PROJECT 2

A COMPUTERIZED SLIDE PROJECTOR SYSTEM

RUSSELL B. CROFT, CDP, BA

A PROJECT
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

Presented in Partial Fulfillment of the Requirements for
the Degree of Master of Science (Applied)
at McGill University
Montreal, Canada

April, 1975

PURPOSE

The original formulation of this problem, like most concepts, was born in the casual conversation of some officers at CNR. In the course of the conversation they pointed out that many times presentations are made about the state of the corporation. The modern technique is to support the presentation with "flip charts" to emphasize various points. These charts contain:

1) plain text (high-lights of the presentation),
2) tables of corporate data shown in financial trends,
3) histograms or plots--an alternate form for presenting tables of data,
4) flow, processing, or work study charts,
5) occassionally cartoons or line drawings.

Recently there has been a trend away from using these flip charts because they are not really suited for large audience presentations or audiences spread out in a large room. The traditional flip chart is being replaced by overhead projectors and particularly projectors using 35mm photographic slides.

The problem with slide presentations is that they do take time and money to prepare, and time is usually at a premium as the presentation is being prepared. However, when the slides are made they form a pseudo-data base for later presentations. Unfortunately some of these slides are obsolete

as soon as they are made. These are the slides dealing with topical data in the form of histograms, plots or tables of current data. When these slides are presented regularly, (monthly or quarterly), managers would like to find some way to update a slide with a new point on a graph or a new column in a table with little cost. Thus, the natural question arose: "Can a computer help?"

When viewed in the context of the computer, the problem of adding information and reformatting a plot or table is really no problem. The real question is: is it worth the bother? or is it worth while trying to formulate the problem as a computer information retrieval problem? Would there be sufficient demand for the information in this form? If the problem was formulated for a computer would there be enhancements available which would not normally be available if a computer was not used?

The justification for using the computer for this project lies in the answer to the last question, i.e. if the problem is formulated with a computer in mind, then there are certain facilities available which are not normally available if a computer was not used. These include fast plotting and re-plotting of ammended data and access to computer financial data bases like CANSIM[1], or various stock markets. And, of course, one must not overlook the novelty of using the computer for this kind of work.

[1] CANSIM is a very comprehensive socio-economic data base maintained by Statistics Canada containing historical time series on factors like population, earnings, productivity, industrial and corporate price indices, statistics related to GNP and so on.

Thus I received a problem formulated as follows:

1) devise a method to store and retrieve corporate information suitable for conference presentations
2) the retrieved information to be displayed via overhead projectors and must fit into some predetermined "page size".
3) design the retrieval to be as much like a standard photographic slide presentation as possible
4) allow for updating and reformatting of information so that current data can be retrieved when needed
5) all retrieval to be made with 30 seconds (say) or during the time that a presenter is dwelling on the topic of a previous slide whichever comes first
6) allow access to financial data bases and in particular to CANSIM

ANALYSIS OF THE PROBLEM - INTRODUCTION

For most projects, a computer analyst's job is well defined, so that he can immediately start to think in terms of his programming method. Such was not the case with project. The first task was to envision just how we could get a computer to act like a slide projector, and to do it in such a way that it could be operated by (perhaps) inexperienced people. In addition I felt that the whole procedure should be designed as much like the manual way as possible so that users could relate to it easily.

For the present manual method, after the person making the presentation has designed his speech, he sketches out some slides to support the talk. Remember that at CN most of these presentations deal with corporate affairs and some of the information must come from CANSIM in some form. The presenter first checks to see if he has any slides in his files that he can use. The remainder is typed and photographed. The completed set of slides is then collated and placed in a projector ready for the presentation. During the presentation every time the presenter presses a button on the projector, a new slide appears and he continues talking. He does have the flexibility to go back to previous slides to re-emphasize what was said, or he can skip ahead a few slides to get a "sneak preview" of other slides. But if the presenter forgets a slide he cannot just stop his presentation and return to his office to get it. Or if someone deviates from the topic slightly and asks "But what about such-and-such? Have you got a slide depicting that?" The presenter must politely say no.

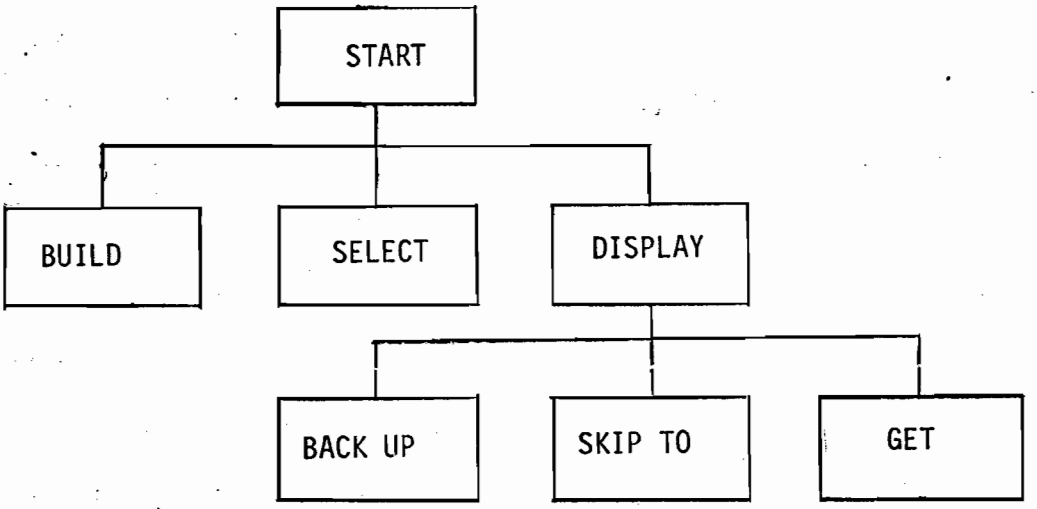


Fig. 1

Process Flow for
Manual Projection

I felt that the above scenario could be simulated exactly even to the point where CANSIM is accessed. There would be some differences and certainly one big exception: my projector, being a computer, would never have run around looking for slides. All these slides would be close at hand. I could envision the ultimate system designed to perform these three basic functions: (See Fig. 1 opp.)

1) Build and store slides for later retrieval
2) Retrieve slides and arrange them for a projection type of display
3) Display slides on a device during a presentation

During the display we will want not only to be able to show the selected slides, but also to:

a) backup or review some slides
b) skip ahead or preview slides
c) (an option not available in the manual method) insert slides while talking.

All these options should be made to work not only on our own bank of stored slides, but also be able (according to my terms of reference) to access CANSIM, some other financial data banks and some special plot functions. One might even envision being able to "play around" with a plot on the screen to answer such questions as, "but what would that plot look like if we did...". This last idea is not impossible, but let's not get into that just yet.

Knowing what we wanted to do just amplified our problems. Of course, since we must be able to access CANSIM and since some man-computer interaction must take place, to be able to accomplish all this, a computer time-sharing service, either commercial or private, must be made available. So from this starting point let us address ourselves to:

PROBLEM 1 - I/O Design

If this project was to be successful at all, some suitable video display device is needed. This device must be able to display enough characters at once to be meaningful and at the same time be large enough to be seen by larger audiences. There are many suitable video devices available with attachments for large audience displays, but most of these have a limited "page size". By this I mean the number of characters that can be displayed at one time. For example the page size of this paper is 64 x 120 characters (8" x 10" paper with a resolution of 8 lines per inch vertically and 12 characters per inch horizontally). Below is a sample of some typical units with their page sizes:

1)	VuCom 1 unit	16 x 80 characters
2)	IPSA 100 unit (APL)	16 x 32 characters
3)	Tektronix 4013 unit	35 x 74 characters
4)	Tektronix 4015 unit	65 x 133 characters

The Tektronix unit also has two extra features:

a) full graphics capability to draw graphs or figures

b) a transmission line adaptor to link the unit to a closed circuit television system

At this stage it was not necessary to choose a particular unit, but it was necessary to know what limitations we were going to have. A page

APPLICATION AREAS

- PRESENTATIONS TO GOVERNMENT, PUBLIC BODIES, MANAGEMENT GROUPS
- CAPITAL BUDGETS
- FINANCIAL PLANNING SIMULATION RESULTS
- MARKET FORECASTS
- OPERATING STATISTICS
- INVESTMENTS IN ROAD PROPERTY AND FLEET
- CANSIM (OR STATSCAN) DATA ON LABOR FORCE, PRICE INDICES, DOMESTIC TRADE ETC

FIG. 2-A Plain Text

TRAIN AND YARD ACCIDENTS - SYSTEM SUMMARY

ACCIDENTS:	1972	1973	•/• IMPROVEMENT
COLLISIONS	92	96	-4.3 •/•
DERAILMENTS	642	555	13.6
SWITCHING	1755	1534	12.6
MISCELLANEOUS	182	127	30.2
TOTAL INCIDENTS	2671	2312	13.4 •/•
TOTAL COSTS($M)	13.8	11.8	14.7
PROPERTY DAMAGE EXCEEDING $750	544	248	54.4
REPORTED TO R.T.C.	245	212	13.5

FIG. 2-B Tabular Data

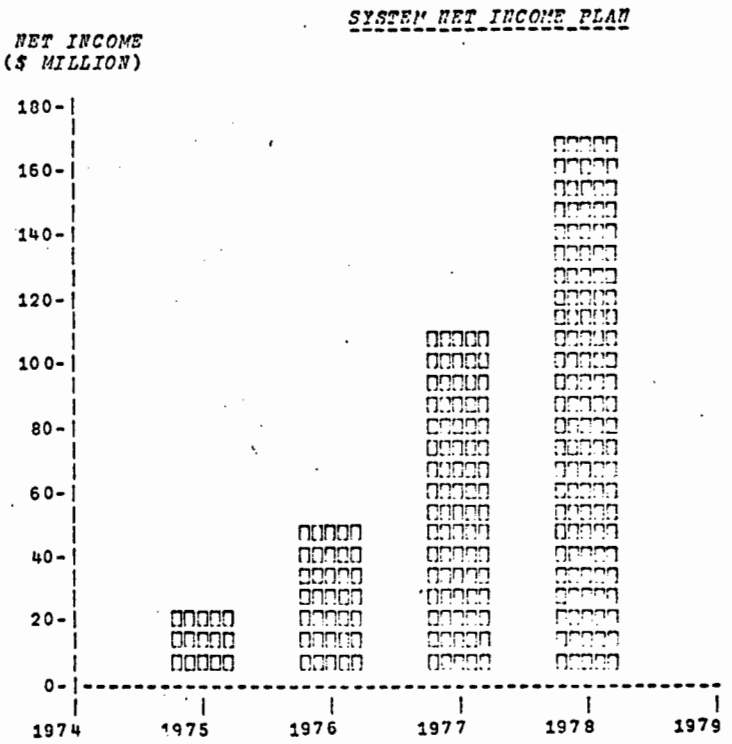


FIG. 2-C A Histogram

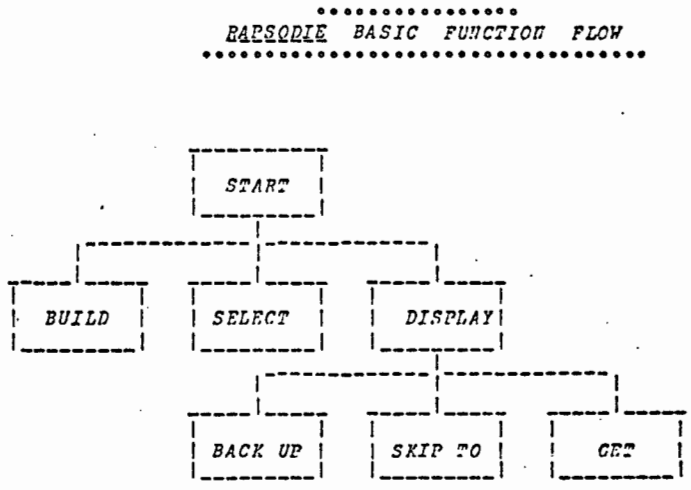


FIG. 2-D Flow Chart

FIG. 2 EXAMPLES OF SLIDES NEEDED

of 35 x 75 characters is certainly adequate so we could now start planning how to display the data with this ultimate restriction in mind.

Opposite is a sample of the kinds of display currently needed for this project to be successful. These are: (See Fig 2 opposite)

A) formatted text of key phrases - summaries or highlights of the presentation

B) tabular data - this data can be supplied by the user or retrieved from CANSIM (say)

C) histograms or plots - an alternate way to present tabular data. These can appear as character plots (shown) or line plots (by using a graphics capability).

D) flow charts - which can be generated as character displays or as graphics displays if the facility is available (not shown)

(not shown) maps, cartoons, or other diagrams - probably generated by graphics facilities only.

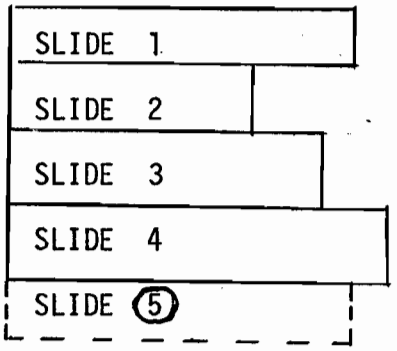
For the present let us limit this discussion to character displays only. Graphics displays can be formulated later without much extra effort, once the technique is learned. I must assume the reader is familiar with computer processing in some way. Consequently, it should be easy to formulate a method to enter and store this kind of character data. One way to do this would be to type one line at a time and store these lines as sequential fixed length

records on some storage medium. It is advisable to store these records using a blokcing factor equal to the number of lines in the display, so that each block contains one complete "slide". For example, for the Tektronix screen of 35 x 75 characters, try to store 35 records (lines) of 75 characters each in one block. To save storage, another way might be to store the data as a fixed number of variable length records. Each record (line) would have the trailing spaces deleted, but the number of records per block would still be the depth of the display (35 lines for the Tektronix unit). Each programmer in his own way can see how he might physically store the data. His method will depend on the computer processing language and access method used. My method will be presented later, but before we get to that topic, we should discuss.

PROBLEM 2 - Storage and Retrieval Method

At the beginning of the project it was implied that no matter what we stored and retrieved, it has to be done randomly. That is, we will want to store data as it comes available and we will want to retrieve the formatted slides generally speaking not in the same order in which it was stored. For random retrieval and display, the only practical method is to use some sort of random access disk organization. Magnetic tapes are really not appropriate because they are sequential and take much time to scan. For instance, if information we wanted was near the end of the tape, it would take 3 - 5 minutes to get to it assuming we had available modern tape drives. On

Slides



Vocabulary

Words	Pointers
Corporate	1 2 4
Planning	1 2 4
Objectives	1
Histogram	2 3 5
Train	3 4
Accident	3
Atlantic	3 5
Population	5

Slide 1 deals with CORPORATE PLANNING OJECTIVES

2 is a CORPORATE PLANNING HISTOGRAM

3 is a TRAIN ACCIDENT HISTOGRAM for ATLANTIC provinces

4 deals with TRAIN PLANNING OBJECTIVES

5 is a POPULATION of ATLANTIC provinces HISTORGRAM

Fig. 3

Inverted File Structure

the other end of the scale, data cells or drums are not readily available, so should also be discarded from consideration in general. If they are available, they could be used (most of the time) to greater effect than disks. Disks (drums if available) allow us to access data immediately when we know exactly on which track and cylinder it resides. This last statement leads directly to this problem - how does one organize data for efficient and fast subsequent retrieval? We must keep in mind that we must retrieve a slide in a relatively short space of time.

To store a piece of information for subsequent retrieval we must store it away with a set of keys. For example, to store a slide showing a HISTOGRAM of the growth of the POPULATION of the ATLANTIC provinces, (data supplied by CANSIM), the slide must be stored with a reference of these words in some way. To do this sequential storage techniques must be rejected in favour of random techniques for the same reasons that sequential storage devices are rejected--they are just too slow. Since we are going to use multiple keys to store the slides, some kind of an INVERTED FILE structure should be used.

The simplest way to explain this technique is by way of a small example (see Fig 3 opposite). For this kind of organization we could set up two files on disk, the first containing all the slides, (in this case 4 slides) and the other containing a vocabulary of keywords which reference these slides (in this case 7 words). Each vocabulary word would have a list of

positions in the slide file of all the slides referenced by that word. For example, the list for the first word, CORPORATE, in the vocabulary file of Fig 3 shows that slides 1, 2 and 4 deal with the subject CORPORATE.

When a new slide is to be added to the file, rather than inter-file it amongst the others (the sequential technique), for convenience, it would be nice if we could place it anywhere we like. So, to avoid extra programming, let us place the new slide at the end of the file.

After the slide has been added to the file, the list for each keyword describing the new slide will be changed to include its position. Thus the lists for HISTOGRAM and ATLANTIC will have "5" (the position of the new slide) added to them. Once again, new keywords (POPULATION in this case) will be added to the end of the vocabulary file, with the location of the new slide (5) recorded.

The retrieval procedure would be equally as simple. To retrieve the ATLANTIC POPULATION HISTOGRAM slide, we would take the set intersection of the lists of position numbers recorded with each of these keywords i.e. (2,3,5) (3,5) (5), and the result (5) is the position of the slides desired.

Granted, the keyword list must be searched sequentially, but this list is usually quite small (say around 100 keywords) compared to the number of slides which could number several hundred. To save search operations on disk, the vocabulary itself (just the words) could be kept in core. Only the "slide

pointer" as we will call them, will remain on this file. This will allow us to use a very fast "in core" search technique for these keys.

The method outline above is the method used for the slide projection system. The method can be used not only to store slides but also to retrieve some of them for projection. To bring the system into perspective, the manual and computer methods might be compared this way:

STEP	MANUAL METHOD	COMPUTER METHOD
1	Slides are typed, photographed and stored in a file box.	Slides are typed into a computer. The computer files them in a slide file with reference to a set of keywords.
2	Slides are selected and placed in a carousel ready for projection in a slide projector.	Slides are selected by supplying a set of keywords and are placed sequentially in a temporary display file.
3	The user displays the slides under "push buttom" control. He can move the carousel forwards or backwards.	The user tells the computer to print the slides from the display file, one at a time. The user can tell the computer to go forwards or backwards. In addition he may tell the computer to select other slides from the slide file whenever he wants.

All we must do is find a computer processing language to help do this work quickly and efficiently.

PROBLEM 3 - Processing Language

Let us summarize what we have discussed so far. We have decided that:

1) A computer time-sharing service will be needed because it allows access to CANSIM and also because there will be some man-computer interaction needed.
2) A video display unit is available to connect to a computer to show our results. Such a unit might be the Tektronix 4013 with transmission line adaptor and graphics capability.
3) Our slides can be entered (for the Tektronix unit) as a series of 35 lines (possibly less) of 75 characters each (possibly less) and stored as one block on a disk.
4) Random storage and retrieval must be used because our data will not be either available or needed in any particular order.
5) An inverted file organization might be the best way to store and retrieve the data.

To this list we might append the following requirements for choosing a processing language. The language used should :

6) be time-sharing oriented
7) be easy to write programmes in (since we do not want to take forever developing the system)
8) be easy to use by ultimate users perhaps even by novices
9) be able to handle character strings easily and have efficient input-output operations

10) be easily programmed for conversational interaction between man and his computer

11) (important) be readily available, both the language and the company offering the service

12) finally (most important) be very reliable, i.e. we do not want the computer to fail just when we need it most, since the computer must be "on-line" during operation of the system.

If such a language could be found then the programming could be completed with a minimum of keyboard effort for both the user and the analyst. COBOL and FORTRAN though good for file handling (FORTRAN may not be the best as far as I/O operations are concerned), do not handle and/or manipulate character strings easily. Nor do they lend themselves to conversational interaction with a minimum of programming effort. ALGOL, LISP, SNOBOL PL/1 and other string processing languages answer most of these objections, but are not readily available. In addition some of these languages are "wordy", and still do not lend themselves to conversational interaction. What is really needed is a good interpretive language, but is there such a language that fits all these stringent requirements? Fortunately, yes, APL[1] does all this and more. In fact, some of the techniques listed in problems 1 and 2 can be greatly simplified by using APL with perhaps only a small increase in cost. I think the extra cost will be worth all the trouble that the language itself will save.

[1] For the balance of this study, the APL language assumed will be Sharp APL, supplied by I.P. Sharp Associates, Toronto; herein refered to only as APL or occasionally SHARP APL.

In particular, SHARP APL has available the Tektronix "Software" to handle the video unit in graphics mode. APL file handling facilities are especially good for this application, since APL stores data as variable length sequential file components. However, each component can be retrieved randomly by specifying its relative position in the file (e.g. by actual component number) - a very useful feature. This means that we do not need to think in terms of 35 lines per stored block on disk. Instead, the lines may be "strung out" or concatenated, with imbedded ASCII carriage returns as separators. This will have exactly the same effect as storing and printing several lines deliminated as "records" in a block. And, again, since we are able to address specific components (records) on disk, all we need to do is keep track of where a particular slide is placed (by component number) for later retrieval. Thus our INVERTED FILE organization technique is going to be greatly simplified. As pointed out in the example, the easiest location to file a new slide is at the end of the file. Again APL helps out because at all times there is available the number of components in a file, so this "last" location is known by number for later reference.

Finally, the very nature of APL encourages modular programming. These modules or FUNCTIONS can be coded to give a high degree of interaction between man and his machine.

In some of the following discussions because of this modular approach, flowcharts may not be possible to write. Instead, a function flow may be

a better way to describe the way things are being done. END OF "COMMERCIAL! Now, with all the background completed, the reader is referred to the next section which is, in effect, a users guide to the storage and retrieval functions.

Most systems eventually get an acronym to identify it. So far we have implied that this is a retrieval and projection system. In addition, the project was formulated as an online display of corporate information to our executives. So what better acronym than:

CN RAPSODIE

(Retrieval And Projection System for

Online Display of Information to Executives)

being

OPUS #1 in APL (SHARP) for video display units!

RAPSODIE FUNCTION DESCRIPTION

Introduction

RAPSODIE is an interactive video display system designed to store character strings representing photographic slide images of information for conference presentation. RAPSODIE is written in APL, consequently, many of the facilities of APL have been exploited to make the operation of this system easy to use. The first of these facilities is the modular programming concept. By modular programming we mean a concept where many small functions (sometimes called subroutines in other languages) are written to perform essentially menial tasks, and then linked together to form one large system of programmes. RAPSODIE is written in some thirty interlocking functions, each of which can stand alone or can be used in combination with other functions.

Because of the interaction of the functions and the way they can be called from each other, it is advantageous to design function names and parameters to approximate the spoken word. This feature gives the appearance of talking to the computer, having the computer understand, and executing personal instructions.

RAPSODIE logically divides itself into four sets of functions:

1) The BUILDER functions - those functions that permit us to enter and store slides for later retrieval

2) The SELECT function - those functions that will permit us to retrieve slides and temporarily store them for later display.

3) The DISPLAY function - those functions that will allow us to display the selected slides in a method not at all unlike a photographic slide projector.

4) Some "housekeeping" functions - functions to clean up garbage floating around, for deleting unwanted slides, and for selective display of slides during programme development.

These four sets of functions will be used to store and retrieve data in an inverted file structure. The inverted file is also modularized into three files. The structure of these files, their uses, and rules to form them are as follows: See Fig A opposite.

File S: <u>The slide file</u>

- This file contains all the slides.
- New slides are added (appended) at the end of this file.
- The file is organized on disk.

File V: <u>The vocabulary file</u>

- This file contains all the keywords needed to describe a slide.
- New keywords are appended to the end of the file.
- This file is contained entirely in "core" as an N x 15 character array. N will be in the order of 100 words
- The position of each keyword is the component number of the slide pointers that this keyword references.

File P: The slide pointers

- This file contains one component for each keyword in file V.
- These components are in exactly the same order as File V.
- Each component contains pointers to each slide referenced by this keyword.
- The pointers are the component numbers of the stored slides.
- This file is organized on disk.

RAPSODIE is simple in concept and design. To emphasize this each section which follows contains both an operating guide and a technical guide rather than separating these guides into separate secitons. For each group of functions, four descriptions are supplied as follows.

a) a general description of the main function of the group. In conjunction with the sample terminal session this description is sufficient to operate RAPSODIE.

b) a sample terminal session with comments

c) a general description of each component function in the group arranged in Alphabetical order. This description is supported by both a logic flow chart and a function flow chart.

d) a detailed listing of the component functions of the group in order of appearance in the function calling tree.

The user is referred to and assumed to have some basic familiarity with the following publications:

a) APL PLUS file Subsystem Instruction Manual, I.P. Sharp Associates Ltd.

b) APL PLUS Plot Facility, I.P. Sharp Associates

c) APL An Interactive Approach, 2nd Edition, Gillman and Rose, Wiley, 1974

LOADING RAPSODIE

RAPSODIE uses the file subsystem, consequently potential users will need authorization to use these files before loading the system. Assuming authorization has been obtained, typing

```
)LOAD 1977269 PRES
```

will not only load the workspace containing all the necessary functions, but the workspace will automatically initialize itself by executing a function called START. START performs several necessary functions:

1) Tabs are set at 10 space intervals. There is nothing magic about the number 10, so tabs can be set to any desired value.
2) The three storage files are "opened", i.e. made accessible for use.
3) Three variables which RAPSODIE uses constantly are initialized. These are NIDS - the number of keywords or id's currently available
   IDS - the matrix of the keywords (id's) themselves
   IDLEN - the current length of the keywords.

THE BUILDER FUNCTIONS

1. a) General Description - "Character String" Slides

By typing BUILD the builder group of functions is accessed. These functions accept character data line by line until a line ending with at least 5 spaces is typed. This can be done by entering "TAB, CR". As an option, the user is asked to supply headings above and below the slide. "TAB, CR" will terminate this input. The resulting set of characters, called a slide is now ready for filing. The user is asked to enter a set of keywords describing the slide entered. (Note to experienced users: portions of keywords may be entered. RAPSODIE is quite capable of finding the complete word by itself.) The keywords are retyped for verification. If the set of keywords entered defines no other slide, then the slide is filed for later reference under the keywords specified. If the keywords specify exactly 1 slide already filed, then the entered slide, assumed to be a replacement for the old one, replaces the old slide on file. If the set of keywords specifies more than one slide already on file, a new, more well-defined set of keywords is requested. After this slide is filed, the user is asked if he wishes to continue. A NO response will terminate the BUILDER.

1 B) SAMPLE TERMINAL SESSION - SLIDE BUILDING

The computer prints '→' before its dialogue The BUILDer is called	*BUILD* → *ENTER SLIDE*
enter the slide	*THIS IS A SLIDE* *IT HAS 2 LINES*
TAB, CR or 5 spaces end input	→ *ENTER TOP HEADINGS:*
	→ *ENTER BOTTOM HEADINGS:*
Keys are entered	→ *INPUT IDENTIFIERS:SLIDE 1.* → *NO SLIDES FOR: SLIDE 1.* → *KEYS SELECTED. OK?*
Keys are retyped for verification	→*SLIDE* →1.
The keys specify no other slide, thus the slide is added	*YES* → *COMP 52 ADDED UNDER: SLIDE 1.*
Let us try another	→ *MORE?* *YES PLEASE* → *ENTER SLIDE*
enter the slide	∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘ | | | *BOX* | ∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘
Headings (title) entered	→ *ENTER TOP HEADINGS:* *DEMO BOX*
5 spaces or TAB, CR	→ *ENTER BOTTOM HEADINGS:*
The keys are not unique	→ *INPUT IDENTIFIERS:CORPORATE* → 5 *SLIDES FOR: CORPORATE* → *KEYS SELECTED. OK?*
We reject the keys and try again	→*CORPORATE* *NO* → *INPUT IDENTIFIERS:SLIDE 2.* → *NO SLIDES FOR: SLIDE 2.* → *KEYS SELECTED. OK?*
This time no other slide is defined	→*SLIDE* →2.
The side is added	*YES* → *COMP 40 ADDED UNDER: SLIDE 2.*
That is all for now	→ *MORE?* *NO THANKS*

2. a) General Description - Plot and Histogram slide BUILDing

To store a plot or histogram the following procedure is used.

1) type: NOPRINT. This supresses the actual printing of the plot before storing the slide.

2) assign to a variable the result of PLOT, e.g.

"plotvariable" ←PLOT "plotdata"

3) type: FILE HEADINGS FOR "plotdata" this instruction behaves just like BUILD except that a slide is not entered. The user will be asked to supply HEADINGS or titles on top of and below the plot in "plotvariable". a "TAB, CR" ends each input as described in BUILD. The plot slide will now be filed using keywords supplied by the user, again, as described in BUILD.

It is assumed that the user knows how to use the APL PLOT Facility. For the present, this is the easiest way to store a plot or histogram slide. Future enhancements will do this more automatically.

2 B) SAMPLE TERMINAL SESSION - PLOT BUILDING

The computer prints → before its dialogue

First we set up the plotter

```
SET ABSCISSA TOTAL 4 SPACED 6 NUMBERED 1
SET ORDINATE TOTAL 3 SPACED 5 NUMBERED 1

COLUMN 1
```

we create some data

and PLOT it

```
X←.5×1+⍳7
IT←⍉3 7⍴X,X,5-X
PLOT IT
```

```
5.00 +
     |
     *                    o
     |
     |   *              o
3.00 +       *      o
     |           *
     |       o      *
     |
     |   o              *
1.00 o---'---+---'---+---'---*
        1.0    2.0    3.0    4.0
```

Just what the doctor ordered let us file it

We turn the print off PLOT the data and save it for later
Now we file the plot

```
NOPRINT
SOMEVAR←PLOT IT

FILE HEADINGS FOR SOMEVAR
```

We enter headings (titles) on top
TAB, CR or 5 spaces end the input

```
→ ENTER TOP HEADINGS:
                  DEMO PLOT
```

Titles at the bottom

```
→ ENTER BOTTOM HEADINGS:

          o  X=Y      *  X=5-Y (5 spaces here)
```

The keys specify no other slide

```
→ INPUT IDENTIFIERS:DEMO SLIDE
→  NO SLIDES FOR: DEMO SLIDE
→ KEYS SELECTED. OK?

→DEMONSTRATION
→SLIDE
NO
→ INPUT IDENTIFIERS:SLIDE
→   2 SLIDES FOR: SLIDE
→ KEYS SELECTED. OK?
```

Verification
But we can change our mind

The new set is not unique

```
→SLIDE
NO
→ INPUT IDENTIFIERS:SLIDE 3.
→  NO SLIDES FOR: SLIDE 3.
→ KEYS SELECTED. OK?
```

We try again

The new set of keys is OK

```
→SLIDE
→3.


YES
→ COMP 35 ADDED UNDER: SLIDE 3.
```

The slide of the

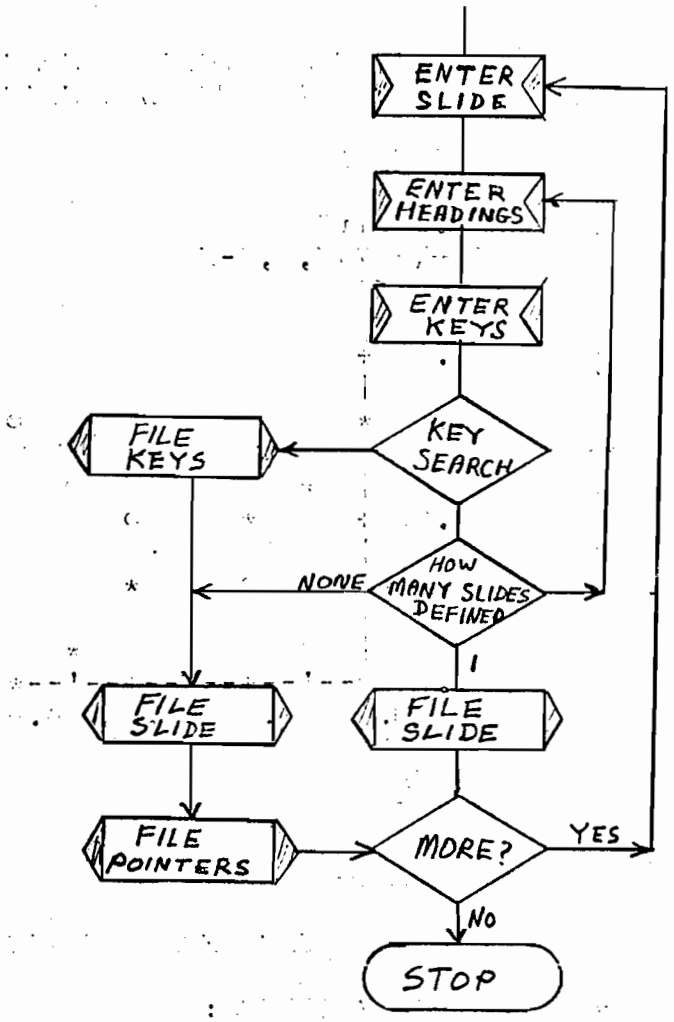


Fig. B

Builder Flow Chart

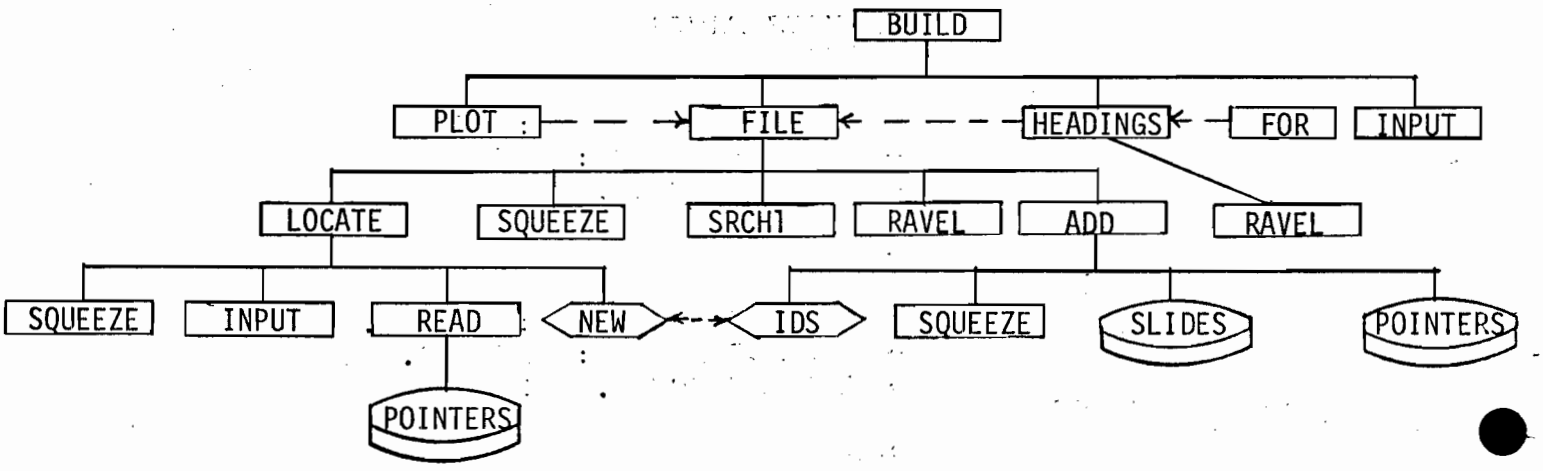


Fig. C

Builder Calling Tree

c) Component Descriptions - BUILDer group

Below is a brief description of the BUILDer group of functions.
Fig. B (opposite) is a flow chart of the BUILDer logic
Fig. C (opposite) shows the BUILDer function calling tree

BUILD	allows entry of slides and stores them by keywords for later retrieval. The following functions are used by BUILD
ADD string	adds 'string' to the slide file and records its location for later retrieval
FILE string	initiates filing of 'string' under keyword control
FOR x	a "do nothing" programme used for readability, the output from FOR is the same as the input (x)
HEADINGS s	allows user to supply headings above and below string 's'
INPUT text	prints 'text' as a message to user and allows a response on the same line
LOCATE	locates all slides in the slide file described by a set of keywords supplied by the user
RAVEL array	converts 'array' to a string with imbedded ASCII carriage returns.
c READ f	reads the component 'c' from the file tied to 'f' (refer to FILE SUBSYSTEM manual if this terminology is meaningless)
SQUEEZE k	removes extra spaces between keywords in the set 'k'
a SRCH1 b	a specialized string searcher which finds all occurances of string 'a' in string 'b'

D) DETAILED FUNCTION LISTING OF BUILDER GROUP

IN ORDER OF APPEARANCE IN CALLING TREE

```
    ∇ BUILD;INP;A;NEW;KI
[1]  MOR:'→ ENTER SLIDE',CR ⋇ INP←CR,CR
[2]  INN:→(~∧/(' ')=¯1↓¯6↑INP←INP,⍞,CR)/INN ⋇ FILE HEADINGS FOR
      INP ⋇ →('N'≠1↑INPUT 20↑'→ MORE?')/MOR
    ∇

    ∇ FILE X;A;KI;NEW
[1]  ENT:A←LOCATE ⋇ '→ KEYS SELECTED. OK?',CR,CR,(RAVEL '→',IDS[
      KI;],[1] NEW),CR
[2]   →(('QN'=1↑⍞),(1 0 =ρA),1)/0,ENT,GO,AD,ENT
[3]  GO:((SQUEEZE IDS[KI;]),CR,'¨',((X≠'¨')/X←(1↑(,'¨') SRCH1 X)
      ↓X←RAVEL X),'¨') FE 8 10 ,A
[4]   '→ COMP ',(,'I3' ΔFMT A),' RE-FILED UNDER: ',(SQUEEZE IDS[
      KI;]),CR,CR ⋇ →0
[5]  AD:ADD CR,'¨',CR,((X≠'¨')/X←(1↑(,'¨') SRCH1 X)↓X←RAVEL X),C
      R,'¨'
    ∇

    ∇ R←LOCATE;T;S;I;X;Y;Z;K
[1]   R←⍳NSLIDES ⋇ I←⍳NIDS ⋇ S←K←(SQUEEZE INPUT 20↑'→ INPUT IDEN
      TIFIERS:'),' ' ⋇ KI←X←⍳0 ⋇ NEW←(0,IDLEN)ρ' '
[2]  F:→((0=ρX),(Z←∧/' '=S←T↓S),¯1≠ρR←(R∈(1+X←(((NIDS,ρY)↑IDS)∧.
      =Y←(¯1+T←S⍳' ')↑S)/I) READ KEYS)/R)/3↑A,D,F,KI←KI,X
[3]  A:→(Z,1)/2↑D,F,ρNEW←NEW,[1]((0=ρX),IDLEN)ρIDLEN↑Y
[4]  D:'→ ',(((3ρ~T),T←3ρ(ρR)≠0)/' NO',,'BI3' ΔFMTρR),' SLIDE',(
      (1≠ρR)/'S'),' FOR: ',K,(ρKI)↓KI←KI,X
    ∇

    ∇ R←SQUEEZE X;S
[1]   R←1↓(S∨1⌽S←X≠' ')/X←' ',,X
    ∇

    ∇ R←INPUT T
[1]   R←(ρT)↓(-ρT)↓⍞,⍞←T
    ∇

    ∇ R←COMP READ FIL;I
[1]   →((ρ,COMP)=ρR←⍳I←0)/0
[2]   R←R,FE 6,,FIL,(,COMP)[I←I+1]
[3]   →(I<ρ,COMP)/2
    ∇

    ∇ R←A SRCH1 B
[1]   R←((B,(ρA)ρ' ')[R∘.+¯1+⍳ρA]∧.=A)/R←(B=1↑A)/⍳ρB
    ∇

    ∇ R←RAVEL X
[1]   R←,X,CR
    ∇
```

```
     ∇ ADD S;N;M;I;V;VV
[1]    (IDS←IDS,[I←1] NEW) FE 3↑ 8 30 1 ,V←KI,NIDS←⍳1↑⍴NEW
[2]    ((SQUEEZE IDS[V;]),S) FE(7+M),10,(M←N≠M)/N←1↑(VV←FE 6 10 1
       ),M←1↑1↓FE 10 10
[3]    ((~VV∈N)/VV) FE 8 10 1
[4]   A1:(N,M/FE(6+4×~M),30,M/V[I]+1) FE(7+M),30,(M←V[I]∈KI)/V[I]
       +1
[5]    →((I←I+1)≤⍴,V)/A1
[6]    '→ COMP ';N;' ADDED UNDER: ',(SQUEEZE IDS[V;]),CR,CR,1↓NID
       S←1↑⍴IDS
     ∇

     ∇ R←HEADINGS X;I1;I2
[1]    ⍞←CR,'→ ENTER TOP HEADINGS:',I1←I2←CR
[2]   IN1:→(~∧/(' ')=¯1↓¯6↑I1←I1,⍞,CR)/IN1
[3]    CR,'→ ENTER BOTTOM HEADINGS:',CR
[4]   IN2:→(~∧/(' ')=¯1↓¯6↑I2←I2,⍞,CR)/IN2
[5]    R←I1,(RAVEL X),I2
     ∇

     ∇ R←FOR X
[1]    R←X
     ∇
```

THE SELECTOR FUNCTIONS

a) General Description

By typing SELECT KEYS the selector group of functions is accessed. These functions allow selection of stored slides under keyword control. The selected slides are placed on a temporary DISPLAY file for presentation at a later date. The user is asked for a set of keywords. RAPSODIE selects all the slides applicable and asks the user if all of them are needed. If some of the slides are not needed the user is asked to define a more well-defined set of keywords. This process is repeated until the user has SELECTed all the slides necessary. If the user wants to continue SELECTing slides he answers YES when asked if he wants to continue. To assist the user a catalogue of slide titles is available.

B) *SAMPLE TERMINAL SESSION - SELECT*

	SELECT KEYS
A display file is made ready We try to find the first slide	→ *OLD DISPLAY DESTROYED* → *INPUT IDENTIFIERS:SLIDE 1*
Found it. Let us keep for the display	→ *1 SLIDE FOR: SLIDE 1* → *WANT THEM ALL? YES* → *MORE? YES PLEASE*
Now let us get all 3 of the test slides including slide 1.	→ *INPUT IDENTIFIERS:SLIDE* → *3 SLIDES FOR: SLIDE*
They have been found	→ *WANT THEM ALL? YES*
We place them on the file we now have slides 1,3,2,1 We do not have to use the entire keyword	→ *MORE? YES* → *INPUT IDENTIFIERS:DEMO SLI* → *NO SLIDES FOR: DEMO SLI*
Let us quit here and se what we have	→ *MORE? NO*

The computer prints '→' before its dialogue

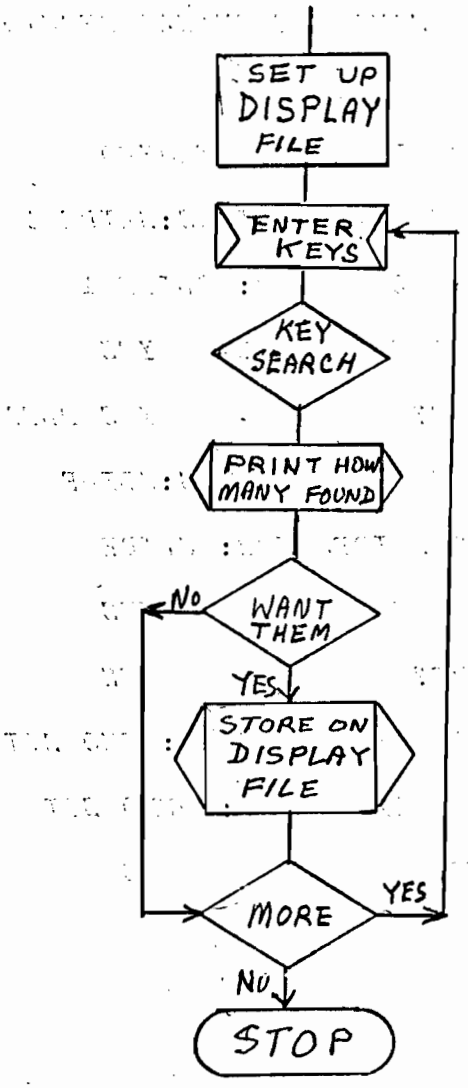


Fig. D
Flow Chart SELECTor

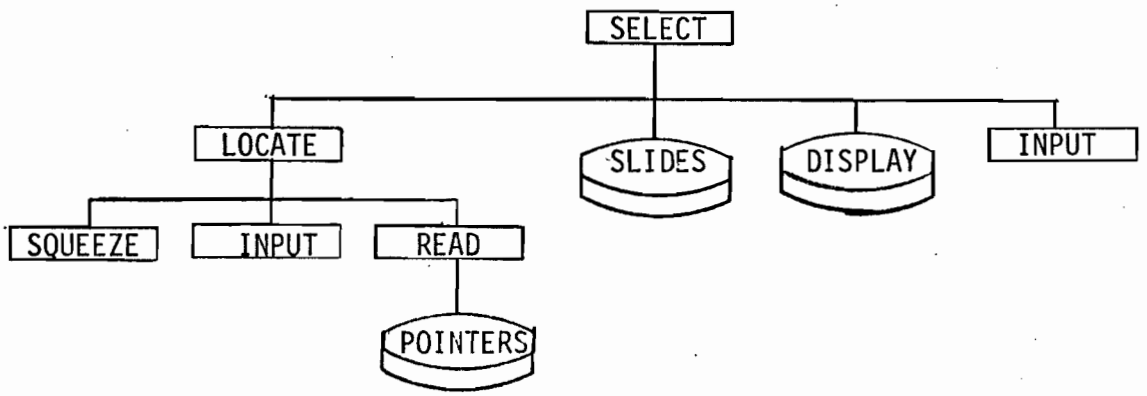


FIG. E The SELECTor Calling Tree

c) Component Function Descriptions - SELECTor group

Fig D (opposite) is a flow chart of the SELECTor logic.

Fig E (opposite) shows the SELECTor function calling tree.

Below is a brief description of the SELECTor group of functions

SELECT KEYS	selects slides (by keywords supplied by the user) from the slide file and stores them on a dispaly file ready for presentation at a later date. The following functions are used by SELECT
INPUT text	prints 'text' as a message to the user and allows a response on the same line
LOCATE	locates all slides in the slide file described by a set of keywords supplied by the user
c READ f	reads the component 'c' from the file tied to 'f'. (The user is referred to the FILE SUBSYSTEM manual if this terminology is meaningless.)
SQUEEZE k	removes extra spaces between keywords in the set 'k'

D) DETAILED FUNCTION LISTING OF SELECTOR GROUP

IN ORDER OF APPEARANCE IN CALLING TREE

```
     ∇ SELECT X;A;I;NEW;KI;V;Y
[1]    FE 13 20 ,-¯1+1↑1↓FE 2↑ 10 20 ,ρ⍞←'→ OLD DISPLAY DESTROYED
       ',CR
[2]    →(((I←1)<ρρX),1)/2↑GO,LP,ρA←,X
[3]   GO:→(0=ρA←LOCATE)/LP+2
[4]    →('QN'=Y←(I←1)↑INPUT 20↑'→ WANT THEM ALL?')/(LP+2),GO
[5]   LP:((¯2+V⍳'¨')↓V←FE 6 10 ,A[I]) FE 7 20
[6]    →((ρ,A)≥I←I+1)/LP
[7]    →('N'≠1↑INPUT 20↑'→ MORE?')/GO
[8]    ('¨',CR,CR,(15ρ' '),'END OF DISPLAY',CR,'¨',CR) FE 7 20
     ∇

     ∇ R←LOCATE;T;S;I;X;Y;Z;K
[1]    R←⍳NSLIDES ⌹ I←⍳NIDS ⌹ S←K←(SQUEEZE INPUT 20↑'→ INPUT IDEN
       TIFIERS:'),' ' ⌹ KI←X←⍳0 ⌹ NEW←(0,IDLEN)ρ' '
[2]   F:→((0=ρX),(Z←∧/' '=S←T↓S),¯1≠ρR←(R∈(1+X←(((NIDS,ρY)↑IDS)∧.
       =Y←(¯1+T←S⍳' ')↑S)/I) READ KEYS)/R)/3↑A,D,F,KI←KI,X
[3]   A:→(Z,1)/2↑D,F,ρNEW←NEW,[1]((0=ρX),IDLEN)ρIDLEN↑Y
[4]   D:'→ ',(((3ρ~T),T←3ρ(ρR)≠0)/' NO',,'BI3' ΔFMTρR),' SLIDE',(
       (1≠ρR)/'S'),' FOR: ',K,(ρKI)↓KI←KI,X
     ∇

     ∇ R←SQUEEZE X;S
[1]    R←1↓(S∨1⌽S←X≠' ')/X←' ',,X
     ∇

     ∇ R←INPUT T
[1]    R←(ρT)↓(-ρT)↓⍞,⍞←T
     ∇

     ∇ R←COMP READ FIL;I
[1]    →((ρ,COMP)=ρR←⍳I←0)/0
[2]    R←R,FE 6,,FIL,(,COMP)[I←I+1]
[3]    →(I<ρ,COMP)/2
     ∇
```

THE DISPLAY FUNCTIONS

a) General Description

By typing DISPLAY, the display group of functions is accessed. These functions start the prepared overhead presentation on the desired video display device. RAPSODIE immediately pauses--the stage is set!

Depressing the CARRIAGE RETURN key will start the DISPLAY of the next slide. After each slide is DISPLAYed RAPSODIE will pause while conversation is in progress. At the end of the DISPLAY, the message END OF DISPLAY is seen.

The pause in the DISPLAY is a request for something to be typed. The normal response is to depress the carriage return as mentioned, however, since RAPSODIE is asking for some input, any set of characters can be typed. APL is unique in that this set of characters can be treated as character input (and in this case ignored) or the characters can be scanned and executed if they form a value set of APL or functions instructions. This is the "back bone" of RAPSODIE. The following functions have been coded to allow the user to treat the video display unit exactly like a photographic projector.

FUNCTION SYNTAX	MEANING
BACK x	DISPLAY the slide that was displayed x slides ago
BACK TO x	DISPLAY the slide x again
SKIP x	DISPLAY the slide x slides ahead of the slide about to be displayed
SKIP TO x	DISPLAY the slide x and return the current slide.
GET set-of-keys	allows the user to select extra slides and insert it (them)

	before displaying the current slide
any valid APL expression	the expression is executed and control is returned to the DISPLAY function.

Each of these functions temporarily suspend the slide presentation to allow the stated operation, after which control is returned to the DISPLAY function. The GET feature allows the user to insert any forgotten information at any point without disturbing the entire presentation. The GET function acts exactly like a mini-SELECT function, i.e. the user is told how many slides satisfy his request and asks if he wants them all. A NO answer returns control to the DISPLAY function, otherwise the slide(s) are displayed and then control is returned.

B) *SAMPLE TERMINAL SESSION - DISPLAY*

DISPLAY

The projector
is made ready
A Carriage return is keyed

..

The first slide!

THIS IS A SLIDE
IT HAS 2 LINES

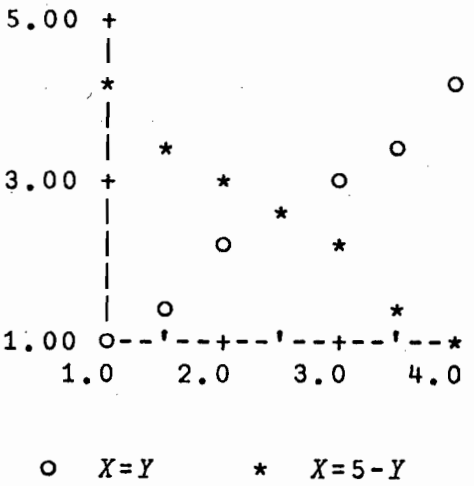
..

carriage return
to continue

..

DEMO PLOT

the next slide



o *X=Y* * *X=5-Y*

the '..'
is a slide
delimiter
use for positioning

..

A *CONTINUE NEXT PAGE*

Let's see that first one again.

```
BACK TO 1
..
```

There it is!

```
THIS IS A SLIDE
IT HAS 2  LINES
```

```
..
```

we may type an executable statement

```
(⍳3) × 7
7  14  21
```

we forgot to put in a slide so let us do it now

```
GET JOKE PUN
→   1 SLIDE FOR: JOKE PUN
→ WANT THEM ALL?     YES
```

..

TWO SONS DECIDED TO LEAVE HOME AND SETTLE IN AUSTRALIA. THERE THEY STARTED TO RAISE BEEF CATTLE AND BECAME VERY SUCCESSFUL.
THEY DECIDED , SINCE THEIR OPERATION WAS GETTING FAMOUS, TO FIND A GOOD NAME FOR THEIR RANCH; THUS, THEY WROTE HOME ASKING THEIR MOTHER TO SUGGEST A NAME. SHE WROTE BACK WITH THE NAME 'FOCUS'. 'WHY ''FOCUS''?' THEY WROTE BACK.
SHE REPLIED, 'BECAUSE THAT IS WHERE THE SUN'S RAYS MEET!!. '

..

we have retrieved successfully during the presentation!

```
A CONTINUE NEXT PAGE
```

Let us see what is coming

SKIP 10

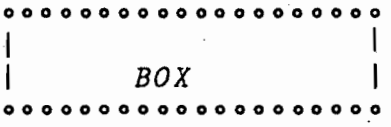
The computer knows there are only 4 slides not 10

END OF DISPLAY

let us continue

DEMO BOX

the presentation continues were we left it



carriage return to continue

the last slide it is the same as the first (we asked for slide 1 twice during SELECT

THIS IS A SLIDE
IT HAS 2 LINES

continue

There is no more the presentation is over

END OF DISPLAY

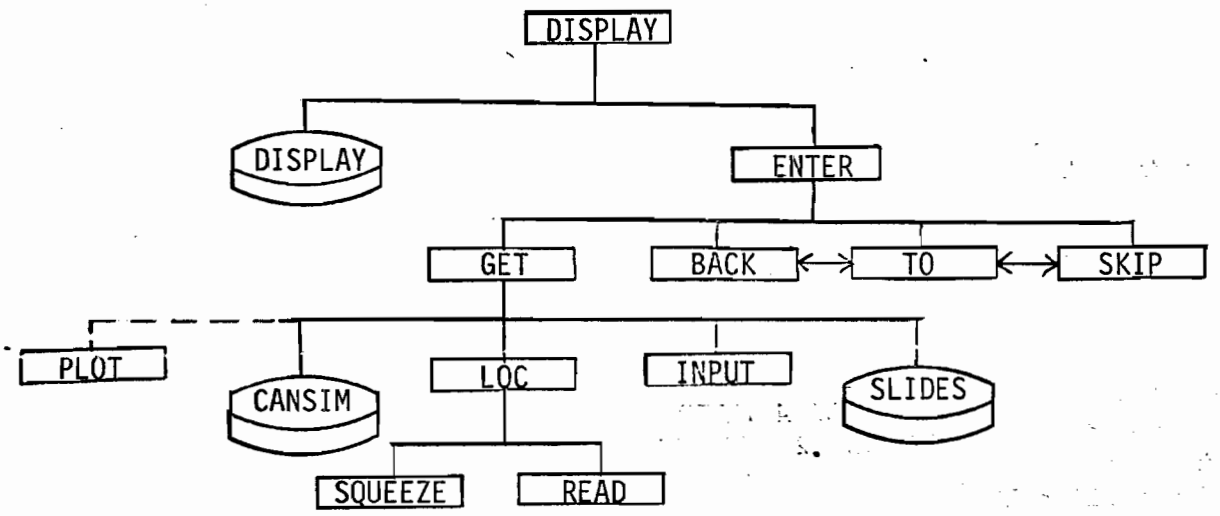


Fig. F
Display Calling Tree

c) Component Function Descriptions - DISPLAY

Fig F shows the DISPLAY function calling tree.

Below is a brief description of the DISPLAY group of functions

DISPLAY	Display selected slides like a slide projector. The following functions are used by DISPLAY
BACK n	described above
ENTER	a specialized function which accepts a character string, converts it (if possible) to an executable function and executes it.
GET string	described above
INPUT text	prints 'text' as a message to the user and allows a response on the same line
LOC k	same as function LOCATE (described earlier) except that the set of keywords (k) is entered as a parameter of the function. This function locates all the slides in the slide file described by the set of keywords 'k'
c READ f	reads component 'c' from the file tied to 'f' (the user is referred to the FILE SUBSYSTEM manual if this terminology is meaningless)
SKIP n	described above
TO n	allows a specific slide to be read. Used with functions SKIP and BACK

D) DETAILED FUNCTION LISTING OF DISPLAY GROUP IN ORDER OF APPEARANCE IN CALLING TREE

```
     ∇ DISPLAY;N;I;EXEC
[1]    I←1↑1,N←¯1+1↑1↓FE 10 20
[2]   LP:ENTER
[3]    FE 6 20 ,I
[4]    →((I←I+1)≤N)/LP
     ∇

     ∇ ENTER;R;EXEC
[1]    →((∧/' '=R),~∧/'GET '=4↑R←⍞)/0⍴E
[2]    R←(4↑R),'''',(4↓R),''''
[3]   E:→(0=0\0/R←3 ΔFD '∇EXEC',CR,'[1]',R,CR,'∇')/0
[4]    EXEC
[5]    →1,⍴R←6 ΔFD 'EXEC'
     ∇

     ∇ BACK X
[1]    FE 6 20 ,1⌈I-X
     ∇

     ∇ R←TO X
[1]    R←I-X
     ∇

     ∇ SKIP X
[1]    FE 6 20 ,(¯1+1↑1↓FE 10 20)⌊I+⌈X
     ∇

     ∇ GET X;J;A;NEW;KI;Y;Z
[1]    →(0=⍴A←LOC X)/0
[2]    →(('Q'=Z),'Y'=Z←(J←1)↑INPUT 20↑'→ WANT THEM ALL?')/0,LP
[3]    →1,⍴X←(INPUT 20↑'→ INPUT IDENTIFIERS:'),' '
[4]   LP:(¯2+Y⍳'⍞')↓Y←FE 6 10 ,A[J]
[5]    →((⍴,A)≥J←J+1↑1,⍴⍞)/LP
     ∇

     ∇ R←LOC S;T;K;I;X;Y;Z;NEW
[1]    R←⍳1↑NSLIDES,I←⍳1↑NIDS,⍴K←(S←SQUEEZE S),' ',KI←X←,,NEW←(0,
       IDLEN)⍴' '
[2]   F:→((0=⍴X),(Z←∧/' '=S←T↓S),¯1≠⍴R←(R∈(1+X←(((NIDS,⍴Y)↑IDS)∧.
       =Y←(¯1+T←S⍳' ')↑S)/I) READ KEYS)/R)/3↑A,D,F,KI←KI,X
[3]   A:→(Z,1)/2↑D,F,⍴NEW←NEW,[1]((0=⍴X),IDLEN)⍴IDLEN↑Y
[4]   D:'→ ',(((3⍴~T),T←3⍴(⍴R)≠0)/' NO',,'BI3' ΔFMT⍴R),' SLIDE',(
       (1≠⍴R)/'S'),' FOR: ',K,(⍴KI)↓KI←KI,X
     ∇

     ∇ R←SQUEEZE X;S
[1]    R←1↓(S∨1⌽S←X≠' ')/X←' ',,X
     ∇

     ∇ R←COMP READ FIL;I
[1]    →((⍴,COMP)=⍴R←⍳I←0)/0
[2]    R←R,FE 6,,FIL,(,COMP)[I←I+1]
[3]    →(I<⍴,COMP)/2
     ∇

     ∇ R←INPUT T
[1]    R←(⍴T)↓(-⍴T)↓⍞,⍞←T
     ∇
```

THE UTILITY FUNCTIONS

a) General description

The following utility functions have been included in this workspace primarily for programme development. The DELETE function, the most important of these, is used to delete obsolete or redundant slides.

By typing DELETE the delete group of functions is accessed. The user is asked to supply a set of keywords. Each slide referenced by this set of keywords will be made available to the user for possible deletion. DELETE will access the slides individually, by printing the actual keywords under which this slide was stored, and ask the user if he really wants to DELETE it. No response will access the next slide, otherwise, DELETE removes the slide, removes all keyword references to it, and appends the freed space to a garbage list for future use. Then the next slides are accessed and so on. The user may type MORE if he wishes to see more of the slide for better identification. If another group of slides is to be DELETEd, the user answers YES when he is asked if he wants to continue.

For the other functions in this group, only the brief description in section c will be given.

B) SAMPLE TERMINAL SESSION - UTILITIES

The computer types '→' before its dialogue

to delete enter the keys. the keys of the first are printed and it was deleted

also the 2nd

finally the third

verification that they have been deleted

```
                DELETE
→ INPUT IDENTIFIERS:DEMO SLIDE
→  NO SLIDES FOR: DEMO SLIDE
→ MORE?                    YES
→ INPUT IDENTIFIERS:SLIDE
→   3 SLIDES FOR: SLIDE

→ SLIDE 3. -COMP 35
→ WANT IT?                 NO
→ SLIDE 3. -COMP 35 DROPPED

→ SLIDE 2. -COMP 40
→ WANT IT?                 NO
→ SLIDE 2. -COMP 40 DROPPED

→ SLIDE 1. -COMP 52
→ WANT IT?                 NO
→ SLDE 1. -COMP 52 DROPPED
→ MORE?                    YES

→ INPUT IDENTIFIERS:SLIDE
→  NO SLIDES FOR: SLIDE
→ MORE?                    NO
```

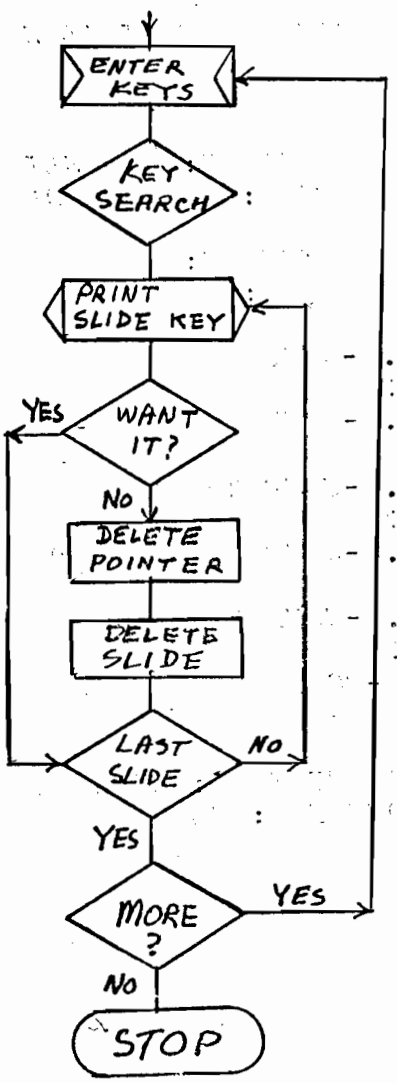


Fig. G
Delete Flow Chart

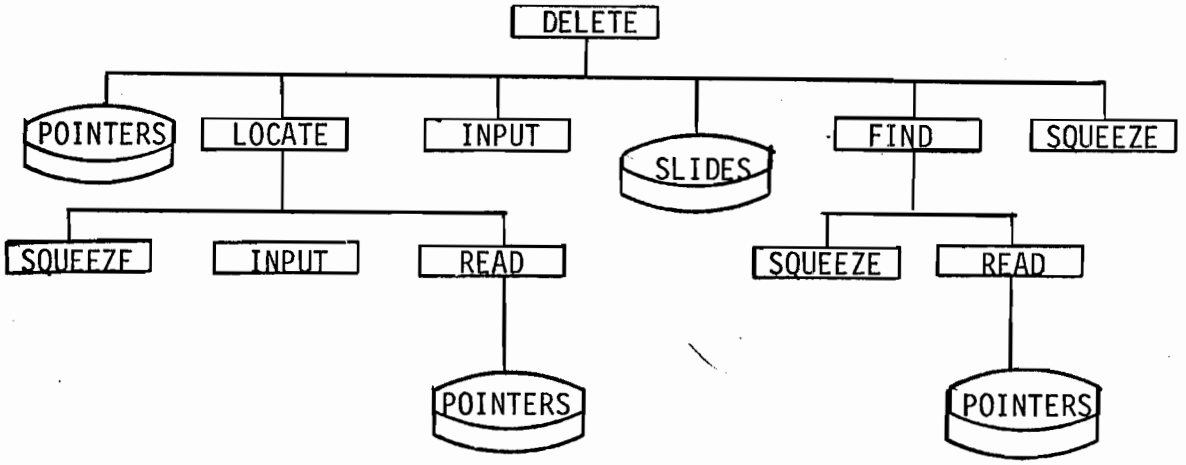


Fig. H
Delete Calling Tree

C) Description of Utility functions

Fig. G in the delete function flow chart
Fig. H is a function calling tree for DELETE.

START	Initializes the worspace by opening the required files and setting required constants. See the Introduction for a more complete description of this function.
DELETE	Allows redundant and obsolete slides to be deleted from the file, and appends the reusable space to a garbage list. All the functions used by Delete are described in the previous groups.
CLEANUP x	Allows for the reorganizing of the file x after extensive revision by adds, changes or deletes.
LIST x	Prints the array x with line numbers
DUMP x	Produces a list of either the keywords and the slides they reference (x=10) or a list of all the keywords used to define specific slides (x=30)
ERASE GARBAGE	ERASES any unwanted and extraneous constants before storing the workspace.
FIND x	A generalized Locate function which locates slides defined by the set of keywords x.

D) DETAILED FUNCTION LISTING OF UTILITIES GROUP

```
    ∇ START;X
[1]   FE 3,FE 18 ⍴ X←TABS 10
[2]   '1977269 SLIDES ' FE 2,SLIDES←10
[3]   '1977269 DISPLAY' FE 2 20
[4]   '1977269 KEYS   ' FE 2,,KEYS← 1 1 ⍴30
[5]   NSLIDES←1↑1↓FE 10,SLIDES ⍴ IDLEN←¯1↑⍴IDS←FE 6 30 1 ⍴ NIDS←
      1↑⍴IDS
    ∇

    ∇ DELETE;A;I;J;KI;NEW;B;X;Y;T;Z
[1]   →((I←1)↑0=⍴A←LOCATE)/1↑(D1+2),Z←0
[2]  D2:→('QMY'=T←1↑INPUT 20↑'→ WANT IT?')/3↑D5,D3,D1,⍴⍞←CR,'→ '
      ,(Y←(¯2+X⍳'¨')↑X←FE 6 10 ,A[I]),(,'⍞ -COMP⍞,BI3' ΔFMT A[I]
      ),CR
[3]   (0⍴' ') FE 3↑ 8 10 ,A[I],(J←1),B←FIND Y
[4]   (A[I],FE 6 10 1) FE 8 10 1
[5]  D4:((A[I]≠Y)/Y←FE 6 30 ,KI[J]+1) FE 8 30 ,KI[J]+1
[6]   →((J←J+1)≤⍴,KI)/D4
[7]  D1:'→ ',(SQUEEZE IDS[KI;]),' -COMP ';A[I];' ',((7⍴~T),T←7⍴'
      N'=T)/'KEPT   DROPPED'
[8]   →((I←I+1)≤⍴,A)/1↑D2,Z←0
[9]  D5:→(('N'≠1↑INPUT 20↑'→ MORE?'),1)/2↑ 1 0 ,Z←0
[10] D3:→D2,⍴⍞←(500⌊Z←500+Z)↑(0⌈Z-50)↓(⍴Y)↓X
    ∇

    ∇ CLEANUP X;N;I;Y
[1]   'TEMP' FE 2↑ 1 40 ,I←1↑1,N←¯1+1↑1↓FE 10,X←,X
[2]   (FE 6,X,I) FE 7 40
[3]   →((I←I+1)≤N)/2
[4]   (Y←, 0 11 ↓(X=FE 18)/[1] FE 19) FE 5,X
[5]   Y FE 15 40
[6]   START
    ∇

    ∇ ERASE XX
[1]   6 ΔFD,(3 ΔWS XX),' '
    ∇

    ∇ R←DUMP X;Y;N;I;A;M
[1]   →1↑A,⍴R← 0 51 ⍴(1+I←1↑0,(N←¯2+1↑1↓FE 10 10),A←((X←,X)=,SLI
      DES,,KEYS)/ 3 2)↑' '
[2]   →(((1↑⍴R←R,[1] IDS[I;],'12BI3' ΔFMT 1 12 ⍴12↑FE 6,X,1+I←I+
      1)<NIDS),1)/A,0
[3]   →((1↑⍴R←R,[1] 51↑SQUEEZE((¯2+1↑(,'¨') SRCH1 Y)↑Y←FE 6,X,1+
      I),,'⍞ S⍞,ZI2' ΔFMT 1+I←I+1)<N)/A
    ∇

    ∇ R←LIST X
[1]   ('I3,⍞  ⍞,',(,'I3' ΔFMT 1↓⍴X),'A1') ΔFMT((((1↑⍴X),1)⍴⍳1↑⍴X
      );X)
    ∇
```

DETAILED LISTING OF DELETE FUNCTION GROUP

IN ORDER OF APPEARANCE IN CALLING TREE

```
      LISTGRP 'DELETEFNS'
    ∇ DELETE;A;I;J;KI;NEW;B;X;Y;T;Z
[1]   →((I←1)↑0=ρA←LOCATE)/1↑(D1+2),Z←0
[2]  D2:→('QMY'=T←1↑INPUT 20↑'→ WANT IT?')/3↑D5,D3,D1,ρ⍞←CR,'→ '
      ,(Y←(¯2+Xι'¨')↑X←FE 6 10 ,A[I]),('⍞ -COMP⍞,BI3' ΔFMT A[I]
      ),CR
[3]   (0ρ' ') FE 3↑ 8 10 ,A[I],(J←1),B←FIND Y
[4]   (A[I],FE 6 10 1) FE 8 10 1
[5]  D4:((A[I]≠Y)/Y←FE 6 30 ,KI[J]+1) FE 8 30 ,KI[J]+1
[6]   →((J←J+1)≤ρ,KI)/D4
[7]  D1:'→ ',(SQUEEZE IDS[KI;]),' -COMP ';A[I];' ',((7ρ~T),T←7ρ'
      N'=T)/'KEPT   DROPPED'
[8]   →((I←I+1)≤ρ,A)/1↑D2,Z←0
[9]  D5:→(('N'≠1↑INPUT 20↑'→ MORE?'),1)/2↑ 1 0 ,Z←0
[10] D3:→D2,ρ⍞←(500⌊Z←500+Z)↑(0⌈Z-50)↓(ρY)↓X
    ∇

    ∇ R←LOCATE;T;S;I;X;Y;Z;K
[1]   R←ιNSLIDES ⋆ I←ιNIDS ⋆ S←K←(SQUEEZE INPUT 20↑'→ INPUT IDEN
      TIFIERS:'),' ' ⋆ KI←X←ι0 ⋆ NEW←(0,IDLEN)ρ' '
[2]  F:→((0=ρX),(Z←∧/' '=S←T↓S),¯1≠ρR←(Rϵ(1+X←(((NIDS,ρY)↑IDS)∧.
      =Y←(¯1+T←Sι' ')↑S)/I) READ KEYS)/R)/3↑A,D,F,KI←KI,X
[3]  A:→(Z,1)/2↑D,F,ρNEW←NEW,[1]((0=ρX),IDLEN)ρIDLEN↑Y
[4]  D:'→ ',(((3ρ~T),T←3ρ(ρR)≠0)/' NO',,'BI3' ΔFMTρR),' SLIDE',(
      (1≠ρR)/'S'),' FOR: ',K,(ρKI)↓KI←KI,X
    ∇

    ∇ R←SQUEEZE X;S
[1]   R←1↓(S∨1ϕS←X≠' ')/X←' ',,X
    ∇

    ∇ R←INPUT T
[1]   R←(ρT)↓(-ρT)↓⍞,⍞←T
    ∇

    ∇ R←COMP READ FIL;I
[1]   →((ρ,COMP)=ρR←ιI←0)/0
[2]   R←R,FE 6,,FIL,(,COMP)[I←I+1]
[3]   →(I<ρ,COMP)/2
    ∇

    ∇ R←FIND S;T;K;I;X;Y;Z;NEW
[1]   R←ι1↑NSLIDES,I←ι1↑NIDS,ρK←(S←SQUEEZE S),' ',KI←X←,,NEW←(0,
      IDLEN)ρ' '
[2]  F:→((0=ρX),(Z←∧/' '=S←T↓S),¯1≠ρR←(Rϵ(1+X←(((NIDS,ρY)↑IDS)∧.
      =Y←(¯1+T←Sι' ')↑S)/I) READ KEYS)/R)/3↑A,D,F,KI←KI,X
[3]  A:→(Z,1)/2↑D,F,ρNEW←NEW,[1]((0=ρX),IDLEN)ρIDLEN↑Y
[4]  D:KI←KI,X
    ∇
```

WHERE DO WE GO FROM HERE?

RAPSODIE in its present form does satisfy the requirements set out in the Introduction with only one exception: I have not explicitly stated how access to CANSIM will fit into the display functions. At the present time, if CANSIM data is needed, they can be retrieved through elementary APL and CANSIM access functions, formulated using much the same technique as for PLOT, stored as a variable, then filed. However, we do plan to add more flexibility to RAPSODIE by adding functions necessary not only to interface with the CANSIM access functions, but also to interface with APL GRAPHICS (the software supplied by Tektronix for use with their video display terminals in graphics mode).

At the present time, slides are stored as character strings which are retrieved and displayed when required. However, APL is able to convert properly formated character strings into executable APL functions. If such a character string is stored as a slide, when it is retrieved, identified as executable, converted to an executable function, then the resulting function can be used quite effectively. For example, we could store a generalized plot function to plot data specified in a pre-defined format. If the Tektronix software is available PLOT will produce line plots of this data. If the Tektronix software is not available, then PLOT will produce the standard character plot by default.

As implied above, we used the CANSIM data base to produce a plot or histogram which is then stored as a character string slide on the slide file. This means that the slide must be reformatted for more up-to-date data. However, we could combine these last two ideas (PLOT and CANSIM) to produce a plot of current data without having to store the result as a slide. The procedure might be something like this:

1) Store as a slide, a character string which can be converted to an executable function to produce a "standard" plot. (This procedure is outlined above.)
2) Write a small function like the GET function (described in the DISPLAY section) to retrieve a specific CANSIM series under keyword control. (The functions of RAPSODIE are able to do this now with only slight modifications).
3) Format the series in a pre-defined format.
4) Retrieve and convert the pre-defined PLOT function stored as a slide.
5) Executed the plot function using the formatted CANSIM series as data.

At the present time only the details need to be worked out to produce such a fantastic result!

When the above modifications are completed then all of the original requirements for the slide projection system will be complete.