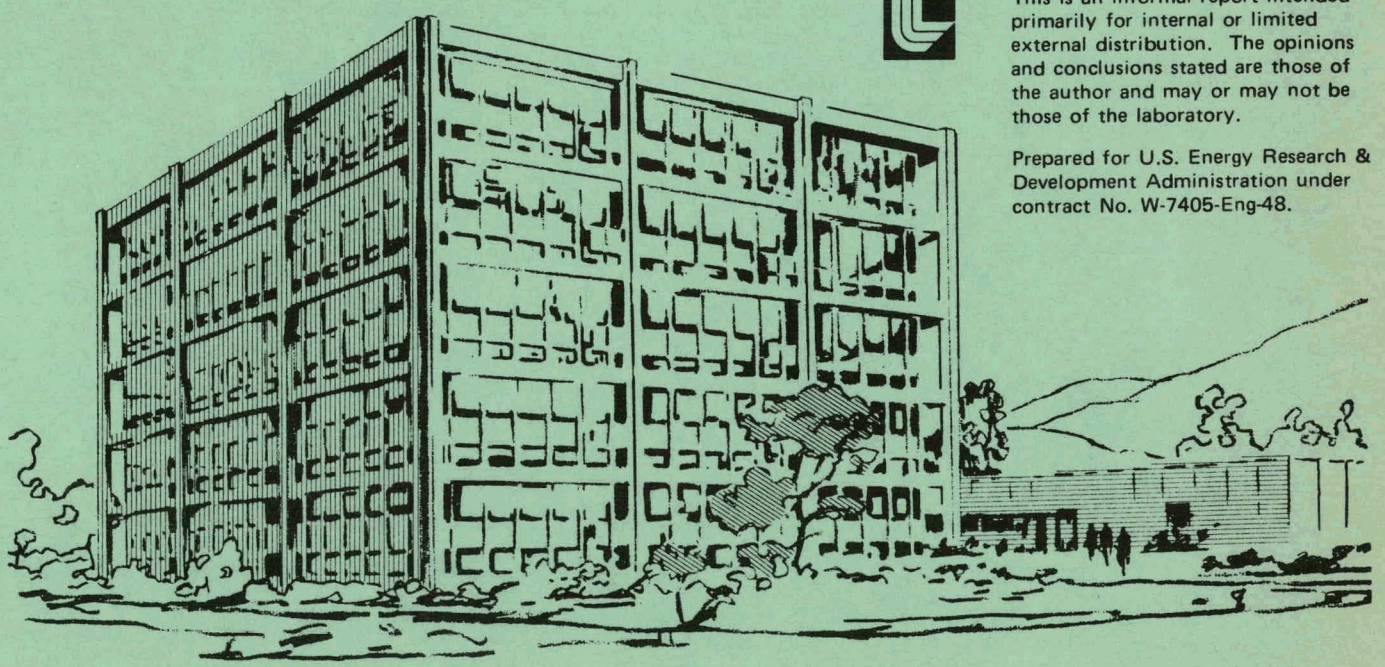
UCID-17378

Lawrence Livermore Laboratory

SET THEORETIC DATA STRUCTURES (STDS):
A TUTORIAL

Edward W. Birss and Jeffry W. Yeh

January 31, 1977



Prepared for U.S. Energy Research & Development Administration under contract No. W-7405-Eng-48.

DISCLAIMER

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

FOREWORD

The Set Theoretic approach to data base management was investigated as a potential solution to the problem of storing and manipulating large data bases. The Set Theoretic approach had generated interest as a technique to manage large amounts of data in a complex yet efficient manner, and a more detailed investigation was begun. This report documents the study of an implementation: Set Theoretic Data Structures.

The Data Management Research Project at Lawrence Livermore Laboratory produced this report on Set Theoretic Data Structures as part of Contract [RA] 76-12 with the Transportation Systems Center of the U.S. Department of Transportation (DOT/TSC). This report will be submitted to contract monitor Alan Kaprelian (Information Division, DOT/TSC, Cambridge, Massachusetts).

CONTENTS

SET THEORETIC DATA STRUCTURES (STDS): A TUTORIAL

ABSTRACT

Extended Set Theory as a data base management discipline has received attention in the data base literature. Set Theoretic Information Systems Corporation has, for some time, marketed a data base system based on the foundation of Extended Set Theory. This system is called Set Theoretic Data Structures (STDS). A series of examples shows that STDS is similar to relational algebraic data base management systems. The advantages of STDS are its straightforward data base design, compact data representation, and flexible, powerful data manipulation operators; while its limitations are its low-level primitive user interface and the partial implementation of the Extended Set Theoretic concepts. To make STDS very attractive, a "user-friendly" interface should be developed, and some distinctive features of Extended Set Theory (such as sets of sets) should be implemented.

INTRODUCTION

The Extended Set Theoretic approach to data base management as proposed by D. L. Childs has received attention as an alternative approach in recent years. In spite of its early history (i.e., 1968), little has been published through the years, Refs. 1 through 3 being the principal publications. W. T. Hardgrave, however, has also actively published in the area of Extended Set Theory.[4-7]

To show the feasibility of the approach, several software implementations of data base management systems using the foundation of Extended Set Theory have been produced. Two of these are the Set Theoretic Data Structure packages (STDS-I and STDS-OS) produced by Set Theoretic Information Systems Corporation. STDS-I and STDS-OS operate on IBM 360/370 and Amdahl computers.

The purpose of this paper is fourfold:

- To present a brief overview of the Extended Set Theory sufficient to motivate the approach
- To present an introduction to STDS-I and to note major differences in STDS-OS
- To provide some example query and update programs to demonstrate the salient characteristics and capabilities of the system

- To assess the capabilities and limitations of the systems.

To illustrate the flexibility and breadth, the examples provided in this report will be illustrated in STDS-I. STDS-I was chosen over STDS-OS as an illustrative tool for two basic reasons: (1) STDS-I is more powerful, has more operations, and is more flexible, and (2) the transportation data bases, which are of primary interest to this contract, were installed under STDS-I. The examples in STDS will be similar to those presented in Ref. 8 and will use the same Presidential data base. Thus, this report, when compared with the articles in Ref. 8, will give the reader a direct comparison and an accurate perspective with which to view an Extended Set Theoretic implementation: STDS.

BACKGROUND

Extended Set Theory was developed by David L. Childs with support from the CONCOMP project at the University of Michigan. Childs realized that computer data structures did not have a rigorous mathematical formulation, and began to develop a definition that would ultimately lead to practical results when applied to the computer environment.

To achieve a mathematical definition of computer data structures, particularly those used in data bases, Childs investigated classical set theory. The choice of classical set theory was a natural first step, because a data base record might be viewed as an n-tuple where each field in the record represents a domain of the n-tuple. However, classical set theory has some definite shortcomings when applied to records and data bases. These problems arise because of the definition of the n-tuple.

A standard classical set theoretic definition of the ordered pair (2-tuple) is:

$$\langle a,b\rangle = \left\{\{a\}, \{a,b\}\right\}.$$

This definition is extended to n-tuples in a straightforward manner:

$$\langle a,b,c \cdots \rangle = \left\{\{a\}, \{a,b\}, \{a,b,c\} \cdots\right\}.$$

When this definition is applied to computer data structures, this definition quickly breaks down:

$$\langle 1,0,1\rangle = \left\{\{1\}, \{1,0\}, \{1,0,1\}\right\} = \left\{\{1\}, \{1,0\}\right\} = \langle 1,0\rangle .$$

This example has demonstrated an anomaly with the classical set theoretic definition of the n-tuple. This anomaly, however, is not the only problem.

Certain obvious classical set theoretic operations on n-tuples are undefined, largely because of the definition of the n-tuple.

To achieve a better foundation for computer data structures, Childs developed a new definition of a set. The set E is defined as:

$$E = \left\{ a_1^{i_1}, a_2^{i_2}, a_3^{i_3}, a_4^{i_4}, \ldots a_n^{i_n} \right\},$$

where a_j is an atom of a set and i_j is a position indicator. Note than an n-tuple is now just a special case of a set:

$$<1,0,1,2> = \left\{ 1^1, 0^2, 1^3, 2^4 \right\}.$$

The advantages of the definition of the set become obvious:

1. There is no problem distinguishing between like elements with different positions.
2. n-tuples need not be considered specially; n-tuples are special cases of sets.
3. All classical set operations may be defined on these sets.
4. New operations on these sets may be defined.

INTRODUCTION TO SET THEORETIC DATA STRUCTURES (STDS)

It is this definition of the set that motivated the implementation of the STDS. The STDS software package was originally developed to experiment with these sets. However, once operational, STDS proved to be quite effective; versions of it are currently being marketed by Set Theoretic Information Systems Corporation. The current version of STDS limits itself to the special case of the set where all elements of the sets are atoms (the n-tuple case).

As alluded to previously, a collection of n-tuples can be viewed as a table:

$Q = \left\{ \langle name^1, age^2 \rangle : \Gamma \right\}$ can be viewed as

Name	Age
Sam	5
Fred	3
Susan	2
Mary	1
.	
.	
.	

(Γ is the membership-condition that must be valid for all members of the set. In this particular example, Γ could be children of Jim Smith.)

The STDS package itself consists primarily of operations on sets or tables. The operations that manipulate these tables can be categorized as follows:

1. I/O operations
2. Set operations
3. Arithmetic operations
4. Utility functions

A relatively brief description of the available Extended Set Theoretic Operations in STDS-I is included in Appendix A. For a more complete description see Refs. 9 and 10. To understand the remaining portions of this report, only these few operations need to be explained:

I/O and Utility	Set Operations
GET	UN
PUT	XPAN8
SETFMT	RMIX8
LIST	XSET
QSFILE	RS8
QRETURN	LEGL

In order to describe the above functions, examples will be drawn from the Presidential data base.[8] For the purposes of explanation, the examples will be limited to two tables, Presidents and Elections:

Presidents

PRES#,	LASTNAME...

Elections

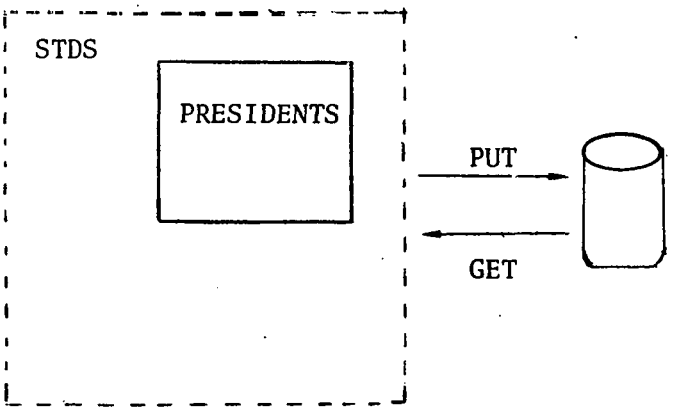
PRES#,	ELEC YR

GET and PUT — Retrieve Sets from Archival Storage (i.e., archive sets)

The GET operation retrieves an already stored set (table) from secondary storage. The set (table) must have previously been stored by a PUT operation.

The PUT operation stores a set on secondary storage.

Example



SETFMT and LIST — Output Operations for Reports

The SETFMT operation is used to define a particular FORTRAN-like FORMAT statement for the purpose of printing the contents of a set (table).

The LIST command performs the actual output of the set using a user-supplied format, or one previously defined by use of the SETFMT operation.

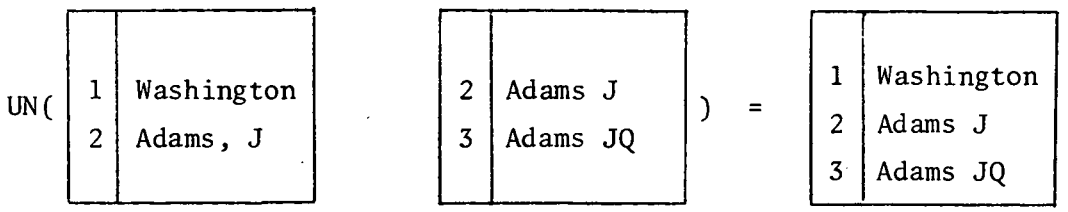
QSFILE and QRETURN — Get Commands from a File

QSFILE is a utility function that permits input to be read from a file instead of the interactive terminal. The last line of the file should contain a QRETURN, to redirect the input from the file back to the interactive terminal.

UN — Produce the Union of Two Sets

Given two sets, a resultant set is constructed that contains each row belonging to either input set. The resultant set does not contain duplicates.

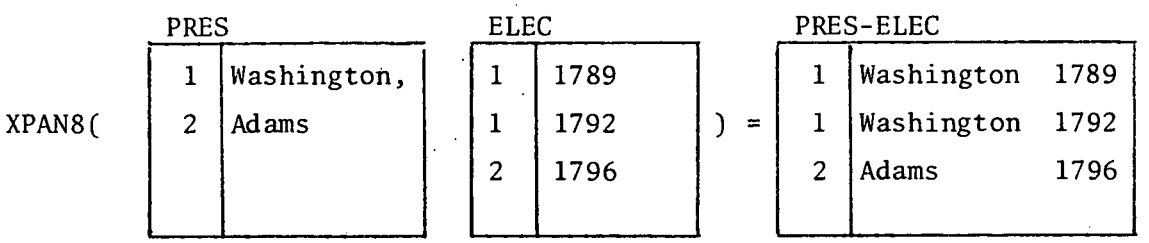
Example



XPAN8 — Expand to Sets (JOIN)

XPAN8 compares the first domains of two inputs, and where a match occurs, constructs a resultant set containing the concatenation of the data fields from the two input sets:

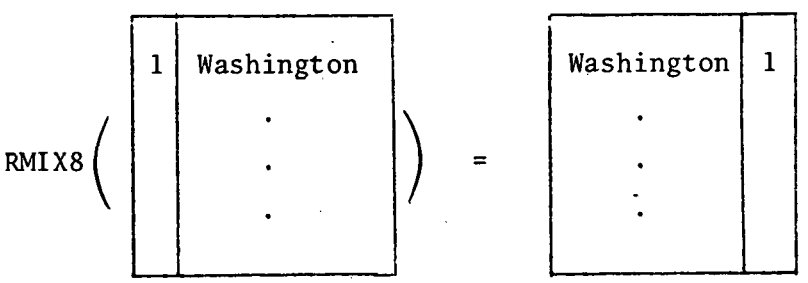
Example



Currently, XPAN8 is not available in STDS-OS.

RMIX8 — Rearrange the Domains of a Set

RMIX8 rearranges domains within a set. An index set is used to specify the rearrangement (see XSET). This operation could be used to permute the President's number with his name as shown below:



XSET — Create an Index Set

XSET is used to create an index set. An index set specifies domain indices (usually bytes), which form the new format of the set using RMIX8.

For example, given a set P containing the character string:

```
          1111111
1234567890123456
NOW IS THE TIME,
```

and the desire to create the resultant set Q:

```
          1111111
1234567890123456
THE TIME IS NOW,
```

we would specify an index set X:

X	TP	FP	LEN
	1	8	9
	10	5	3
	13	1	4

where the domains of the index set X are:

TP — to position; the position the field is to be in the resultant set

FP — from position, the position the field is coming from in the input set

Len — the length of the field.

Once the index set X has been created using XSET, the RMIX8 operation is performed that references P as an input set, Q as a resultant set, and X as an index set.

RS8 — Restriction Operation

RS8 is a restriction operation that compares 8-bit domains in an input set with those in a restriction set, and produces a resultant set that contains rows of the input set that meet the restriction criteria of the restriction set.

Example

PRES

1	Washington
2	Adams J
3	Adams JQ

ELEC

2	1796

RS8 on domain 1 of PRES, using ELEC as restriction set, yields:

2	Adams J

LEGL — Logical Compare of Components of a Set

LEGL performs a logical compare of two components I and J of a set, and produces three resultant sets; SL whose components are such that I < J, SE such that I = J, SG such that I > J.

Example

LEGL (I J) =

I	J
1	2
2	2
3	1
4	5

SL

1	2
4	5

SE

2	2

SG

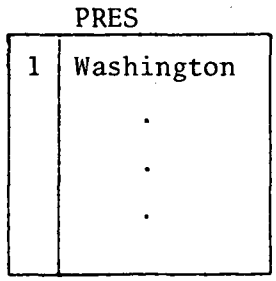
3	1

The STDS User Interface

STDS-I has a rather low-level user-interface for manipulating sets. Unlike the examples shown in the previous section the domains are not named and have a physical orientation. Each set has a domain declaration that is either 8, 16, or 32 bits. The length of the n-tuples (rows of the set) are integral multiples of the domain declaration. Because logical quantities stored are frequently larger than the physical domain declaration, logical

domains are referenced by giving the physical domain index (position) and a length (number of domains). However, STDS-OS does permit naming of domains.

Example



Set PRES might have a domain declaration of 32, but then all logical domains (fields) would have to begin on word boundaries on IBM 360/370 computers. Hence a reasonable alternative is to use an 8-bit (1 byte) domain declaration. If 2 bytes are allocated for president number, and ten for president's last name, then the president number is domain 1 with length 2, and the president's last name is domain 3 with length 10.

The physical orientation of the STDS-I package requires the user to know the position, format, and length of the data on the storage device. In subsequent examples, except where explicitly indicated, all fields are character representation (i.e., 8-bit domains).

STDS is accessed in two possible fashions:

1. By programming language (FORTRAN or COBOL) using CALL statements.
2. By an interactive interface. This interface is extremely simple and essentially requires the user to name the operation and to provide the parameters. The only basic difference between this interface and the programming language interface is that the user need not code the four letters "CALL"; otherwise the interfaces are nearly identical.

To provide a comprehensive set of examples of use of STDS, more detail about the presidential data will be presented in the next section.

PRESIDENTIAL DATA BASE

The Presidential data base contains information about the Presidents of the United States and their associated Congresses and Administrations, plus selected information about the States of the Union. This data base was used in a recent issue of Computing Surveys (Ref. 8 edited by E. Sibley), which contains tutorials on the various data base management disciplines including the relational, CODASYL and hierarchic data base approaches. In addition, this data base has also been implemented under IBM's Information Management

System (IMS).[11] Using this data base as our example, we will provide the reader with a vehicle for comparing the STDS system with other data base management systems.

A second advantage of the Presidential data base is that the information is well recognized and understood by a majority of people in the United States. This data base is small enough to be manageable but still complex enough to have different types of records and relationships. These include items, groups, repeating items, repeating groups, one-to-many relationships, and many-to-many relationships.

The implementation of the Presidential data base under the STDS system was divided into three steps:

1. The study of the raw data and their interrelationships
2. The design of an STDS data base and the loading of raw data into the STDS data base
3. The design and implementation of the application programs.

In this section, we will discuss step 1 in detail. Steps 2 and 3 will be presented in the subsequent sections.

Presidential Data

The raw data available in the Presidential data were divided into five groups:

1. Personal data on the Presidents: name, birthdate, state-born-in, height, party, college, ancestry, religion, occupation, date-of-death, cause-of-death, father, mother, wife, date-of-marriage, number-of-children, election-year, administration-number, Congress-number.
2. Data for each Presidential Administration: administration-number, inauguration date, president, vice-president, new-states-admitted to the Union during that administration.
3. Data on States: state-name, year-admitted, capital, area, area-ranking, population, population-ranking, number-of-electoral-votes, cities and city-populations.
4. Data on each Congress: congress-number, the major-parties and the number of their senators in the Senate, the major-parties and the number of their representatives in the House.
5. Data on each election: year, winner, winning-party, winner's total-votes, loser, losing-party, loser's total-votes.

A detailed description of the Presidential data base can be found in Ref. 8.

Relationships among Presidential Data Groups

The relationships among the Presidential data groups can be divided into two categories: the relationships between a group and its elements, and the relationships among groups.

Relationships between group and its elements — The relationships between each group (as mentioned in the above section) and its elements can be described in Table 1.

Table 1. Relationships between group and its elements.

Element	Degree of relationship
A. President	
Name, birthdate, state-born-in, height, party, college, ancestry, religion, date-of-death, cause-of-death, father, mother	1:1
Occupation	1:m, $m \geq 0$
Wife, date-of-marriage, number-of-children	1:m, $m \geq 0$
Election-year	1:n, $n \geq 1$
Administration-number	1:n, $n \geq 1$
Congress-number	1:n, $n \geq 1$
B. Administration	
Administration-number, inauguration-date, president, vice-president	1:1
New-state-admitted	1:m, $m \geq 0$
C. States	
State-name, year-admitted, capital, area, area-ranking, population, population-ranking, electoral-votes	1:1
City, city-population	1:n, $n \geq 1$
D. Congress	
Congress-number	1:1
Major-party in Senate, number-of-senators	1:n, $n \geq 1$
Major-party in House, number-of-representatives	1:n, $n \geq 1$
E. Election	
Year, winner, winner's party, winner's votes	1:1
Loser, loser's party, loser's votes	1:n, $n \geq 1$

Relationships among Groups — The relationships among the five data groups are described in Fig. 1.

STDS Implementation of Presidential Data Base

When the constrained definition of the extended sets allowed by STDS is given, the primary design decisions in developing the presidential data base are involved in the design of the n-tuples. One of the primary characteristics of n-tuples is that they have a fixed length. Consequently, one of the first design criteria that must be applied to the Presidential data is the factoring out of all repeating data. This step yields 13 sets:

ST. PRES = [pres#, pkey, last-name, first-name, initial, month-born, day-born, year-born, native-state-key, state-born-in, height, party, college, ancestry, religion, month-died, day-died, year-died, cause-of-death, father's name, mother's-name]

ST.OCCUP = [pres#, occupation]

ST. SPOUSE = [pres#, spouse-name, month-married, day-married, year-married, number-of-children]

ST.EYEAR = [pres#, election-year]

ST.PADM = [pres#, administration-number]

ST.CONG = [pres#, congress-number]

ST.ADMIN = [administration-number, month-inaug, day-inaug, year-inaug, pkey, vp-first-name, vp-last-name]

ST.NSTATE = [administration-number, new-state-admitted, state-key]

ST.STATE = [state-key, state-name, year-admitted, capital, area, area-ranking, population, pop-ranking, electoral-votes]

ST.CITY = [state-key, state-name, city-name, city-population]

ST.SENATE = [congress-number, party, number-of-senators]

ST.HOUSE = [congress-number, party, number-of-representatives]

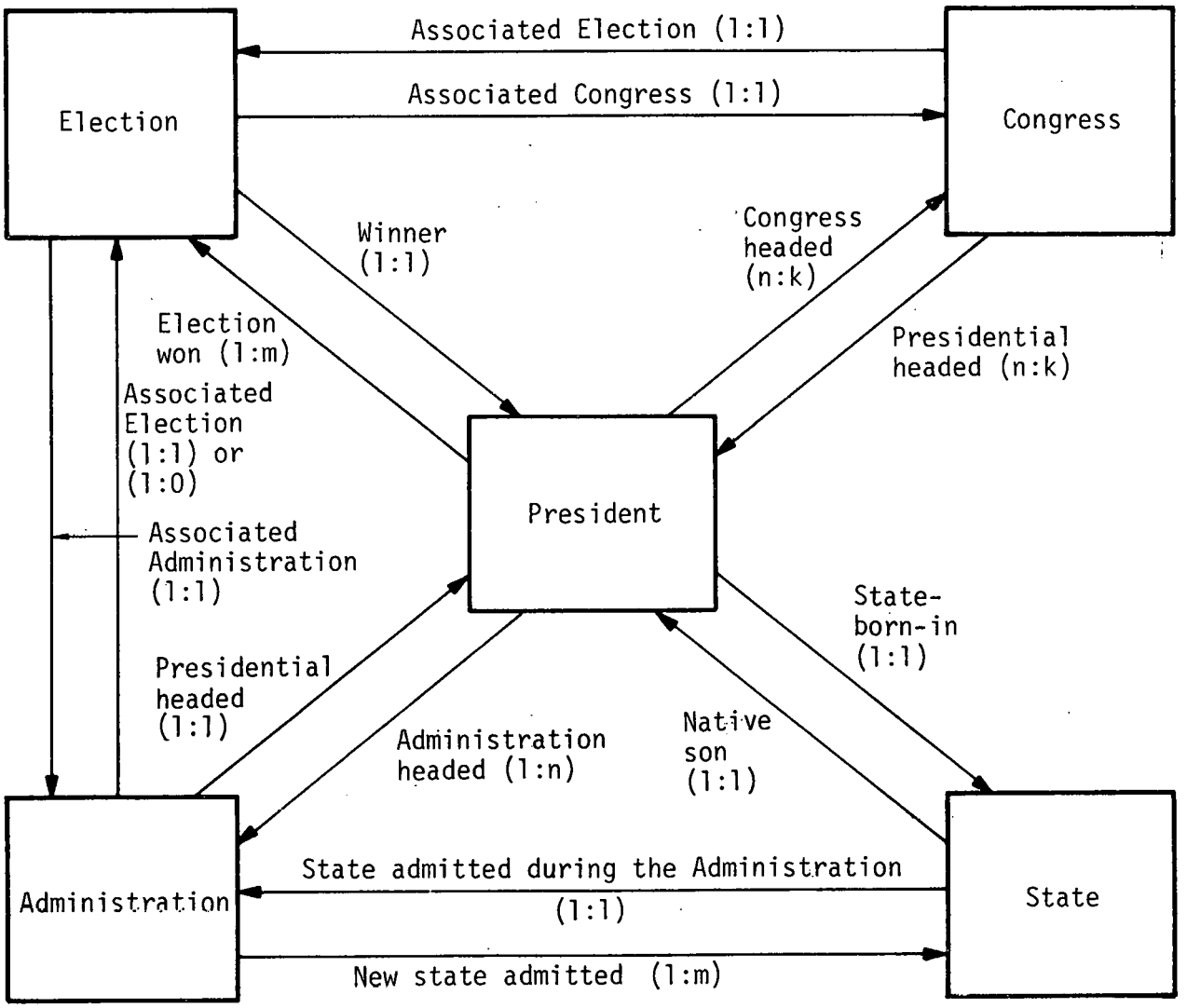
ST.ELECT = [election-year, year, winner, winner's party, winner's-votes, loser, loser's party, loser's-votes]

Partial listings of the contents of these sets are found in Appendix B.

To make this example more concise, the ST.EYEAR and ST.ELECT sets could be modified to factor out repeating winner information:

ST.EYEAR = [pres#, election-year, year, winner, winner's party, winner's electoral-votes]

ST.ELECT = [eyear, loser, loser's party, loser's-electoral-votes]



Note that the integer $m \geq 0$ and the integers $n,k \geq 1$.

Fig. 1. Relationships among five data groups.

The relationships between sets are symbolically represented by data values. For example, the many-to-many relationship between congress and president is represented using the key fields of the sets ST.PRES (pres#) and ST.CONG (congress-number) dynamically at query time. To find all congresses associated with a president, XPAN8 ST.PRES with ST.CONG; to find all congresses associated with a president, XPAN8 ST.CONG with ST.PRES.

The relationships in the STDS implementation are diagrammatically represented in Fig. 2. Notice that factoring out all repeating data into additional sets has decomposed M:N relationships into 1:n and m:1 relationships. Thus the simple operation of factoring out repeating data results in straightforward data base design.

As mentioned previously, the STDS-I system does not provide a symbolic naming facility. In fact, if the domain declaration is 8-bit, first-name in the set ST.PRES is referred to as the field that is 10 domains long and begins at domain 25.

The set listings in the Appendix B show that all the set data are character strings. Obviously, the numeric fields could be compacted by using binary representations. Another improvement would be to replace all state names with state-identifiers, thereby compacting state names. A new set which establishes a correspondence between state identifier and state name could also be constructed.

Data Base Design in STDS is a fairly straightforward process of factoring out repeating data. Once the sets have been designed, the raw data must be converted into the set formats. For the presidential data base, the raw data (Appendix C) was loaded into an STDS format by a FORTRAN program (Appendix D). After the sets have been filled with data, they may be manipulated. The next section will show how such data can be manipulated.

SOME APPLICATION PROGRAMS

After the data base has been loaded into the STDS system, a series of application programs can be written to retrieve and to update the data base. There application programs can be written and stored in a command file and later activated through the command QSFILE. They can also be typed in one by one at the time using the STDS interactive interface. For the purpose of this discussion, the latter is presented in this section; however, for human and machine efficiency the former method is recommended.

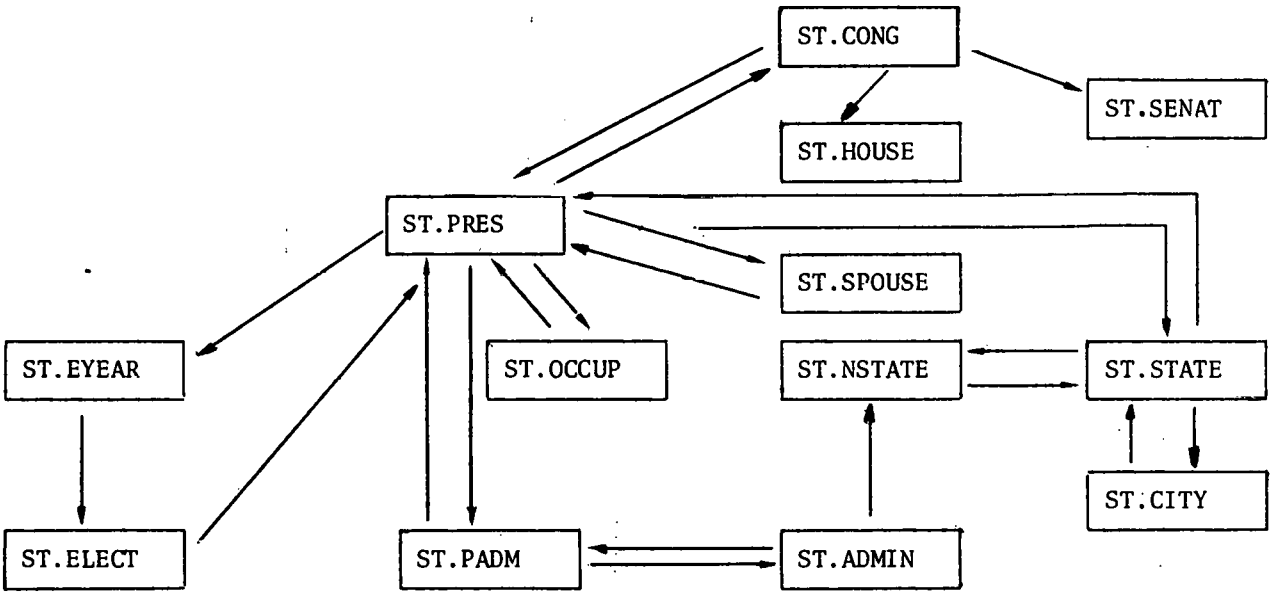


Fig. 2. Relationships in STDS implementation.

Some examples presented in this section are also found in Ref. 8. A comparison of the STDS and the DBTG approach based on one of these examples is presented in a later section, "Advantages of STDS."

Sample Retrieval Programs

Four examples are presented in this section. In each example, the purpose of the program is first stated, and then a set-approach algorithm is described in detail. Finally, an STDS-I program is presented.

Example 1 — The first example presented here is the same example used by R. Taylor in Ref. 8. The problem is to find all states that have more than one president as a native son, and then to print out the names of those states together with the numbers of the presidents who were born in that state.

Our algorithm* for solving this problem is to form a set of tuples [state-born-in, president's-last name, first-name, initial] from the ST.PRES and then to tabulate the number of presidents born in the same state. The result from this tabulation is a set of pairs of the state-born-in and the number of presidents. A listing of this set is the final result to this problem. An example of an STDS-I program for implementing the above algorithm is presented in Fig. 3.

All elements in an STDS set are sorted by the lexicographical order of their first domain. Therefore, the final listing is in alphabetical order by the names of the states. If the order by the president's number is desired, one may simply switch the position of the pair [state-born-in, number-of-presidents] to that of the pair [number-of-presidents, state-born-in], and then print out the set.

Example 2 — The second example is to list all the occupations together with the presidents' names in the alphabetical order of occupations.

This problem involves acquiring access to two sets, namely the sets ST.PRES and ST.OCC, because the set ST.OCC contains only the president's number (first, second...) but not the president's name.

Our algorithm for solving this problem involved the following three steps:

1. Extract a set of [pres#, president's-last-name, first-name, initial] tuples from the set ST.PRES, and call it set P.
2. XPAN8 (Join) the set ST.OCC together with the set P based on the president's number in both sets. The result of this operation is a

* Those interested in a comparison between DBTG and STDS approaches may compare the difference between the algorithm presented here and the one in R. Taylor's paper.[8]

STDS-I Commands	Commentary
GET (P,'ST.PRES')	P <= ST.PRES
XSET (X)	
1, 83, 10	state-born-in
11, 13, 30	president's last-name, first-name, initial
$ENDFILE	
RMIX8 (1, P, X, Q, 1)	Q ← [state-born-in, president's-name]
XTAB (10)	
DMR (1, 1, Q, R)	R ← [state-born-in, number-of-president]
LIST (R, 1, 1000, '(2X, 10A1, 4X, I4)')	

Fig. 3. STDS-I commands for Example 1.

set of [pres#, president-occupation, president's-last name, first-name, initial] tuples. Let this set be named PC.

3. RMIX8 (Project) the set PC into a set of [president-occupation, president's-last-name, first-name, initial] tuples. A listing of this set is the result of this problem.

A portion of an STDS-I program for this algorithm is presented in Fig. 4.

Note that, in the result of the above program, the occupations are listed all the time, even if they are repeated continuously. In some cases, one may want to list the occupations only when they first appear in the list. This problem may be solved by using a FORTRAN program to list the resulting set from STDS in any desired format. It is not recommended that this problem be solved by using STDS commands; to solve this problem in STDS, one would have to repeat the following procedure a number of times:

- Get one occupation from the set ST.OCC, and then find all the presidents who have this occupation.

The number of repetitions, N, is equal to the number of different occupations in the set ST.OCC. This procedure by itself is not a simple one (a similar example is presented in the next problem). Adding to the complexity is the fact that, each time this procedure is invoked, gaining access to the set ST.PRES is required at least once. Thus, a total of N accesses to the set ST.PRES would be required.

A good rule of thumb in programming with STDS is to remember that the basic unit in STDS is a set (e.g., a file) but not an element (e.g., record). To best utilize the power of STDS, one should retrieve information through sets, but not through elements.

Example 3

The purpose of this example is to illustrate the retrieval of information from an STDS on an element basis. As noted in the previous example, it is not recommended that information be retrieved that would require the traversal of one element at a time in an STDS set.

The problem illustrated by this example is to find all the Congresses served by a given president, and then to print out the given president's number followed by the Congressional term, and the number of senators and the number of members of the House of Representatives in each party.

Our algorithm is, first, to find the president number of the given president. Next, we use this president number to construct a set of all Congresses with whom he served. Then we extract Congress numbers one by one

STDS-I Commands	Commentary
GET (PRES,'ST.PRES')	PRES <= ST.PRES
XSET (X)	
1, 1, 2	pres#
3, 13, 30	president's last_name, first_name, initial
$ENDFILE	
RMIX8(1,PRES,X,P,1)	P ← [pres#, president's name]
GET (OCC,'ST.OCC')	OCC <= ST.OCC
XPAN8 (2, OCC, PRES, PC, 1)	PC ← [pres#, occupation, president's name]
XSET (Y)	
1, 3, 40	occupation, president's name
$ENDFILE	
RMIX8 (1, PC, Y, R, 1)	R ← [occupation, president's name]
LIST (R, 1, 1000 '(2X, 10A1, 4X, 30A1)')	

Fig. 4. STDS-I commands for Example 2.

from this set, and use this Congress number to get the Congress term and the number of senators and the number of the House of Representatives in each party. More detailed procedures are presented as follows:

1. For the given president name, extract the corresponding president number from the set ST.PRES.
2. Use the president number extracted from 1, to construct a set of Congresses with whom he served.
3. For each Congress number in the above set perform the following steps:
   (a) Based on the given Congress number, extract a subset from the set ST.SENATE that contains only the given Congress number.
   (b) Restrict (Project) the [party, number-of-senators] out from the above set and print the resulting set.
   (c) For the same Congress number, extract a subset from the set ST.HOUSE that contains only the given Congress number.
   (d) Restrict (Project) the [party, number-of-representatives] out from the above set and print the resulting set.

An STDS-I program for the above algorithm is presented in Fig. 5.

Example 4 — The purpose of this example is to show the power of STDS through a complex problem. The problem is to find all presidents such that the "majority party" in the Congress is different from the president's party. We defined a majority party in the Congress as the majority party in both the Senate and the House of that Congress. Because the information about the Senate and the House are stored in two separate sets (ST.SENATE and ST.HOUSE), we have to find the majority party of each Senate and that of each House, and then we have to determine the majority party in the Congress, if any. After finding the majority party in the Congress, we could find the president it served and compare the president's party with the congressional party. The operations required to accomplish this query are presented in Figs. 6a and 6b.

Sample Update Programs

In addition to the retrieval of information from a data base, updating a data base is another primary function of the data base management system. The purpose of updating a data base is to keep the information in a data base current. An example of this is the updating of the Presidential Data Base in order to incorporate a new state just admitted to the Union. Another example of updating the Presidential Data Base is the addition of the names of senators

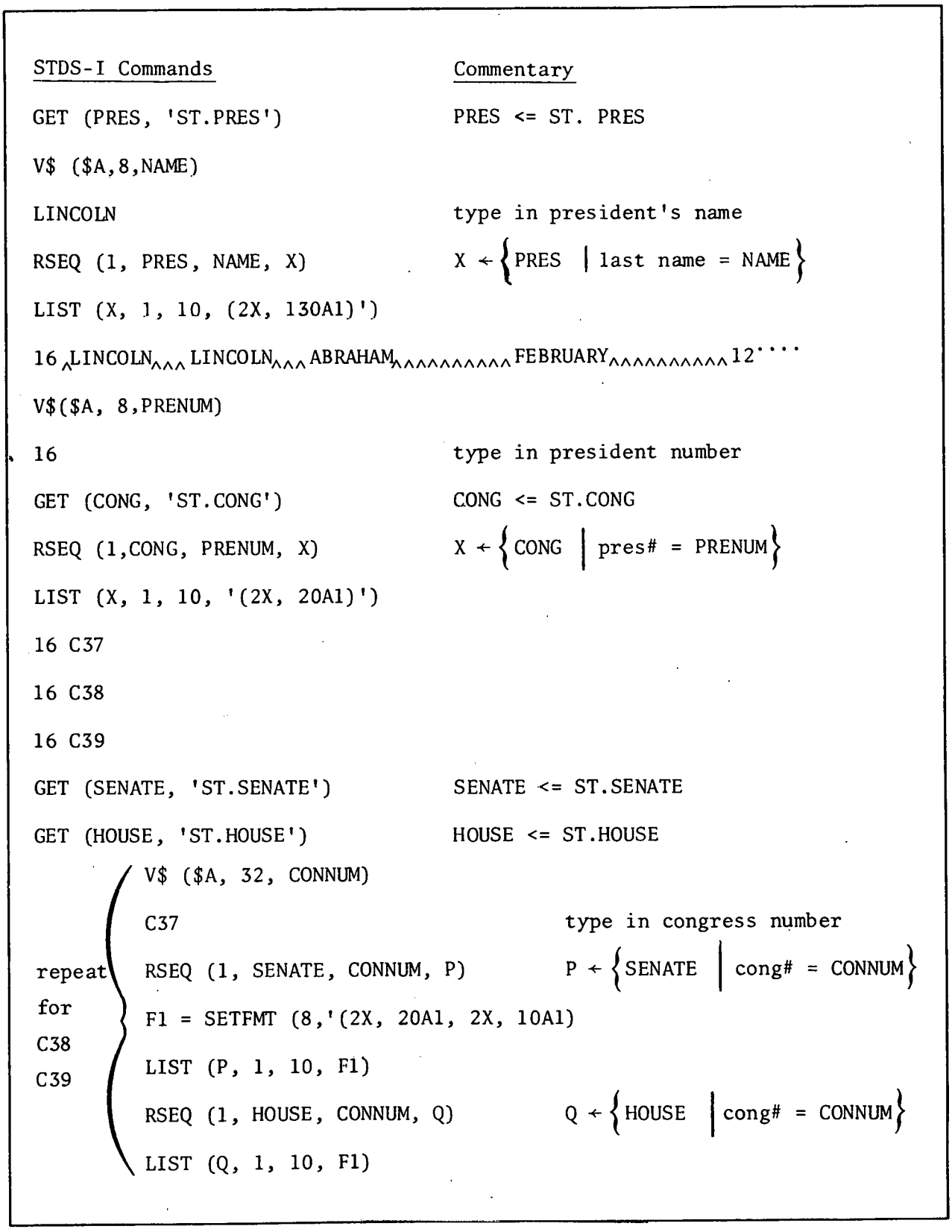


Fig. 5. STDS-I commands for Example 3.

STDS-I Commands	Commentary
: (a) Find a set of majority parties in each Senate	
GET (SENATE, 'ST.SENATE')	SENATE <= ST. SENATE
XSET (X)	
1, 1, 4	cong#
5, 15, 10	number-of-senators
$ENDFILE	
XSET(Y)	
1, 40, 0	
$ENDFILE	
TABF (4, Y)	
RMIX8 (1, SENATE, X, SMAX, Ø)	SMAX ← [cong#, highest-number-of-senator]
XSET (Z)	
1, 1, 4	cong#
5, 15, 10	# of senators
15, 5, 10	party
$ENDFILE	
RMIX8 (1, SENATE, Z, XSENAT,Ø)	XSENAT ← [cong#, # of senators, party]
RS8 (1, XSENAT, SMAX, SMAJ)	SMAJ ← [cong#, highest-number-of-senators, party]
: (b) Find a set of majority parties in each House	
GET (HOUSE, 'ST.HOUSE')	HOUSE <= ST.HOUSE
TABF (4, Y)	
RMIX8 (1, HOUSE, X, HMAX, Ø)	HMAX ← [cong#, highest-number-of-representatives]
RMIX8 (1, HOUSE, Z, XHOUSE, Ø)	XHOUSE ← [cong#, # of representatives, party]
RS8 (1, XHOUSE, HMAX, HMAJ)	HMAJ ← [cong#, highest # of rep., party]
: (c) Find a set of majority parties in both the Senate and the House	
XPAN8(4, SMAJ, HMAJ, CMAJ,2) :	CMAJ ← [cong#, # of maj. party's senators, , maj. party, # of maj. party's rep., maj party]
LEGL (CMAJ, 15, 10, 35, 10, SL, SE, SG) :	compare majority party in the Senate and the House
XSET (X)	
1, 1, 4	cong#
5, 15, 10	party

Fig. 6a. First half of STDS-I commands for Example 4.

```
$ENDFILE

RMIX8 (1, SE, X, MAJ, 1)                    MAJ ← [cong#, maj.-party in both
:                                                 the Senate and the House]

: (d) Construct a set of majority parties in the Congress and the president
      they served

GET (CONG, 'ST.CONG')                       CONG <= ST.CONG

XSET

1, 3, 4                                     cong#

5, 1, 2                                     pres#

$ENDFILE

RMIX8 (1, CONG, XCONG, 1)                   XCONG ← [cong#, pres#]

XPAN8 (4, XCONG, MAJ, MAJ CON, 10)          MAJCON ← [cong#, pres#, maj.-party]

XSET (X)

1, 5, 2                                     pres#

3, 1, 4                                     cong#

7, 7, 10                                    maj.-party

$ENDFILE

RMIX8 (1, MAJCON, MAJOR, 1)                 MAJOR ← [pres#, cong#, maj-party]

:

: (e) Find a set of president's names and their parties

:

GET (PRES, 'ST.PRES')                       PRES <= ST.PRES

XSET (X)

1, 1, 2                                     pres#

3, 13, 30                                   president's last-name, first-name, init.

33, 103, 10                                 president's party

$ENDFILE

RMIX8 (1, PRES, X, PPARTY, 1)               PPARTY ← pres#, pres-name, pres-party]

:

: (f) Find a set of majority parties in the Congress which is not the president's
      party

XPAN8 (2, PPARTY, MAJOR, PCPART, 1)         PCPART ← [pres#, pres-name, pres-party,
                                                      cong#, maj-party]

LEGL (PCPART, 33, 10, 47, 10, SL, SE, SG)   compare pres-party and maj-party

UN (SL, SG, RESULT)                         RESULT is a subset of PCPART where
                                             pres party different with major
                                             party

LIST (RESULT, 1, 999, '(56A1)')
```

Fig. 6b. Second half of STDS-I commands for Example 4.

and congressmen, as well as the names of the newly elected president and his administration.

Updating an STDS data base involves the change of the contents of some sets in the data base. In order to change a set in STDS one has to create a differential set[12] for the set to be changed. A differential set is a set of all updated elements of a given set. For example, a differential set of a set of states, ST.STATE, may be a set of all newly admitted states. To add these new states into the set ST.STATE is to "union" the set ST.STATE and its differential set. Similarly, to delete some elements from a set is to subtract its differential set from the original set. As for the content-modification or replacement of some elements in a set, one can simply create a differential set consisting of both the elements to be replaced and the elements to replace them; at that point, the symmetric difference between the original set and its differential set equals the updated set. For example, the original set A = {a,b,c} and the updated set is expected to be B = {a,d,c}. One can create a differential set D = {b,d}; then the summetric difference between the sets A and D is equal to the updated set; i.e., $A \Delta D = \{a,d,c\} = B$.

Differential sets may be used as the temporary sets for updating the data bases. They can also be used as a permanent "errata list" for the data base in the following sense. Rather than update a data base each time a change is desired, a small collection of modified records is maintained on the differential sets. If the changes to the data base continue, the differential sets grow to a sufficient length that the updating costs become justifiable, at which time a physical update to the data base may be performed. This method significantly reduces the update costs and has been widely used in many data base applications.[12] STDS is one of the best structures for implementing the concept of the differential sets.

In this section we will illustrate two examples of updating the STDS-I Presidential data base, one using physical updating and the other using differential sets.

Example 5. — To Admit a New State — Referring to the discussion of the Presidential data base, we recall that to enter a new state into the data base we have only to know the administration number under which the state is admitted. No other set except ST.NSTATE, ST.STATE, and ST.CITY in the data base need be affected by the addition of a new state. For the sake of simplicity, we assume that the area ranking and the population ranking have not been changed by the addition of the new state.

Our algorithm to admit this new state is to create three state sets that contain the information of the new state with respect to the three sets, ST.NSTATE, ST.STATE, and ST.CITY. Then, we replace these three old sets, respectively, by the union of the old set and its corresponding new state set. An STDS-I program for this example is presented in Fig. 7.

Example 6. — To Update Presidential Data Base after Each General Election — After each general election, a new president, a new administration, a new congress, and new election results will be added into the data base. As may be recalled from the section "Presidential Data Base," ten out of the total of thirteen sets have to be updated. The majority of these updates require simply inserting one or two new elements into an existing set, but this insertion involves the creation of a new set (physical file), copying the entire old set plus the new elements into the new set, and then destroying the old set. The cost of these steps may be very expensive, and thus the concept of differential sets may be beneficial to the overall system operation.

To implement the concept of differential sets, one has to create a small data base containing all new records from the election.

Note that this differential data base is considerably smaller than the original data base and therefore the creation cost and the future updating costs are greatly reduced. Some other advantages for having such a small differential data base are presented in Ref. 12. An example of the STDS-I program for creating this differential data base is presented in Figs. 8a and 8b.

ADVANTAGES OF STDS

In this and the following section the advantages and the potential problems of the STDS system are presented based on the experiences and opinions of the authors.

In the design of an STDS data base, one is immediately impressed by the simplicity of its design task. The STDS uses the key-values to indicate the relationships among sets, and the actual linkages of these relationships are formulated at query-time. This approach eliminates the problem of deciding the physical linkages at the design phase, and leaves only one task to the designers, namely the partitioning of a data base into a collection of minimal relevant sets.

STDS-I Commands	Commentary
: (a) Update ST.NSTATE	
DATA (NUNSTA, 8, 24, '(24A1)'	Read in NU.NSTATE
GET (NSTATE, 'ST.NSTATE')	NSTATE <= ST.NSTATE
UN(NSTATE, NUNSTA, NSTATE)	NSTATE ← NSTATE ∪ NUNSTA
PUT (NSTATE, 'ST.NSTATE')	NSTATE => ST.NSTATE
: (b) Update ST.STATE	
DATA (NUSTATE, 8, 90m '(90A1)', 'NU.STATE')	Read in NU.STATE
GET (STATE, 'ST.STATE')	STATE <= ST.STATE
UN (STATE, NUSTATE, STATE)	STATE ← STATE ∪ NUSTATE
PUT (STATE,'ST.STATE')	STATE => ST.STATE
: (c) Update ST.CITY	
DATA (NUCITY, 8, 40, '(40A1), 'NU.CITY')	Read in NU.CITY
GET (CITY,'ST.CITY')	CITY <= ST.CITY
UN (CITY, NUCITY, CITY)	CITY ← CITY ∪ NUCITY
PUT (CITY, 'ST.CITY')	CITY => ST.CITY

Fig. 7. STDS-I commands for Example 5.

STDS-I Commands (Comments are prefaced by :)

```
: (a) Create a differential set D.PRES for the set ST.PRES
DATA (NUPRES, 16, 101, '(101a2)", 'NU.PRES')

PUT (NUPRES, 'D.PRES')

: Future reference to ST.PRES has to be changed to (PRES Δ DPRES)

: (b) Create a differential set D.OCC for the set ST.OCC
DATA (NUOCC, 32, 3, '(3A4)', 'NU.OCCUP')

PUT (NUOCC, 'D.OCCUP')

: Future reference to ST.OCCUP has to be changed to (OCCUP Δ DOCCUP)

: (c) Create a differential set D.SPOUSE for the set ST.SPOUSE
DATA (NUSPOU, 32, 13, '(13A4)', 'NU.SPOUSE')

PUT (NUSPOU, 'D.SPOUSE')

: Future reference to ST.SPOUSE has to be changed to (SPOUSE Δ DSPOUSE)

: (d) Create a differential set D.EYEAR for the set ST.EYEAR
DATA (NUEYEA, 32, 2, '(2A4)', 'NU.EYEAR')

PUT (NUEYEY, 'D.EYEAR')

: Future reference to ST.EYEAR has to be changed to (EYEAY Δ DEYEAR)

: (e) Create a differential set D.PADM for the set ST.PADM
DATA (NUPADM, 16, 3, '(3A2)', 'NU.PADM')

PUT (NUPADM, 'D.PADM')

: Future reference to ST.PADM has to be changed to (PADM Δ DPADM)
```

Fig. 8a. First half of STDS-I commands for Example 6.

```
: (f) Create a differential set D.CONG for the set ST.CONG
DATA (NUCONG, 16, 3, '(3A2)', 'NU.CONG')

PUT (NUCONG, 'D.CONG')

: Future reference to ST.CONG has to be changed to (CONG Δ DCONG)

: (g) Create a differential set D.ELEC to the set ST.ELEC

DATA (NUELEC, 8, 75, '(75A1)', 'NU.ELEC')

PUT (NUELEC, 'D.ELEC')

: Future reference to ST.ELEC has to be changed to (ELEC Δ DELEC)

: (h) Create a differential set D.ADMIN for the set ST.ADMIN

DATA (NUADMI, 8, 63, '(63A1)', 'NU.ADMIN')

PUT (NUADMI, 'D.ADMIN')

: Future reference to ST.ADMIN has to be changed to (ADMIN Δ DADMIN)

: (i) Create a differential set D.SENATE for the set ST.SENATE

DATA (NUSENA, 32, 6, '(6A4)', 'NU.SENATE')

PUT (NUSENA, 'D.SENATE')

: Future reference to ST.SENATE has to be changed to (SENATE Δ DSENATE)

: (j) Create a differential set D.HOUSE for the set ST.HOUSE

DATA (NUHOUS, 32, 6, '(6A4)', 'NU.HOUSE')

PUT (NUHOUS, 'D.HOUSE')

: Future reference to ST.HOUSE has to be changed to (HOUSE Δ DHOUSE)
```

Fig. 8b. Second half of STDS-I commands for Example 6.

The use of key values to indicate the relationships among sets also provides a higher degree of data independence, which allows the user to update (i.e., to add, to delete, or to replace) relations independently of one another without tedious modifications of pointers which link one set to another set. Key values also allow the user to restructure a data base into any number of sets in order to reduce the physical storage requirements or the processing time of a data base. The set theoretical approach offers further potential storage reduction through the representation of a set in terms of its implicit membership condition (e.g., a function or a statement that defines the membership of a set). However, this ability for storing an abstract set has not been implemented in either version of the current STDS systems.

The representation of a data base as a collection of sets in STDS-I encourages the manipulation of data on a large aggregate of data. It is sensible to perform the same operations on each element in a set at a time because each set contains all relevant data. To illustrate this point further, let us compare the STDS-I and the DBTG implementations[8] of the same problem as presented in Example 1. The DBTG implementation requires a loop of nine statements, and the STDS-I implementation requires only two statements without a loop. The basic difference is that the DBTG approach operates on one record at a time, while the STDS approach operates on a set of relevant records at one time. It is also worth noting that in the STDS approach, the user does not have to be bothered with traversing pointer chains (e.g., DBTG sets) or any other physical linkages.

The extended set operations provided by the STDS-I are a powerful set of data manipulation operations equivalent to the power of the relational algebra. This set of operations could be used as a basis for a high level nonprocedural language for retrieving information from a collection of sets. It could also be used as a basis for implementing a high-level relational calculus language.

The STDS-I provides two ways to access these operations: one through an interactive interface, and the other through FORTRAN or COBOL call statements. The ability to access these operations through FORTRAN (or COBOL) call statements is required for some classes of queries that cannot be directly answered by using STDS commands (as shown in Example 2). In addition, FORTRAN (or COBOL) provides a simple extension to new data manipulation functions and

a linkage to any application program, such as a graphic or statistical program.

SHORTCOMINGS OF STDS

Although the STDS system has many advantages, it also has some disadvantages. The principal disadvantage of STDS-I is its low-level user interface. This interface, as previously indicated, does not allow the user to symbolically name domains. Instead, the user references a logical domain by a physical domain index (e.g., byte offset) and a length (number of bytes). Not only is this cumbersome, but it forces the user to be cognizant of the data representation. A related problem, domain declaration, requires the user to stipulate the number of domains in terms of physical quantities, in this case, bytes, halfwords, and fullwords.

STDS-OS does permit the user to reference logical domains symbolically. In STDS-OS, a uniform domain declaration is presented to the user; therefore, physical quantities used to store the address data are transparent to the user. Although STDS-OS permits naming of components of sets (columns), its principal disadvantage is its orientation toward operations within sets but not among sets. Of primary concern, the XPAN8 operation (JOIN) is not present. The lack of the XPAN operation makes it impossible to perform queries that encompass elements from more than one set.

Certain operations require that a particular domain be the (physically) first domain before the operation is performed. This requirement obliges the user to perform a cumbersome series of operations:

1. XSET command to express the rearrangement
2. RMIX8 command to do the rearranging
3. XPAN8 operation
4. Another XSET
5. Another RMIX8 to transform the set back to its original orientation.

The shortcomings of the user interface are also demonstrated by the primitive report writer facilities. The facilities in STDS-OS (e.g., RPG) scarcely compare with commercial report writers; STDS-I is void of any such features.

Similar to the relational systems, the user is burdened with some of the same problems that face the relational DBMS community. Two chief problems are normalization and procedural queries. Normalization is the process of

constructing n-tuples in such a manner that no repeating data, functional dependencies or transitive dependencies exist within an n-tuple. In STDS, this process is left up to the user. Procedural queries, such as those that print out the following report, are difficult to perform (see Examples 2 and 3):

Adams J
    Butcher
    Baker
    Candlestick maker

Adams JQ
    Fireman

The difficulty stems from the fact that in STDS and in relational systems, one works with sets, not just with records.

There are some miscellaneous disadvantages of STDS-I, such as not handling variable length data, not providing recovery or rollback facilities, and no security facilities (STDS-OS does permit passwords on sets, however).

Finally, STDS does not present to the user all the power of Extended Set Theory that would distinguish it from a relational system. The principal difficulty is the inability to have sets of sets. This inability hinders one's ability to compact data, to have automatic differential sets, and to have abstract sets — sets where a function is stored to generate data values rather than the data instances itself. In its present form, STDS does not provide a user with more capabilities than a relational system.

CONCLUSION

STDS provides a user with a very flexible and extremely powerful tool with which complex manipulation of data can be performed rapidly. In spite of its flexibility, STDS-I does not provide the user with a sufficiently "user-friendly" interface to allow noncomputer scientists to easily work with a data base. STDS-OS is a step in the right direction, but it currently has insufficient power to handle relationships between sets.

The potential capabilities of the Extended Set Theory seem to be very powerful and attractive; however, the current implementations do not provide the user with all the capabilities of Extended Set Theory. Almost all the

features (such as sets of sets and abstract sets) that would distinguish STDS from relational systems are not provided.

Because of its low-level nature, STDS does not have a sufficiently "user-friendly" interface for unsophisticated users. It is, however, a system that appears to be of interest to designers of relational data base management systems, and could possibly provide a more efficient means of implementing a relational system.

Despite some drawbacks in STDS, the Extended Set Theory has good potential, and, therefore, we recommend that:

1. A "user-friendly" interface be developed
2. Distinctive features of the Extended Set Theory be implemented.

Should these two additional features be incorporated into STDS, it would be a very attractive system.

APPENDIX A.

Complete Set of STDS-I Commands

Input/Output:

ATOS	transforms an array to a set
CORE	transfers a set from peripheral storage into core
DASD	specifies whether sets are forced to core or disk
DATA	allows formatted entry of data
ENTER	constructs a set
FREE	releases a set from the universe
GARRAY	constructs a set from an array
GBLOCK	transfers a block of elements from a set into an array
GET	enters a set from peripheral storage
PARRAY	writes a set to a file
PUT	stores a set
SGET	unscrambles and enters a set from peripheral storage
SINK	creates a permanently null set, or sink
SPUT	scrambles and stores a set
STODS	transfers a set to a dataset
UNIV	opens a universe
VOL	specifies the MTS volume on which a dataset is to be created
LIMIT	initializes a tape
TARRAY	retrieves a set which has been stored on tape in array form
TGET	retrieves a set which has been stored on tape in set form
TPUT	stores a set on tape in set form

Updating and Expanding:

DD	redefines the domain declaration of a set
DMR	performs a domain restriction
INDX	creates an index set (STDS*)
INDEX	creates an index set (FORTRAN)
INUM	converts portions of a set's components to integers
ISBMAX	redefines implicit set and buffer maxima
KDMR	creates a keyed domain
MIX8	rearranges the domains of an n-tuple using an index set
MULTU	performs a multiple update
MULTU1	performs multiple modification of bit-domains
REFMT	reformats a set
RMIX8	rearranges specified domains of an n-tuple
SETFMT	associates an index with a format
UPDT	updates specified values in a set
UPDT1	updates a one-bit domain in a set
XPAN8	expands the components of one set with domains from another set's components
ZPAK	removes specified zones from a set and packs the results

Restrictions and Set Operations:

BRSA	performs an arithmetic between restriction
BRSL	performs a logical between restriction
BRS8	performs a logical between restriction forcing eight-bit domains
DIN	returns a domain intersection of two sets
DRL	returns a domain relative complement of two sets
DSD	returns a domain symmetric difference of two sets
DUN	returns a domain union of two sets
IN	returns the intersection of two sets
LEGA	performs a arithmetic comparison of two components
LEGL	performs a logical comparison of two components
NRS1	performs a not restriction on one-bit domains
NRS8	performs a not restriction
RL	returns the relative complement of two sets
RSEQ	performs an equal-to restriction by a constant
RSGEA	performs an arithmetic greater-than-or-equal-to restriction by a constant
RSGEL	performs a logical greater-than-or-equal-to restriction by a constant
RSGTA	performs an arithmetic greater-than restriction by a constant
RSGTL	performs a logical greater-than restriction by a constant
RSLEA	performs an arithmetic less-than-or-equal-to restriction by a constant
RSLEL	performs a logical less-than-or-equal-to restriction by a constant
RSLTA	performs an arithmetic less-than restriction by a constant
RSLTL	performs a logical less-than restriction by a constant
RSNE	performs a not-equal-to restriction by a constant
RS1	performs a restriction on one-bit domains
RS8	performs a restriction
SD	returns the symmetric difference of two sets
SUBSET	returns a subset of a given set
UN	returns the union of two sets

Arithmetic Operations:

BITS	sets bits in a four-byte variable
CARTH	performs component arithmetic
SUM	calculates the sum, mean, minimum, maximum, and standard deviation for specified byte-domains of a set
V$	defines a constant and associates it with a symbolic name

Operation and Set Information:

CARD	returns the cardinality of a set
COMMANDS	lists the available STDS* commands
CSIZE	causes core size to be printed after every operation
DDEC	returns the domain declaration for a set
GETELM	retrieves an element of a set
INFO	prints information about a dataset
LFMT	prints a format
LIST	prints specified subsets of a set
LISTU	prints information about the sets in the current universe
LISTV	prints the contents of a variable
MINKEY	returns the minimum key length for a set
NDOM	returns the number of domains in a set
TIME	prints information about the amount of time used since the last call to TIME

Line Files:

QCALC	performs calculations in pseudo-registers
QSFILE	specifies a source file
QRETURN	causes return from a line file to STDS*
QSKIP	allows a skip operation in a line file
QSREG	initializes pseudo-registers

Utility Functions:

DONE	leaves STDS* and deletes all temporary sets
ECHO	sets echo on cr off
HISTOF	prints a histogram based on the floating point data in a given domain
MTS	returns control to the system
RES	restores the original (master) sink
SPRINT	specifies the system sink
STERR	specifies an error return entry point in a program which calls set operations
TAU	deletes duplicate elements
TAUOFF	allows implicit internal representation of sets
TAUON	establishes explicit representation for all sets manipulated by set operations
TEST	allows calls to user-supplied subroutines or functions

APPENDIX B.

Partial Listings of Sets in Presidential Data Base

```
 8 VANBUREN  VAN BUREN MARTIN              DECEMBER          5      1782NEW YORK   5FT. 6IN. DEMOCRATIC            DUTCH
DUTCH REF.JULY              24      1862ASTHMA     ABRAHAM   MARIA
 9 HARRISON  HARRISON  WILLIAM   H.        FEBRUARY          9      1773VIRGINIA   5FT. 8IN. WHIG      HAMP.-SYD.ENGLISH
EPISCOPAL APRIL             4       1841PHEUMONIA BENJAMIN  ELIZABETH
1Ø TYLER    TYLER     JOHN                MARCH             29     1790VIRGINIA   6FT. ØIN. WHIG      WM.-MARY  ENGLISH
EPISCOPAL JANUARY           18      1862FEVER      JOHN      MARY
```

Set ST.PRES

```
1 FARMER
1 SOLDIER
1 SURVEYOR
2 LAWYER
2 TEACHER
3 LAWYER
3 WRITER
4 LAWYER
5 LAWYER
5 SOLDIER
6 LAWYER
6 SECRETARY
7 LAWYER
7 SADDLER
7 SOLDIER
```

Set ST.OCCUP

```
2 ABIGAIL   OCTOBER     25      1764      5
3 MARTHA    JANUARY      1      1772      6
4 DOLLEY    SEPTEMBER   15      1794      Ø
5 ELIZABETH FEBRUARY    16      1786      3
6 LOUISA    JULY        26      1797      4
7 RACHEL    AUGUST      15      1791      Ø
8 HANNAH    FEBRUARY    21      18Ø7      4
9 ANNA      NOVEMBER    25      1795     1Ø
```

Set ST.SPOUSE

```
4 E18Ø6
4 E1812
5 E1816
5 E182Ø
6 E1824
7 E1828
7 E1832
8 E1836
9 E184Ø
```

Set ST.EYEAR

CAL	CALIFORNIA	185ØSACRAMENTO	158693	3	19221ØØØ	1	4Ø
COLORADO	COLORADO	1876DENVER	1Ø4247	8	2Ø48ØØØ	3Ø	6
CONN	CONN.	1788HARTFORD	5ØØ9	48	2959ØØØ	24	8
DELAWARE	DELAWARE	1787DOVER	2Ø57	49	534ØØØ	46	3
FLORIDA	FLORIDA	1845TALLAHASSE	5856Ø	22	616ØØØØ	9	14
GEORGIA	GEORGIA	1766ATLANTA	58876	21	4568ØØØ	15	12
HAWAII	HAWAII	1959HONOLULU	6424	47	778ØØØ	4Ø	4
IDAHO	IDAHO	1890BOISE	83557	13	7Ø5ØØØ	41	4

Set ST.STATE

KANSAS	KANSAS	KANSASCITY	121ØØ1
KANSAS	KANSAS	TOPEKA	119484
KANSAS	KANSAS	WICHITA	254698
KENTUCKY	KENTUCKY	LOUISVILLE	39Ø639
LA	LOUISIANA	BATONROUGE	1541ØØ
LA	LOUISIANA	NEWORLEANS	627525
LA	LOUISIANA	SHREVEPORT	16Ø535
MARYLAND	MARYLAND	BALTIMORE	939Ø24

Set ST.CITY

C16	FEDERALIST	7
C17	DEM-REP	44
C17	FEDERALIST	4
C18	DEM-REP	44
C18	FEDERALIST	4
C19	ADMIN.	26
C19	JACKSONIAN	2Ø

Set ST.SENATE

C16	DEM-REP	156
C16	FEDERALIST	27
C17	DEM-REP	158
C17	FEDERALIST	25
C18	DEM-REP	187
C18	FEDERALIST	26
C19	ADMIN.	1Ø5
C19	JACKSONIAN	97

Set ST.HOUSE

E1812	1812MADISON	DEM-REP	128CLINTON	INDEP.	89
E1816	1816MONROE	DEM-REP	183KING	FEDERALIST	34
E182Ø	182ØMONROE	DEM-REP	231ADAMS	INDEP.	1
E1824	1824J.Q. ADAMS	DEM-REP	84CLAY	INDEP.	37
E1824	1824J.Q. ADAMS	DEM-REP	84CRAWFORD	INDEP.	41
E1824	1824J.Q. ADAMS	DEM-REP	84JACKSON	INDEP.	99
E1828	1828JACKSON	DEMOCRATIC	178ADAMS	NAT-REP	83

Set ST.ELECT

```
 2 A3
 3 A4
 3 A5
 4 A6
 4 A7
 5 A8
 5 A9
 6 A1Ø
 7 A11
 7 A12
 8 A13
```

Set ST.PADM

```
 7 C21
 7 C22
 7 C23
 7 C24
 8 C25
 8 C26
 9 C27
1Ø C27
1Ø C28
11 C29
11 C3Ø
```

Set ST.CONG

A1	APRIL	3Ø	1789WASHINGT	JOHN	ADAMS
A1Ø	MARCH	4	1825ADAMSJQ	JOHN	CALHOUN
A11	MARCH	4	1829JACKSON	JOHN	CALHOUN
A12	MARCH	4	1833JACKSON	MARTIN	VAN BUREN
A13	MARCH	4	1837VANBUREN	RICHARD	JOHNSON
A14	MARCH	4	1841HARRISON	JOHN	TYLER
A15	APRIL	6	1841TYLER		
A16	MARCH	4	1845POLK	GEORGE	DALLAS
A17	MARCH	4	1849TAYLOR	MILLARD	FILLMORE

Set ST.ADMIN

```
A3Ø IDAHO       IDAHO
A3Ø MONTANA     MONTANA
A3Ø N.D.        ND
A3Ø S.D.        SD
A3Ø WASHINGTON  WASH
A3Ø WYOMING     WYOMING
A31 UTAH        UTAH
```

Set ST.NSTATE

APPENDIX C.

Partial Listings of Raw Input Data

•
•
•

STATES HAWAII HAWAII 1959HONOLULU 6424 47 77ØØØØ
4Ø 4 1HONOLULU 294194STATES IDAHO IDAHO
189ØBOISE 83557 13 7Ø5ØØØ 41 4 Ø
STATES ILLINOIS ILLINOIS 1818SPRING. 564ØØ 25 1Ø974ØØØ
4 26 3CHICAGO 355Ø4Ø4ROCKFORD 132109PEORIA
1Ø3162STATES INDIANA INDIANA 1816INDIANAP. 36291 38
5Ø67ØØØ 12 13 6INDIANAP. 476258GARY 17832Ø
FORTWAYNE 172594EVANSVILLE 144463SOUTH BEND 132445HAMMOND 111698
STATES IOWA IOWA 1846DES MOINES 5629Ø 26 2746ØØØ
25 9 2DES MOINES 2Ø6739CED. RAP. 1Ø3545STATES
KANSAS KANSAS 1861TOPEKA 82264 14 23Ø3ØØØ 29
7 3WICHITA 254698KANSASCITY 12191Ø1TOPEKA 119484

•
•
•

PRES HARRISON HARRISON BENJAMIN AUGUST 2Ø 1833
OHIO OHIO 5FT. 6IN. REPUBLICANMIAMI O. ENGLISH PRESBYT. 1
LAWYER MARCH 13 19Ø1PNEUMONIA JOHN ELIZABETH 2
CAROLINE OCTOBER 20 1853 2MARY APRIL 6
1896 1 1E1868 1A3Ø 2C51
C52 PRES MCKINLEY MCKINLEY WILLIAM JANUARY 29
1843OHIO OHIO 5FT.1ØIN. REPUBLICANALLEGHANY SCOT-IRISHMETHODIST
2LAWYER TEACHER SEPTEMBER 14 19Ø1ASSASSIN. WILLIAM
NANCY 1IDA JANUARY 25 1871 2 2
E1896 E19ØØ 2A32 A33 3C55 C56
C57 PRES ROOSEVET ROOSEVELT THEODORE OCTOBER 27
1858NEW YORK NEWYORK 5FT. 1Ø.REPUBLICANHARVARD DUTCH DUTCHREF.
2LAWYER PUB.OFF. JANUARY 6 1919RHEUMATISHTHEODORE
MARTHA 2ALICE OCTOBER 27 188Ø 1EDITH
DECEMBER 2 1886 5 1E19Ø4 2A34
A35 4C57 C58 C59 C6Ø PRES TAFT
TAFT WILLIAM H. SEPTEMBER 15 1857OHIO OHIO
6FT. ØIN. REPUBLICANYALE SCOT-IRISHUNITARIAN 1LAWYER MARCH
8 193ØDEBILITY ALPHONSO LOUISE 1HELEN JUNE
19 1886 3 1E19Ø8 1A36 2
C61 C62 PRES WILSON WILSON WOODROW DECEMBER
28 1856VIRGINIA VIRGINIA 6FT. ØIN.DEMOCRATICPRINCETON ENGLISH
PRESBYT. 2LAWYER TEACHER FEBRUARY 3 1924HEART DIS.
JOSEPH JESSE 2ELLEN JUNE 24 1885 3
EDITH DECEMBER 18 1915 Ø 2E1912 E1916
2A37 A38 4C63 C64 C65 C66

•
•
•

ELECTION E1848 1848TAYLOR WHIG 163 1CASS
DEMOCRATIC 127ELECTION E1852 1852PIERCE DEMOCRATIC 254
1SCOTT WHIG 42ELECTION E1856 1856BUCHANNAN
DEMOCRATIC 174 2FREMONT REPUBLICAN 114FILLMORE AMERICAN
8ELECTION E186Ø 186ØLINCOLN REPUBLICAN 18Ø 3
DOUGLAS DEMOCRATIC 12BRECKRIDGESOU. DEM. 72BELL CONSTIT.
39ELECTION E1864 1864LINCOLN REPUBLICAN 212 1
MC CLELLANDEMOCRATIC 21ELECTION E1868 1868GRANT REPUBLICAN
214 1SEYMOUR DEMOCRATIC 8ØELECTION E1872 1872
GRANT REPUBLICAN 286 1GREELEY DEMOCRATIC 66

•
•
•

```
                                                                    ADMIN
A2Ø       MARCH                 4      1857BUCHANAN          1JOHN       BRECKRIDGE
          3MINNESOTA MINN        OREGON    OREGON    KANSAS    KANSAS     ADMIN
A21       MARCH                 4      1861LINCOLN           1HANNIBAL   HAMLIN
          2W.VA.     WVA         NEVADA    NEVADA    ADMIN     A22        MARCH
          4       1865LINCOLN              1ANDREW   JOHNSON             ØADMIN
A23       APRIL                 15     1865JOHNSONA          Ø           1NEBRASKA
NEBRASKA  ADMIN      A24         MARCH                 4      1869GRANT              1
SCHUYLER  COLFAX                ØADMIN    A25         MARCH              4      1873
GRANT                 1HENRY     WILSON               1COLORADO COLORADO   ADMIN
A26       MARCH                 4      1877HAYES             1WILLIAM    WHEELER
          ØADMIN     A27         MARCH                 4      1881GARFIELD           1
CHESTER   ARTHUR                ØADMIN    A28         SEPTEMBER          2Ø     1881
ARTHUR                Ø         ØADMIN    A29         MARCH               4     1885
CLEVELAN              1THOMAS    HENDRICKS           Ø
```

```
                                                    CONGRESS  C7                2
FEDERALIST          14DEM-REP          18           2FEDERALIST         36DEM-REP
        69CONGRESS  C8                  2FEDERALIST          9DEM-REP          25
          2FEDERALIST          39DEM-REP          1Ø2CONGRESS  C9                2
FEDERALIST           7DEM-REP          27           2FEDERALIST         25DEM-REP
       116CONGRESS  C1Ø                 2FEDERALIST          6DEM-REP          28
          2FEDERALIST          24DEM-REP          118CONGRESS  C11               2
FEDERALIST           6DEM-REP          28           2FEDERALIST         48DEM-REP
        94CONGRESS  C12                 2FEDERALIST          6DEM-REP          3Ø
          2FEDERALIST          36DEM-REP          1Ø8CONGRESS  C13               2
FEDERALIST           9DEM-REP          27           2FEDERALIST         68DEM-REP
       112CONGRESS  C14                 2FEDERALIST         11DEM-REP          25
          2FEDERALIST          65DEM-REP          117CONGRESS  C15               2
FEDERALIST          1ØDEM-REP          34           2FEDERALIST         42DEM-REP
       141CONGRESS  C16                 2FEDERALIST          7DEM-REP          35
          2FEDERALIST          27DEM-REP          156
```

APPENDIX D.

Program to Load Raw Data into STDS Data Base

```
C       ***  SER4:PRES.TEST              <1Ø,Ø4,76>      ***
C
        SUBROUTINE TEST (NUM,SET)
        IMPLICIT INTEGER (A-X,Z,$)
        INTEGER B(6),C(13),INC(13)/13*Ø/,NB(6)/6*Ø/
        INTEGER STAT/'STAT'/,PRES/'PRES'/,ELEC/'ELEC'/,CONG/'CONG'/,ADMI/'ADMI'/
        LOGICAL*1 A(5ØØØØ),BA(32768),BB(32768),BC(32768),BD(32768),BE(32768,BF(32768)
        LOGICAL*1 XX(4),ZZ(4)
        INTEGER LC(13)/9Ø,4Ø,2Ø2,12,52,12,6,6,8Ø,7Ø,24,24,24/
        EQUIVALENCE(NB(1),NBA),(NB(2),NBB),(NB(3),NBC),(NB(4),NBD),(NB(5),NBE),(NB(6),NBF)
        EQUIVALENCE (Z,ZZ(1)),(X,XX(1))
        NUMAX=NUM*8Ø
        IF(NUM.GT.Ø) GO TO 1Ø1
        PRINT 1ØØ
 1ØØ    FORMAT('  *** TEST(CARD,ARRAY) ~ STATE,CITY,PRES,OCC,SPOUC,EYEAR,',
        *',PADM,CONG,ELEC,ADMIN,NSTATES,SENATE,HOUSE ***')
        RETURN
C
 1Ø1    L=Ø
        LL=Ø
        SW1=Ø
        SW2=Ø
        SW3=Ø
        SW4=Ø
        X=Ø
        CALL GBLOCK(1,SET,1,NUM,5ØØØØ,A(1))
        B(1)=$AD(BA)
        B(2)=$AD(BB)
        B(3)=$AD(BC)
        B(4)=$AD(BD)
        B(5)=$AD(BE)
        B(6)=$AD(BF)
        DO 1Ø2 I=1,13
 1Ø2    CALL $OUT(C(I),INC(I),LC(I),Ø,8,LC(I),2,2)
        PRINT 9ØØ1
 9ØØ1   FORMAT('~OUT')
C
 2ØØ    L=L+LL
        LL=Ø
        IF(L.GE.NUMAX) GO TO 6ØØØ
        ZZ(1)=A(L+1)
        ZZ(2)=A(L+2)
        ZZ(3)=A(L+3)
        ZZ(4)=A(L+4)
        IF(Z.EQ.STAT)  GO TO 1ØØØ
        SW1=SW1+1
        IF(Z.EQ.ELEC)  GO TO 2ØØØ
        SW2=SW2+1
        IF(Z.EQ.PRES)  GO TO 3ØØØ
        SW3=SW3+1
        IF(Z.EQ.ADMI)  GO TO 4ØØØ
        SW4=SW4+1
        IF(Z.EQ.CONG)  GO TO 5ØØØ
        PRINT 9ØØ9,Z,L,LL
 9ØØ9   FORMAT('Z,L,LL:',Z9,216)
        GO TO 6ØØØ
C
C                                       STATES[C(1)1, CITY(C(2)]
 1ØØØ   CALL IMVC(9Ø,BA,NBA,A,L+1Ø)
        NBA=NBA+9Ø
        LL=11Ø
        Z=Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        XX(4)=A(L+LL-1)
```

```
        Z=(X-24Ø)*1Ø+Z
 1ØØ1   IF(Z.EQ.Ø) GO TO 2ØØ
        DO 11ØØ I=1,Z
        CALL IMVC(2Ø,BB,NBB,A,L+1Ø)
        NBB=NBB+2Ø
        CALL IMVC(2Ø,BB,NBB,A,L+LL)
        NBB=NBB+2Ø
 11ØØ   LL=LL+2Ø
        GO TO 2ØØ
C
C                    PRES[C(3)], OCC[C(4)], SPOUCE[C(5)], EYEAR[C(6)], PADM[C(7)], CONG[C(8)]
 2ØØØ   IF(SW1.GT.1) GO TO 2ØØ1
        DO 2ØØ2 I=1,2
        CALL $EMPTY(B(I),INC(I),NB(I))
        CALL $SET(C(I),INC(I),1)
 2ØØ2   NB(I)=Ø
        CALL PUTD(C(1),'ST.STATE')
        CALL PUTD(C(2),'ST.CITY')
        PRINT 9ØØ2
 9ØØ2   FORMAT(' 1ØØØ')
C                                                PRES-C3
 2ØØ1   CALL IMVC(2,BA,NBA,SW1,2)
        NBA=NBA+2
        CALL IMVC(14Ø,BA,NBA,A,L+1Ø)
        NBA=NBA+14Ø
        LL=16Ø
        Z=Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 21ØØ
C                                                OCCUP-4
        DO 2Ø1Ø I=1,Z
        CALL IMVC(2,BB,NBB,SW1,2)
        NBB=NBB+2
        CALL IMVC(1Ø,BB,NBB,A,L+LL)
        NBB=NBB+1Ø
 2Ø1Ø   LL=LL+1Ø
 21ØØ   CALL IMVC(60,BA,NBA,A,L+LL)
        NBA=NBA+60
        LL=LL+7Ø
        Z=Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 22ØØ
C                                                SPOUCE-5
        DO 211Ø I=1,Z
        CALL IMVC(2,BC,NBC,SW1,2)
        NBC=NBC+2
        CALL IMVC(5Ø,BC,NBC,A,L+LL)
        NBC=NBC+5Ø
 211Ø   LL=LL+5Ø
C                                                EYEAR-6
 22ØØ   Z=Ø
        LL=LL+1Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 23ØØ
        DO 221Ø I=1,Z
        CALL IMVC(2,BD,NBD,SW1,2)
        NBD=NBD+2
        CALL IMVC(1Ø,BD,NBD,A,L+LL)
        NBD=NBD+1Ø
 221Ø   LL=LL+1Ø
C                                                PADMIN-7
 23ØØ   Z=Ø
```

```
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 24ØØ
        DO 231Ø I=1,Z
        CALL IMVC(2,BE,NBE,SW1,2)
        NBE=NBE+2
        CALL IMVC(4,BE,NBE,A,L+LL)
        NBE=NBE+4
 231Ø   LL=LL+1Ø
C                                                   CONG-8
 24ØØ   Z=Ø
        LL=LL+1Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 2ØØ
        DO 241Ø I=1,Z
        CALL IMVC(2,BF,NBF,SW1,2)
        NBF=NBF+2
        CALL IMVC(4,BF,NBF,A,L+LL)
        NBF=NBF+4
 241Ø   LL=LL+1Ø
        GO TO 2ØØ
C                                                   ELEC[C(9)]
 3ØØØ   IF(SW2.GT.1) GO TO 3Ø1Ø
        DO 3ØØ1 I=1,6
        CALL $EMPTY(B(I),INC(I+2),NB(1))
        CALL $SET(C(I+2),INC(I+2),1)
        PRINT 9ØØ3
 9ØØ3   FORMAT(' 2ØØØ')
 3ØØ1   NB(I)=Ø
C
 3Ø1Ø   CALL IMVC(5Ø,BA,NBA,A,L+1Ø)
        NBA=NBA+5Ø
        LL=7Ø
        Z=Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.EQ.Ø) GO TO 2ØØ
        CALL IMVC(3Ø,BA,NBA,A,L+LL)
        NBA=NBA+3Ø
        LL=LL+3Ø
        IF(Z.EQ.1) GO TO 2ØØ
        DO 31ØØ I=2,Z
        CALL IMVC(5Ø,BA,NBA,A,L+1Ø)
        NBA=NBA+5Ø
        CALL IMVC(3Ø,BA,NBA,A,L+LL)
        NBA=NBA+3Ø
 31ØØ   LL=LL+3Ø
        GO TO 2ØØ
C                                                   ADMIN[C(1Ø)]
 4ØØØ   IF (SW3.GT.1) GO TO 4Ø1Ø
        CALL $EMPTY(B(1),INC(9),NBA)
        CALL $SET(C(9),INC(9),1)
        NBA=Ø
        PRINT 9ØØ4
 9ØØ4   FORMAT(' 3ØØØ')
 4Ø1Ø   CALL IMVC(5Ø,BA,NBA,A,L+1Ø)
        LL=7Ø
        NBA=NBA+5Ø
        Z=Ø
        ZZ(4)=A(L+LL)
        Z=Z-24Ø
        IF(Z.NE.Ø) GO TO 4Ø11
        BA(NBA+1)=A(L+LL-1)
        CALL IMVC(19,BA,NBA+1,BA,NBA)
```

```
 4Ø11    CALL IMVC(2Ø,BA,NBA,A,L+LL)
         LL=LL+2Ø
 4Ø12    NBA=NBA+2Ø
         LL=LL+1Ø
         Z=Ø
         ZZ(4)=A(L+LL)
         Z=Z-24Ø
         IF(Z.EQ.Ø) GO TO 2ØØ
C                                                NSTATES-11
         DO 42ØØ I=1,Z
         CALL IMVC(4,BB,NBB,A,L+1Ø)
         NBB=NBB+4
         CALL IMVC(2Ø,BB,NBB,A,L+LL)
         NBB=NBB+2Ø
 42ØØ    LL=LL+2Ø
         GO TO 2ØØ
C
 5ØØØ    IF(SW4.GT.1) GO TO 5Ø1Ø
         DO 5ØØ1 I=1,2
         CALL $EMPTY(B(I),INC(I+9),NB(I))
         CALL $SET(C(I+9),INC(I+9),1)
 5ØØ1    NB(I)=Ø
         PRINT 9ØØ5
 9ØØ5    FORMAT(' 4ØØØ')
 5Ø1Ø     LL=3Ø
         Z=Ø
         ZZ(4)=A(L+LL)
         Z=Z-24Ø
         IF(Z.EQ.Ø) GO TO 52ØØ
C                                                SENATE-12
         DO 51ØØ I=1,Z
         CALL IMVC(4,BA,NBA,A,L+1Ø)
         NBA=NBA+4
         CALL IMVC(2Ø,BA,NBA,A,L+LL)
         LL=LL+2Ø
 51ØØ    NBA=NBA+2Ø
 52ØØ    LL=LL+1Ø
         Z=Ø
         ZZ(4)=A(L+LL)
         Z=Z-24Ø
         IF(Z.EQ.Ø) GO TO 2ØØ
C                                                HOUSE-13
         DO 53ØØ I=1,Z
         CALL IMVC(4,BB,NBB,A,L+1Ø)
         NBB=NBB+4
         CALL IMVC(2Ø,BB,NBB,A,L+LL)
         LL=LL+2Ø
 53ØØ    NBB=NBB+2Ø
         GO TO 2ØØ
C
 6ØØØ    DO 6ØØ1 I=1,2
         CALL $EMPTY(B(I),INC(I+11),NB(I))
         CALL $SET(C(I+11),INC(I+11),1)
 6ØØ1    NB(I)=Ø
         PRINT 9ØØ6
 9ØØ6    FORMAT('PUTD')
         CALL PUTD(C(3),'ST.PRES')
         CALL PUTD(C(4),'ST.OCC')
         CALL PUTD(C(5),'ST.SPOUC')
         CALL PUTD(C(6),'ST.EYEAR')
         CALL PUTD(C(7),'ST.PADM')
         CALL PUTD(C(8),'ST.CONG')
         CALL PUTD(C(9),'ST.ELEC')
         CALL PUTD(C(1Ø),'ST.ADMIN')
         CALL PUTD(C(11),'ST.NSTATE')
         CALL PUTD(C(12),'ST.SENATE')
         CALL PUTD(C(13),'ST.HOUSE')
         DO 61ØØ I=1,13
 61ØØ    CALL FREE(C(I))
         RETURN
         END
```

REFERENCES

1. D. L. Childs, "Feasibility of a Set-Theoretic Data Structure: A General Structure Based on a Reconstituted Definition of Relation," *Proceedings of IFIP Congress, 1968* (North-Holland Publishing Co., Amsterdam, 1969), pp. 420-430.

2. D. L. Childs, "Description of a Set Theoretic Data Structure," *AFIPS Conference Proceedings*, Vol. 33 Part 1, (AFIPS Press, Montvale, NJ, 1968), pp. 557-564.

3. D. L. Childs, "Extended Set Theory: A Formalism for the Design, Implementation, and Operation of Information Systems," *Current Trends on Programming Methodology*, Vol. 4, R. T. Yeh, Ed. (Prentice-Hall, Inc., Englewood Cliffs, NJ — to be published).

4. W. T. Hardgrave, "A Technique for Implementing a Set Processor," *Assoc. Comput. Mach. FDT* 8, 86 (1976); (also published in *ACM SIGPLAN Notices*, Vol. II (1976).

5. W. T. Hardgrave, *Set Processing: A Tool for Data Management*, Department of Information Systems Management, University of Maryland, Information Systems Management Technical Report #6 (1976).

6. W. T. Hardgrave, *Accessing Technical Data Bases Using STDS: A Collection of Scenarios*, Institute for Computer Applications in Science and Engineering, Hampton, VA, ICASE Report 75-8 (1975).

7. W. T. Hardgrave "Set Processing in a Network Environment," Institute for Computer Applications in Science and Engineering, Hampton, VA, ICASE Report 75-7 (1975).

8. E. H. Sibley, Ed., "Computing Surveys," *J. Assoc. Comput. Mach.* 8, 1-151 (1976).

9. *STDS Reference Manual*, Set Theoretic Information Systems Corp., Ann Arbor, MI (1975).

10. *STDS-OS Reference Manual Version 3.02*, Set Theoretic Information Systems Corp., Ann Arbor, MI.

11. D. E. Bakkom, E. W. Birss, and S. Woodison, *IMS Logical Data Bases: An Illustrative Example*, University of Michigan, Ann Arbor, MI, Data Translation Technical Report 76DB2 (1976).

12. D. G. Severence, and G. M. Lohman, "Differential Files: Their Application to the Maintenance of Large Data Bases," *ACM Trans. Database Sys.* 1, 256-267 (1976).

WTF/gw

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $3.00

Page Range	Domestic Price	Page Range	Domestic Price
001–025	$ 3.50	326–350	10.00
026–050	4.00	351–375	10.50
051–075	4.50	376–400	10.75
076–100	5.00	401–425	11.00
101–125	5.50	426–450	11.75
126–150	6.00	451–475	12.00
151–175	6.75	476–500	12.50
176–200	7.50	501–525	12.75
201–225	7.75	526–550	13.00
226–250	8.00	551–575	13.50
251–275	9.00	576–600	13.75
276–300	9.25	601–up	*
301–325	9.75		

*Add $2.50 for each additional 100 page increment from 601 to 1,000 pages; add $4.50 for each additional 100 page increment over 1,000 pages.