
;
        PASSPROC = "$PASS.ELEM('OTHER-GEN1,OTHER-GEN2,OTHER-G
EN3,UPPER-CASE,
ARRANGE,PART-TEXT,33C,33E,33M,33O,56,56C',BREAK.NUM)/
```

Appendix I: SPIRES FILEDEF for Statute Law

```
$CALL(HRENT) /$CALL(CAP) /A48,'.','','.',' ',':',' ',),'
','(',' ',';',' ','-',' ',' ','*',' ',' ','""',' ',' ',':',' '
/$BREAK/$TEST.LEN(EX.LE,2)/$EXCLUDE
('THE,OF,TO,IN,THAT,AND,OR,AS,BE,FOR,BY,SHALL,IS,ON,
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')";
     PTR-GROUP = POINTER;
     COMMENTS = linkage for BROWSE index, index is made by
fetching values of specified elements, passing them through
procedure HRENT to remove extraneous leading 'h' used for
typesetting purposes occasionally, breaking on spaces,
removing values <= 2 characters long or in stop list;
   INDEX-NAME = REC05;
     SEARCHTERMS = SCHEDULE,SCH, SCHED;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SCHED',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC06;
     SEARCHTERMS = PARAGRAPH,PARA,P,SECTION,SECT;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SECT',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC07;
     SEARCHTERMS = ACT-TITLE,A;
       SRCPROC = $CAP/$BREAK/$TRUNC.SRC('?');
       PASSPROC = $PASS.ELEM('ACT-TITLE',WORD)/$BREAK/$NULL;
     PTR-GROUP = POINTER;
     COMMENTS = truncated searches allowed using ? as
traling character;
   INDEX-NAME = REC08;
     SEARCHTERMS = RECNO;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('RECNO',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC10;
     SEARCHTERMS = CHAPTER,CHAP;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('CHAP',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC11;
     SEARCHTERMS = HEADER,HEAD;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('HEAD',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC12;
     SEARCHTERMS = MARG,MARG-NOTE,M;
       SRCPROC = $CAP/$SQU /$TRIM
/$BREAK/$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR,AS,BE,
FOR,BY,SHALL,IS,ON,ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT
,SUCH,AN,IF,MAY,
WHERE,A,SUBSECTION,PARAGRAPH,ABOVE,WITH,IT')/$TRUNC.SRC('?')
;
       PASSPROC = "$PASS.ELEM('MARG-N-OTHER',BREAK.NUM)
/$CALL(ISARRANGE)/$CALL(HRENT)/$CALL(CAP)/A48,'.',' ',',','
',':',' ',),' ','(',' ',';',' ','-',' ',' ','*',' ',' ','""','
',''''',' '/
$BREAK/$TEST.LEN(EX.LE,2)/$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR
,AS,BE,FOR,BY,SHALL,IS,ON,
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')";
     PTR-GROUP = POINTER;
     COMMENTS = defines linkage for MARG-N-OTHER. Index is
built by fetching value for this element, setting the value
to THE through the procedure ISARRANGE if ARRANGE has a
value, removing extraneous 'h' through HRENT procedure,
capitalizing, converting punctuation to spaces, breaking on
spaces, removing values <= 2 characters long or in stop list
;
   INDEX-NAME = REC14;
     SEARCHTERMS = SI;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SI',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC15;
     SEARCHTERMS = YEAR;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('YEAR',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC16;
     SEARCHTERMS = SUBSCHEDULE,SUBSCHED,SCHEDM;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SCHEDM',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC17;
     SEARCHTERMS = MISC,SUBSECTION,SUBSECT;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('MISC',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC18;
     SEARCHTERMS = CON;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('CON',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC19;
     SEARCHTERMS = SCHEDHIGH;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SCHEDHIGH',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC20;
     SEARCHTERMS = SUBHIGH;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SUBHIGH',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC21;
     SEARCHTERMS = SECTHIGH;
       SRCPROC = $INT;
       PASSPROC = $PASS.ELEM('SECTHIGH',BINARY)/$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = REC22;
     SEARCHTERMS = SCHEDN;
```

```
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('SCHEDN',BINARY)/$NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC23;
      SEARCHTERMS = TAPE;
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('TAPE',BINARY)/$NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC24;
      SEARCHTERMS = SUBPART;
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('SUBPART,CHAPTER.SCT',BINARY)/$
NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC25;
      SEARCHTERMS = DUPL;
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('DUPL',BINARY)/$NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC26;
      SEARCHTERMS = PARTLOW, PART;
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('PARTLOW,
PARTLOW.SCT',BINARY)/$NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC27;
      SEARCHTERMS = PARTHIGH;
        SRCPROC = $INT;
        PASSPROC = $PASS.ELEM('PARTHIGH,PARTHIGH.SCT',BINARY)
/$NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC28;
      SEARCHTERMS = KIND;
        SRCPROC = $CAP;
        PASSPROC = $PASS.ELEM('KIND',PHRASE)/ $NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC29;
      SEARCHTERMS = TYPESCH;
        SRCPROC = $CAP;
        PASSPROC = $PASS.ELEM('TYPESCH',PHRASE)/ $NULL;
      PTR-GROUP = POINTER;
    INDEX-NAME = REC30;
      SEARCHTERMS = TYPESUBSCH;
        SRCPROC = $CAP;
        PASSPROC = $PASS.ELEM('TYPESUBSCH',PHRASE)/ $NULL;
      PTR-GROUP = POINTER;
 GOALREC-NAME = ZCASE;
    PTR-ELEM = POINTER;
      EXTERNAL-NAME = ITEM;
        PASSPROC = A170;
        COMMENTS = defines global linkage (access to goal
records);
    INDEX-NAME = ZCASE2;
      SEARCHTERMS = WORD, W,INFO;
        SRCPROC = $CAP/$SQU/$TRIM /$BREAK/
$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR,AS,BE,FOR,BY,SHALL,IS,ON,
```

```
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')/$TRUNC.SRC('?');
        PASSPROC = "$PASS.ELEM('GEN-BOLD-TYPE,CASE-MAIN-TEXT,
CASE-UNID, GEN-UNID ,OMISS',WORD)/  A48,'.',' ',',',
',':',' '),' ','(',' ',';',' ','-',' ','*',' ','""',
',''',' '/
$BREAK/$TEST.LEN(EX.LE,2)/$EXCLUDE('THE,OF,TO,IN,THAT,AND,OR
,AS,BE,FOR,BY,SHALL,IS,ON,
ACT,THIS,ANY,SECTION,UNDER,WHICH,PART,NOT,SUCH,AN,IF,MAY,WHE
RE,A,SUBSECTION,
PARAGRAPH,ABOVE,WITH,IT')";
      PTR-GROUP = POINTER;
      COMMENTS = defines linkage for WORD (INFO) index,
built in similar way to INFO in the STATLT subfile (REC02
record-type);
    INDEX-NAME = ZCASE3;
      SEARCHTERMS = DATE,D;
        SRCPROC = $DATE(TRUNC,,W);
        PASSPROC = $PASS.ELEM('DATE',BINARY);
      PTR-GROUP = POINTER;
 SUBFILE-NAME = STATLT;
    EXP = The SPIRES subfile STATLT has been constructed to
provide;
    EXP = a small self-contained full-text legal retrieval
sysytem;
    EXP = and is separately known as SPILAW.It has been
designed;
    EXP = for the purposes of teaching and research into
electronic;
    EXP = retrieval methods. It uses the hierarchical
arrangement of;
    EXP = SPIRES to organize a data base with some of the
structures that;
    EXP = are inherent in legal texts;
    EXP = Experiments are in progress to determine the
optimum size;
    EXP = of the logical unit (an 'item').For the present, a
record;
    EXP = has been set at a paragraph for case law reports
and as a section;
    EXP = for statutes.;
    EXP;
    EXP = The subfile STATLT uses as base data the;
    EXP = Newcastle polytecnic Electronic Law LibrarY
(NELLY) which;
    EXP = includes the Property group and Landlord & Tenant
group of Statutes;
    EXP = in Force together with about 2Mb of case law
selected;
    EXP = for use in teaching in the same area of law.;
    EXP = Further information may be obtained from Dr
Michael Heather ,;
    EXP = School of Law,Newcastle Polytechnic.;
    GOAL-RECORD = REC01;
      ACCOUNTS = PUBLIC;
```

Appendix I: SPIRES FILEDEF for Statute Law

```
        FORMAT = STATLT.OUT;
        SECURE-SWITCHES = 3;
        SUBGOAL = REC40, ZSCH01, ZTRACK;
        LOG = 1, 2;
    GOAL-RECORD = REC01;
      ACCOUNTS = CL22, LAU1, LAU2;
        FORMAT = STATLT.OUT;
        SUBGOAL = REC40, ZSCH01, ZTRACK;
        LOG = 1, 2;
 SUBFILE-NAME = WORD INDEX;
    GOAL-RECORD = REC02;
      ACCOUNTS = CL22, LAU1, LAU2;
 SUBFILE-NAME = BROWSE INDEX;
    GOAL-RECORD = REC04;
      ACCOUNTS = CL22, LAU1, LAU2;
 SUBFILE-NAME = SPISPILAW;
    EXP = An overview of SPILAW;
    GOAL-RECORD = REC13;
      ACCOUNTS = CL22, LAU1, LAU2;
 SUBFILE-NAME = SECTION.IN.PART;
    EXP = holds the part identification information of each
section;
    GOAL-RECORD = REC40;
      ACCOUNTS = CL22, LAU1, LAU2;
    GOAL-RECORD = REC40;
      ACCOUNTS = PUBLIC;
        SECURE-SWITCHES = 3;
 SUBFILE-NAME = CASES;
```

```
    GOAL-RECORD = ZCASE;
      ACCOUNTS = PUBLIC;
        SECURE-SWITCHES = 3;
    GOAL-RECORD = ZCASE;
      ACCOUNTS = CL22, LAU1, LAU2;
  SUBFILE-NAME = LANDLORD AND TENANT CASES;
    GOAL-RECORD = ZCASE;
      ACCOUNTS = PUBLIC;
        SECURE-SWITCHES = 3;
    GOAL-RECORD = ZCASE;
      ACCOUNTS = CL22, LAU1, LAU2;
  SUBFILE-NAME = PARA.IN.SUBSCHEDULE;
    EXP = holds the subschedule identification information
of each paragraph;
    GOAL-RECORD = ZSCH01;
      ACCOUNTS = CL22, LAU1, LAU2;
    GOAL-RECORD = ZSCH01;
      ACCOUNTS = PUBLIC;
        SECURE-SWITCHES = 3;
  SUBFILE-NAME = TRACK;
    GOAL-RECORD = ZTRACK;
      ACCOUNTS = PUBLIC;
        SECURE-SWITCHES = 1, 2;
    GOAL-RECORD = ZTRACK;
      ACCOUNTS = CL22, LAU1, LAU2;
        SECURE-SWITCHES = 4;
  FILE-ACCESS = MASTER;
    ACCOUNTS = CL22;
```

```
FILE = LAU1.STATLT.DEF;
COMMENTS = Generated by FILE DEFINER 06/27/86;
DEFDATE = Fri. June 27, 1986;
MODDATE = Fri. June 27, 1986;
MODTIME = 15:12:54;
RECORD-NAME = REC40;
  REMOVED;
  FIXED;
    KEY = YEARCHAPSECT;
      OCC = 1;
      LEN = 9;
      INPROC = $LENGTH(9);
    ELEM = PARTSPEC;
      OCC = 1;
      LEN = 6;
      INPROC = $LENGTH(6);
RECORD-NAME = STATLT;
  REMOVED;
  FIXED;
    KEY = ID;
      OCC = 1;
      LEN = 36;
      INPROC = $LENGTH(36);
    ELEM = RECNO;
      OCC = 1;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  REQUIRED;
    ELEM = ACCESS-NO;
      OCC = 1;
      INPROC = $MAX.LEN(252)/ $SQU;
      ALIASES = AN, SN;
    ELEM = ACT-TITLE;
      OCC = 1;
      INPROC = $SQU;
      ALIASES = AT, TITLE;
    ELEM = KIND;
      OCC = 1;
    ELEM = DATE;
      OCC = 1;
      LEN = 4;
      INPROC = $DATE;
      OUTPROC = $DATE.OUT;
    ELEM = TAPE;
      OCC = 1;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = MARG-N-OTHER;
      ALIASES = MARG, MN, XOE-4T;
    ELEM = BROWSE;
      ALIASES = BROW;
    ELEM = INFO;
      ALIASES = W, WORD;
    ELEM = SCHEDULE-TEXT;
      ALIASES = X37-3T;
    ELEM = SECT-MAIN-TEXT;
      ALIASES = X5A-2T;
    ELEM = TEXT-IN-COL;
      ALIASES = ENGLISH, X7A-4T;
    ELEM = TEXT;
      ALIASES = X7F-3T;
    ELEM = SINGLE-INDENT2;
      ALIASES = X7B-2T;
    ELEM = MARG-N-REF-ACT;
      ALIASES = XOC-4T;
    ELEM = NOT-IND-SUB-SIN;
      ALIASES = XF7-2T;
    ELEM = PART-TEXT;
      ALIASES = X03-5T;
    ELEM = NOTE-EMB-SIN-IND;
      ALIASES = X5B-2T;
    ELEM = SINGLE-INDENT-SM;
      ALIASES = XF2-2T;
    ELEM = MARG-N-ACT-RUFF;
      ALIASES = XF6-2T;
    ELEM = SECTUNK12;
      ALIASES = XF5-2T;
    ELEM = SECTUNK11;
      ALIASES = X2E-2T;
    ELEM = GEN-BOLD-TYPE;
      ALIASES = X02-2T;
    ELEM = SCHSOURCE;
      OCC = 1;
    ELEM = DETAIL;
      TYPE = STR;
      INPROC = $sort(ascend.err);
    ELEM = KEY;
      OCC = 1;
    ELEM = KEYHEAD;
      OCC = 1;
    ELEM = TYPESCH;
      OCC = 1;
    ELEM = TYPESUBSCH;
      OCC = 1;
    ELEM = SO;
    ELEM = OUTNO;
    ELEM = LARGACT;
      OCC = 1;
    ELEM = PARTLOW.SCT;
      OCC = 1;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = PARTHIGH.SCT;
      OCC = 1;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = CHAPTER.SCT;
```

```
      OCC = 1;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SOW;
    ELEM = OUTNOW;
  VIRTUAL;
    ELEM = SI;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = YEAR;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = CHAP;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = HEAD;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SCHED;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SCHEDHIGH;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SCHEDM;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SUBHIGH;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SCHEDN;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SECT;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SECTHIGH;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = MISC;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = CON;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = DUPL;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = PARTLOW;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = PARTHIGH;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
    ELEM = SUBPART;
      LEN = 4;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  STRUCTURE = DETAIL;
    FIXED;
      KEY = SEQ;
        OCC = 1;
        LEN = 2;
        INPROC = $INT(2);
        OUTPROC = $INT.OUT;
    OPTIONAL;
      ELEM = GEN-BOLD-TYPE;
        ALIASES = X02-1;
      ELEM = UPPER-CASE;
        ALIASES = X03-1;
      ELEM = ARRANGE;
        ALIASES = XF3-1;
      ELEM = PREAMBLE;
        ALIASES = XF0-1;
      ELEM = MINOR-COL-HEAD;
        ALIASES = X16-1;
      ELEM = TWO-COLUMNS;
        ALIASES = ARRANGE-SECTIONS, X61-1;
      ELEM = GEN-SUBHEADING;
        ALIASES = X2D-1;
      ELEM = SMALL-REPRO-TXT;
        ALIASES = XF2-1;
      ELEM = REPR-LONG-TITLE;
        ALIASES = XF5-1;
      ELEM = GENUNK3;
        ALIASES = XF7-1;
      ELEM = SMALL-REPRO-MRG;
        ALIASES = XF8-1;
      ELEM = SMALL-REPRO-COL;
        ALIASES = X18-1;
      ELEM = MARG-N-REF-ACT;
        ALIASES = X0C-4;
      ELEM = CROSSNOTE;
        ALIASES = X4B-4;
      ELEM = FOOTNOTE;
  ALIASES = X3F-4;
ELEM = FOOTN-OLD-STAT;
  ALIASES = XF9-4;
ELEM = OMISS;
  ALIASES = XF1-4;
ELEM = OLD-STAT-BANNER;
  ALIASES = X7F-4;
ELEM = TEXT-IN-COL;
  ALIASES = ENGLISH, X7A-4;
ELEM = GEN-UNID;
  ALIASES = LATIN;
ELEM = SCHEDULE-TEXT;
  ALIASES = X37-3;
ELEM = CENTRE-HEAD;
  ALIASES = X2E-3;
ELEM = NUMB-PARA;
  ALIASES = X5A-3;
ELEM = TEXT;
  ALIASES = X7F-3;
ELEM = SINGLE-IND-SUB;
  ALIASES = X0B-3;
ELEM = SINGLE-IND-SUB2;
  ALIASES = X6C-3;
ELEM = DOUBLE-IND-SM;
  ALIASES = X5B-3;
ELEM = DOUBLE-IND-SM2;
  ALIASES = NIL3, X5D-3;
ELEM = DOUBLE-IND-SM3;
  ALIASES = X7D-3;
ELEM = DOUBLE-IND-SM4;
  ALIASES = X4D-3;
ELEM = TRIPLE-IND;
  ALIASES = X4E-3;
ELEM = TRIPLE-IND2;
  ALIASES = X10-3;
ELEM = COL-PR-HEAD;
  ALIASES = X11-3;
ELEM = COL-PR-HEAD2;
  ALIASES = X3D-3;
ELEM = COL-PR-DOUBLE;
  ALIASES = X12-3;
ELEM = COL-PR-DOUBLE2;
  ALIASES = X3C-3;
ELEM = COL-PR-TEXT;
  ALIASES = X13-3;
ELEM = PRINT-TABULAR;
  ALIASES = XF0-3;
ELEM = STARRED-NOTE-SM;
  ALIASES = XF2-3;
ELEM = COL-PR-HEAD-UNRL;
  ALIASES = ARRANGE-SCHEDULE, XF3-3;
ELEM = COL-PR-HEAD-REPR;
  ALIASES = XF6-3;
ELEM = COL-PR-HEAD-MED;
  ALIASES = X61-3;
ELEM = SIN-IND-SUB-NOTE;
```

```
  ALIASES = X7B-3;
ELEM = COL-PRINT;
  ALIASES = XTOF00-3;
ELEM = FORM-TIT-BOLD-UC;
  ALIASES = X2F-3;
ELEM = INSTRUCTION;
  ALIASES = XF7-3;
ELEM = CARRIAGE-CONTR;
  ALIASES = X0D-3;
ELEM = COL-PR-HEAD3;
  ALIASES = X25-3;
ELEM = COL-PR-HEAD-SM;
  ALIASES = XF4-3;
ELEM = DOC-TIT-CNT-ITAL;
  ALIASES = X2D-3;
ELEM = DBL-IND-SUB-NIL2;
  ALIASES = X50-3;
ELEM = PART-TEXT-IN-SCH;
  ALIASES = X03-3;
ELEM = SCHED-UNID;
ELEM = SECT-MAIN-TEXT;
  ALIASES = X5A-2;
ELEM = CONT-TEXT;
  ALIASES = X7F-2;
ELEM = INDENT-TEXT-SUB;
  ALIASES = NIL6, X25-2;
ELEM = SINGLE-INDENT-SM;
  ALIASES = XF2-2;
ELEM = SINGLE-INDENT;
  ALIASES = X6C-2;
ELEM = HALF-I-SIN-I-SUB;
  ALIASES = X7B-2;
ELEM = DBL-INDENT-LIST;
  ALIASES = X7D-2;
ELEM = DBL-IND-SUB-NIL;
  ALIASES = X50-2;
ELEM = NOT-IND-SUB-SIN;
  ALIASES = XF7-2;
ELEM = REPR-MARG-NOT-SM;
  ALIASES = XF8-2;
ELEM = NOTE-EMB-SIN-IND;
  ALIASES = X5B-2;
ELEM = DOUBLE-INDENT;
  ALIASES = X5C-2;
ELEM = DOUBLE-INDENT2;
  ALIASES = X5D-2;
ELEM = DBL-IND-FST-MORE;
  ALIASES = X4D-2;
ELEM = TRIPLE-IND-LIST;
  ALIASES = X60-2;
ELEM = TRIPLE-IND-LIST2;
  ALIASES = X4E-2;
ELEM = MARG-N-ACT-RUFF;
  ALIASES = XF6-2;
ELEM = SECTUNK11;
  ALIASES = X2E-2;
```

```
      ELEM = SECTUNK12;
        ALIASES = XF5-2;
      ELEM = SECT-UNID;
      ELEM = PART-TEXT;
        ALIASES = X03-5;
      ELEM = SINGLE-IND-SM-P;
        ALIASES = XF2-5;
      ELEM = REP-MARG-NOT-SMP;
        ALIASES = XF8-5;
      ELEM = PART-SUBHEADING;
        ALIASES = X2D-5;
      ELEM = CONT-TEXT-PART;
        ALIASES = X7F-5;
      ELEM = NOT-IND-SUB-SINP;
        ALIASES = XF7-5;
      ELEM = INDENT-TEXT-SUBP;
        ALIASES = X25-5;
      ELEM = PART-UNID;
RECORD-NAME = ZIN04;
  REQUIRED;
    KEY = RECNO;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN05;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = ACT-TITLE;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN06;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = KIND;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN07;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = DATE;
      INPROC = $DATE;
      OUTPROC = $DATE.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
```

```
        OUTPROC = $HEX.OUT;
 RECORD-NAME = ZIN08;
   COMBINE = ZIN04;
   REQUIRED;
     KEY = MARG-N-OTHER;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
       INPROC = $HEX;
       OUTPROC = $HEX.OUT;
 RECORD-NAME = ZIN09;
   COMBINE = ZIN04;
   REQUIRED;
     KEY = BROWSE;
       ALIASES = COL-PR-HEAD-MED, COL-PR-HEAD-UNRL,
DOC-TIT-CNT-ITAL, GEN-SUBHEADING, MINOR-COL-HEAD,
PART-SUBHEADING, PART-TEXT, PART-TEXT-IN-SCH, TWO-COLUMNS;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
       INPROC = $HEX;
       OUTPROC = $HEX.OUT;
 RECORD-NAME = ZIN10;
   COMBINE = ZIN04;
   REQUIRED;
     KEY = INFO;
       ALIASES = ARRANGE, CARRIAGE-CONTR, CENTRE-HEAD,
COL-PR-DOUBLE, COL-PR-DOUBLE2, COL-PR-HEAD,
COL-PR-HEAD-REPR, COL-PR-HEAD-SM, COL-PR-HEAD2,
COL-PR-HEAD3, COL-PR-TEXT, COL-PRINT, CONT-TEXT,
CONT-TEXT-PART, CROSSNOTE, DBL-IND-FST-MORE,
DBL-IND-SUB-NIL, DBL-IND-SUB-NIL2, DBL-INDENT-LIST,
DOUBLE-IND-SM, DOUBLE-IND-SM2, DOUBLE-IND-SM3,
DOUBLE-IND-SM4, DOUBLE-INDENT, DOUBLE-INDENT2,
FOOTN-OLD-STAT, FOOTNOTE, FORM-TIT-BOLD-UC, GEN-BOLD-TYPE,
GEN-UNID, GENUNK3, HALF-I-SIN-I-SUB, INDENT-TEXT-SUB,
INDENT-TEXT-SUBP, INSTRUCTION, MARG-N-ACT-RUFF,
MARG-N-REF-ACT, NOT-IND-SUB-SIN, NOT-IND-SUB-SINP,
NOTE-EMB-SIN-IND, NUMB-PARA, OLD-STAT-BANNER, OMISS,
PART-TEXT, PART-UNID, PREAMBLE, PRINT-TABULAR,
REP-MARG-NOT-SMP, REPR-LONG-TITLE, REPR-MARG-NOT-SM,
SCHED-UNID, SCHEDULE-TEXT, SECT-MAIN-TEXT, SECT-UNID,
SECTUNK11, SECTUNK12, SIN-IND-SUB-NOTE, SINGLE-IND-SM-P,
SINGLE-IND-SUB, SINGLE-IND-SUB2, SINGLE-INDENT,
SINGLE-INDENT-SM, SINGLE-INDENT2, SMALL-REPRO-COL,
SMALL-REPRO-MRG, SMALL-REPRO-TXT, STARRED-NOTE-SM, TEXT,
TEXT-IN-COL, TRIPLE-IND, TRIPLE-IND-LIST, TRIPLE-IND-LIST2,
TRIPLE-IND2;
   OPTIONAL;
     ELEM = POINTER;
       TYPE = LCTR;
       INPROC = $HEX;
       OUTPROC = $HEX.OUT;
 RECORD-NAME = ZIN11;
   COMBINE = ZIN04;
   REQUIRED;
```

```
    KEY = SCHEDULE-TEXT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN12;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = UPPER-CASE;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN13;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = TYPESCH;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN14;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = TYPESUBSCH;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN15;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SI;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN16;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = YEAR;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN17;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = CHAP;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN18;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = HEAD;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN19;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SCHED;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN20;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SCHEDHIGH;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN21;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SCHEDM;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN22;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SUBHIGH;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN23;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SCHEDN;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN24;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SECT;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN25;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SECTHIGH;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN26;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = MISC;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN27;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = CON;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN28;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = DUPL;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN29;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = PARTLOW;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN30;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = PARTHIGH;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZIN31;
  COMBINE = ZIN04;
  REQUIRED;
    KEY = SUBPART;
      INPROC = $INT(4);
      OUTPROC = $INT.OUT;
  OPTIONAL;
    ELEM = POINTER;
      TYPE = LCTR;
      INPROC = $HEX;
      OUTPROC = $HEX.OUT;
RECORD-NAME = ZSCH01;
  REMOVED;
  FIXED;
    KEY = YEARCHAPSCHPARA;
```

```
        OCC = 1;
        LEN = 12;
        INPROC = $LENGTH(12);
      ELEM = SUBSCHSPEC;
        OCC = 1;
        LEN = 4;
        INPROC = $LENGTH(4);
 GOALREC-NAME = STATLT;
   PTR-ELEM = POINTER;
      EXTERNAL-NAME = RECORD;
        GOALREC-KEY = ID;
        PASSPROC = $PASS.LCTR;
   INDEX-NAME = ZIN04;
      SEARCHTERMS = RECNO;
        SRCPROC = $INT(4);
        PASSPROC = $PASS.ELEM('RECNO',NUMERIC)/ $NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN05;
      SEARCHTERMS = ACT-TITLE, AT, TITLE;
        SRCPROC = $WORD;
        PASSPROC = $PASS.ELEM('ACT-TITLE',2)/ $WORD(PASS)/
$NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN06;
      SEARCHTERMS = KIND;
        SRCPROC = $CAP;
        PASSPROC = $PASS.ELEM('KIND',1)/ $NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN07;
      SEARCHTERMS = DATE;
        SRCPROC = $DATE(TRUNC);
        PASSPROC = $PASS.ELEM('DATE',NUMERIC)/ $NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN08;
      SEARCHTERMS = MARG-N-OTHER, MARG, MN, XOE-4T;
        SRCPROC = $WORD;
        PASSPROC = $PASS.ELEM('MARG-N-OTHER',2)/
$WORD(PASS)/ $NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN09;
      SEARCHTERMS = BROWSE, BROW, MINOR-COL-HEAD,
TWO-COLUMNS, GEN-SUBHEADING, COL-PR-HEAD-UNRL,
COL-PR-HEAD-MED, DOC-TIT-CNT-ITAL, PART-TEXT-IN-SCH,
PART-TEXT, PART-SUBHEADING;
        SRCPROC = $WORD;
        PASSPROC = $PASS.ELEM('BROWSE,MINOR-COL-HEAD,TWO-COLU
MNS,GEN-SUBHEADING,COL-PR-HEAD
-UNRL,COL-PR-HEAD-MED,DOC-TIT-CNT-ITAL,PART-TEXT-IN-SCH,PART
-TEXT,PART-S
UBHEADING',2)/ $WORD(PASS)/ $NULL;
      PTR-GROUP = POINTER;
   INDEX-NAME = ZIN10;
      SEARCHTERMS = INFO, WORD, W, SECT-MAIN-TEXT,
TEXT-IN-COL, TEXT, SINGLE-INDENT2, MARG-N-REF-ACT,
NOT-IND-SUB-SIN, PART-TEXT, NOTE-EMB-SIN-IND,
SINGLE-INDENT-SM, MARG-N-ACT-RUFF, SECTUNK12, SECTUNK11,
```

```
| GEN-BOLD-TYPE, GEN-BOLD-TYPE, ARRANGE, PREAMBLE,
| SMALL-REPRO-TXT, REPR-LONG-TITLE, GENUNK3, SMALL-REPRO-MRG,
| SMALL-REPRO-COL, MARG-N-REF-ACT, CROSSNOTE, FOOTNOTE,
| FOOTN-OLD-STAT, OMISS, OLD-STAT-BANNER, TEXT-IN-COL,
| GEN-UNID, SCHEDULE-TEXT, CENTRE-HEAD, NUMB-PARA, TEXT,
| SINGLE-IND-SUB, SINGLE-IND-SUB2, DOUBLE-IND-SM,
| DOUBLE-IND-SM2, DOUBLE-IND-SM3, DOUBLE-IND-SM4, TRIPLE-IND,
| TRIPLE-IND2, COL-PR-HEAD, COL-PR-HEAD2, COL-PR-DOUBLE,
| COL-PR-DOUBLE2, COL-PR-TEXT, PRINT-TABULAR,
| STARRED-NOTE-SM, COL-PR-HEAD-REPR, SIN-IND-SUB-NOTE,
| COL-PRINT, FORM-TIT-BOLD-UC, INSTRUCTION, CARRIAGE-CONTR,
| COL-PR-HEAD3, COL-PR-HEAD-SM, DBL-IND-SUB-NIL2, SCHED-UNID,
| SECT-MAIN-TEXT, CONT-TEXT, INDENT-TEXT-SUB,
| SINGLE-INDENT-SM, SINGLE-INDENT, HALF-I-SIN-I-SUB,
| DBL-INDENT-LIST, DBL-IND-SUB-NIL, NOT-IND-SUB-SIN,
| REPR-MARG-NOT-SM, NOTE-EMB-SIN-IND, DOUBLE-INDENT,
| DOUBLE-INDENT2, DBL-IND-FST-MORE, TRIPLE-IND-LIST,
| TRIPLE-IND-LIST2, MARG-N-ACT-RUFF, SECTUNK11, SECTUNK12,
| SECT-UNID, SINGLE-IND-SM-P, REP-MARG-NOT-SMP,
| CONT-TEXT-PART, NOT-IND-SUB-SINP, INDENT-TEXT-SUBP,
| PART-UNID;
|          SRCPROC = $WORD;
|          PASSPROC = $PASS.ELEM('INFO,SECT-MAIN-TEXT,TEXT-IN-CO
| L,TEXT,SINGLE-INDENT2,MARG-N-R
| EF-ACT,NOT-IND-SUB-SIN,PART-TEXT,NOTE-EMB-SIN-IND,SINGLE-IND
| ENT-SM,MARG-
| N-ACT-RUFF,SECTUNK12,SECTUNK11,GEN-BOLD-TYPE,GEN-BOLD-TYPE,A
| RRANGE,PREAM
| BLE,SMALL-RFPRO-TXT,REPR-LONG-TITLE,GENUNK3,SMALL-REPRO-MRG,
| SMALL-REPRO-
| COL,MARG-N-REF-ACT,CROSSNOTE,FOOTNOTE,FOOTN-OLD-STAT,OMISS,O
| LD-STAT-BANN
| ER,TEXT-IN-COL,GEN-UNID,SCHEDULE-TEXT,CENTRE-HEAD,NUMB-PARA,
| TEXT,SINGLE-
| IND-SUB,SINGLE-IND-SUB2,DOUBLE-IND-SM,DOUBLE-IND-SM2,DOUBLE-
| IND-SM3,DOUB
| LE-IND-SM4,TRIPLE-IND,TRIPLE-IND2,COL-PR-HEAD,COL-PR-HEAD2,C
| OL-PR-DOUBLE
| ,COL-PR-DOUBLE2,COL-PR-TEXT,PRINT-TABULAR,STARRED-NOTE-SM,CO
| L-PR-HEAD-RE
| PR,SIN-IND-SUB-NOTE,COL-PRINT,FORM-TIT-BOLD-UC,INSTRUCTION,C
| ARRIAGE-CONT
| R,COL-PR-HEAD3,COL-PR-HEAD-SM,DBL-IND-SUB-NIL2,SCHED-UNID,SE
| CT-MAIN-TEXT
| ,CONT-TEXT,INDENT-TEXT-SUB,SINGLE-INDENT-SM,SINGLE-INDENT,HA
| LF-I-SIN-I-S
| UB,DBL-INDENT-LIST,DBL-IND-SUB-NIL,NOT-IND-SUB-SIN,REPR-MARG
| -NOT-SM,NOTE
| -EMB-SIN-IND,DOUBLE-INDENT,DOUBLE-INDENT2,DBL-IND-FST-MORE,T
| RIPLE-IND-LI
| ST,TRIPLE-IND-LIST2,MARG-N-ACT-RUFF,SECTUNK11,SECTUNK12,SECT
| -UNID,SINGLE
| -IND-SM-P,REP-MARG-NOT-SMP,CONT-TEXT-PART,NOT-IND-SUB-SINP,I
| NDENT-TEXT-S
| UBP,PART-UNID',2)/ $WORD(PASS)/ $NULL;
|      PTR-GROUP = POINTER;
```

```
   INDEX-NAME = ZIN11;
     SEARCHTERMS = SCHEDULE-TEXT, X37-3T;
       SRCPROC = $CAP;
       PASSPROC = $PASS.ELEM('SCHEDULE-TEXT',1)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN12;
     SEARCHTERMS = UPPER-CASE, X03-1;
       SRCPROC = $WORD;
       PASSPROC = $PASS.ELEM('UPPER-CASE',2)/ $WORD(PASS)/
$NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN13;
     SEARCHTERMS = TYPESCH;
       SRCPROC = $CAP;
       PASSPROC = $PASS.ELEM('TYPESCH',1)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN14;
     SEARCHTERMS = TYPESUBSCH;
       SRCPROC = $CAP;
       PASSPROC = $PASS.ELEM('TYPESUBSCH',1)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN15;
     SEARCHTERMS = SI;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SI',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN16;
     SEARCHTERMS = YEAR;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('YEAR',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN17;
     SEARCHTERMS = CHAP;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('CHAP',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN18;
     SEARCHTERMS = HEAD;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('HEAD',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN19;
     SEARCHTERMS = SCHED;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SCHED',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN20;
     SEARCHTERMS = SCHEDHIGH;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SCHEDHIGH',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN21;
     SEARCHTERMS = SCHEDM;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SCHEDM',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
```

```
   INDEX-NAME = ZIN22;
     SEARCHTERMS = SUBHIGH;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SUBHIGH',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN23;
     SEARCHTERMS = SCHEDN;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SCHEDN',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN24;
     SEARCHTERMS = SECT;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SECT',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN25;
     SEARCHTERMS = SECTHIGH;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SECTHIGH',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN26;
     SEARCHTERMS = MISC;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('MISC',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN27;
     SEARCHTERMS = CON;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('CON',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN28;
     SEARCHTERMS = DUPL;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('DUPL',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN29;
     SEARCHTERMS = PARTLOW;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('PARTLOW',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN30;
     SEARCHTERMS = PARTHIGH;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('PARTHIGH',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
   INDEX-NAME = ZIN31;
     SEARCHTERMS = SUBPART;
       SRCPROC = $INT(4);
       PASSPROC = $PASS.ELEM('SUBPART',NUMERIC)/ $NULL;
     PTR-GROUP = POINTER;
 SUBFILE-NAME = STATLT.DEF;
   GOAL-RECORD = STATLT;
     ACCOUNTS = LAU1;
 SUBFILE-NAME = SECTION.IN.PART;
   GOAL-RECORD = REC40;
     ACCOUNTS = LAU1;
```

Appendix II: SPIRES FILEDEF for Statute Law derived from File Definer

```
SUBFILE-NAME = PARA.IN.SUBSCHEDULE;                           |      ACCOUNTS = LAU1;
  GOAL-RECORD = ZSCH01;                                       |
```