EGIDE INPUT FORMS EDITOR

Deenadhayalan Govender

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg, *1986*

DECLARATION

I declare that this project report is my own unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University. Most of the information for this project was obtained while in the employ of Telecommunication Technologies (Pty) Ltd., which was considered to be an extension of the University of the Witwatersrand for the purpose of this project.

9th day of JANUARY 1986

Abstract

This project report describes the analysis, design and development of a screen-based editor for the input forms for the SA128 Digital Main Exchange used by the South African Post Office.

The development was done on a DEC VAX-11/750 minicomputer and a DEC VT125 terminal equipped with the Advanced Video Option. The VAX Form Management System was used extensively to describe the input forms and to reduce the complexity of the input-output operations to the terminal.

The Egide Input Forms Editor (IFE) developed meets its specification fully, and is functioning satisfactorily thus far. The response time is limited mainly by the speed of the VAX-11 Form Management System.

Acknowledgements

The author is grateful to TELECOMMUNICATION TECHNOLOGIES (PTY) LTD for allowing this project to be submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

CONTENTS Page

CONTENTS Page

CONTENTS Page

LIST OF FIGURES

LIST OF FIGURES continued

LIST OF TABLES

CHAPTER 1

THE E10 DIGITAL EXCHANGE AND ITS DATABASE GENERATION PROCESS

1.0 Introduction

This chapter briefly describes the *E10* digital exchange manufactured by Telecommunication Technologies (Pty) Ltd. A more detailed description of the database required for each E10 installation is then given, followed by the history and development of Egide, the database generation process. This is followed by a discussion of the input data required by this process, and the associated constraints with respect to the input data *format* required by Egide. These constraints form the major part of the specification for the Egide Input Forms Editor (IFE), the user-friendly tool to enable a semi-skilled operator to input the data for the database generation process. The description of the design and development of IFE constitutes the major part of this report.

1.1 The E10 Digital Exchange

This is merely a brief introduction to the E10 Exchange serving to bring into focus the problem to be solved. The interested reader is referred to Teltech (1985a) for a more detailed description.

The E10 Digital Main Exchange, known as the SA128E in South Africa, has been developed for 30-channel PCM operation. The maximum capacity of this

exchange is 384 PCM channels. A fully equipped E10 exchange configuration could typically consist of 45000 subscribers and 5600 trunk circuits.

Figure 1.1 is a block diagram of the E10 showing the four major subsystems. These are:
1) subscriber and circuit connections,
2) time-space-time switching network,
3) control units,
4) operation and maintenance centre (OMC), which can be shared by a number of E10's.
These subsystems are now described briefly.

1.1.1 Subscriber and Circuit Connections

The interface with a subscriber is via a subscriber equipment. The Subscriber Connection Unit (URA in Figure 1.1) can accomodate 1000 subscriber lines. These lines are concentrated to 120 in the URA, which is consequently also called an Electronic Line Concentrator. Traffic flexibility is ensured by allowing the traffic to be concentrated on 2, 3 or 4 PCM links, corresponding to a maximum configuration of 120 circuits. After concentration, the analog signals are pulse code modulated before being processed by the Switching Unit (CX).

A Concentrator can be located at the same site as the rest of the E10, in which case it is called a Local Concentrator (URAL), or it can be installed at a distance from the E10, in which case it is called a Satellite or Distant Concentrator (URAD). The URAD serves remote subscribers, and is connected to the E10 via 2, 3 or 4 PCM links.

The Multiplex Connection Unit (URM in Figure 1.2),

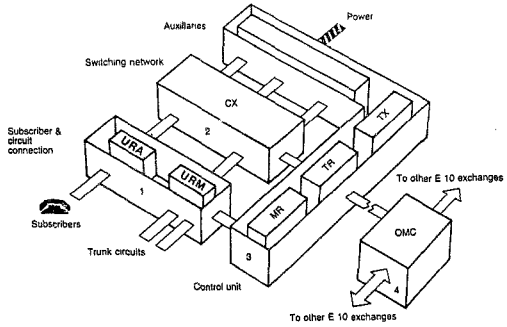


Figure 1.1 E10 Exchange structure

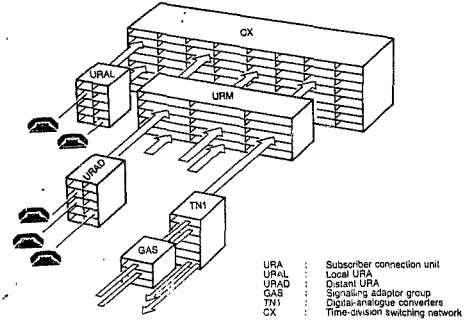


Figure 1.2 Subscriber and multiplex connection units switching network

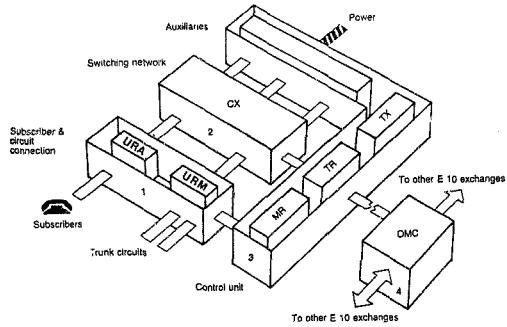


Figure 1.1 E10 Exchange structure

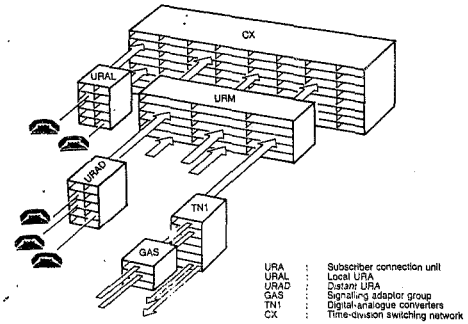


Figure 1.2 Subscriber and multiplex connection units switching network

is connected to incoming PCM links from Satellite Concentrators, electronic and electromechanical exchanges. In the latter case analog to digital conversion is necessary together with matching signalling before the URM. These functions are provided by the Analog-Digital Converter (TN1 in Figure 1.2) and the Signalling Adaptor Group (GAS) respectively.

1.1.2 The Switching Network

The time-space-time (TST) Switching Network (CX in Figure 1.2) provides 4-wire switching between the time slots allocated to the calling and called parties. The Switching Network can handle 384 30-channel PCM systems in the maximum configuration, and is modular in steps of 64 PCM systems.

1.1.3 The Control Unit

The operations carried out in the Subscriber Connection Units and the Switching Network are monitored by the Control Unit, the components of which are shown in Figure 1.3. All Control Unit components are duplicated to increase reliability.

1.1.4 The Multiregister (MR)

Calls are set up and released by the Multiregister, which is microprocessor-based. Each Multiregister is made up of 256 registers which are implemented in RAM. A fully equipped exchange has 6 Multiregisters, enabling the E10 to handle 52 call

attempts per second.

1.1.5 The Translator (TR)

The Translator contains files which describe all details of the subscribers and of the circuits connected to the E10. These include the correspondence between the directory number and the equipment number, classes of service, circuit groups, subscriber groups (PABX function) and the different charging rates.

1.1.6 The Charging Unit (TX)

The Chargir ·-it records the billing details of each subsc ·· ·nd is used in the invoicing of subscriber:

1.1.7 The Marker (MQ)

The Marker handles the routing of data between the various units of the exchange.

1.1.8 The OC

This is the interface between the E10 and the Operation and Maintenance Centre.

1.1.9 The Operation and Maintenance Centre (OMC)

The OMC consists of a general purpose Mitra minicomputer with periherals such as magnetic tape

and disc, printers and VDU's. Each OMC can maintain 100 000 subscribers distributed over a maximum of 6 different E10's.

The OMC operation functions are:
(a) exchange Translator memory changes when adding new subscribers, modifying subscriber discriminations, modifying inter-exchange routing, change to trunk circuits;
(b) updating individual subscriber accounts;
(c) testing subscriber lines and trunk circuits and recording test results;
(d) monitoring the traffic handled by the exchanges

The OMC maintenance functions are:
(a) running continuous fault-finding programs;
(b) registering faults;
(c) reconfiguring any E10 when any unit of that E10 signals a fault of sufficient severity to warrant disabling it;
(d) switching a faulty unit to test mode and analysing the resulting fault signals to identify the faulty card for maintenance personnel.

1.2 The Control Units

Since most of the data entered via IPE on the input forms is relevant to the control units, these units will now be discussed in greater detail. Figure 1.3 illustrates the control units of the E10.

The control units are used mainly to set up and release calls. Only those control units which access databases, viz. the Multiregister, Marker, Taxer and Translator, will be discussed in greater detail here as the intention is to bring the

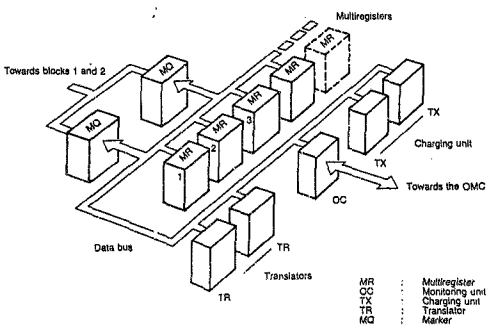


Figure 1.3 Control units

databases into perspective. This is necessary in order to familiarise the reader with the use and contents of the database, thus enabling him to appreciate the significance of the set of input forms which specify the database contents.

1.2.1 The Multiregister

The multiregister is a microprogrammed computer which can process 256 call set-up and call-release tasks simultaneously.

An example of the use of the Multiregister is now given. The case is that of setting up a call between two dial telephone subscribers connected to l.-al subscriber connection units (URAL's), after the Multiregister has been notified of the calling party off-hook condition. The characteristics of the calling party, such as connection unit and the equipment numbers, are stored by the Multiregister. It then interrogates the Translator for that subscriber's discriminations. These are, inter alia, pushbutton telephone, suspended service, call forwarding and abbreviated dialing facilities, and restricted service such as not being entitled to international calls.

If the calling party is still present and the discriminations do not forbid him from making a call, the Multiregister instructs the Switching unit to connect dial tone to the subscriber. The dial tone is obtained from the Frequency Sender/Receiver unit, and is connected through the Switching unit to the subscriber via the Connection unit (URAL).

1.2.2 The Translator

The Translator provides the Multiregisters, on request, with the subscriber and circuit data required to set up and release calls. This data is stored in files, the structure of each file differing with its function. The Translator is stored in RAM, each word being 16 bits long with a 6 bit self-correcting code. The maximum memory capacity is 768 kilowords.

The major functions of the Translator are:
a) provide the subscriber or trunk circuit data to the Multiregister;
b) translate prefixes and the first digits received;
c) translate called party directory numbers into equipment and Connection unit numbers, or trunk circuit numbers, as appropriate;
d) control services such as call forwarding abbreviated dialing, hot line and recorded call.
e) execute, together with the OMC, operating, management, maintenance and test functions.

The main Translator files and their contents are listed in Table 1.1, and an example of the use of the Tranlator in the analysis of directory numbers is described in Figure 1.4.

1.2.3 The Taxer

The Taxer calculates the charge for each call according to a number of factors. These include the time and type of day, the type of calling party equipment and the type of call (local, national,

Table 1.1 Main translation files

Category	File	Description
Memory description and occupancy	FIAF	File base addresses (physical location of files)
	FOCE	Record occupancy*
Telephone equipments	FOUR	Type of connection unit - traffic observation indications
	FALO	Discriminations and directory numbers of local subscribers
	FCIR	Identical to FALO, but for trunk circuits
Analysis and Routing	FPREA	Prefixes dialled by subscribers, I/C trunk access codes
	FIANA	1 file per type of dialling, regional, national, etc.
	FROU	First choice routing numbers, plus charging data
	FACH	O G routing, local routing and special files (transfer to recorded announcements, etc.)
Circuit groups	FIFA	Circuit group characteristics
	FIDET	First and final addresses of circuit equipments and number of next section**
	FIART	I/C sections
Subscribers	FARI	Equipment numbers of local subscribers
	FGROU	Number of first subscriber line section
	FADE	Additional subscriber discriminations
Ancillary (test and observation)		

Glossary of terms:

*One file:	several records	**A section:	a number of consecutive equipments
One record.	several identical items	A circuit group:	set of several sections (trunk circuits)
One item:	set of 16-bit words	A group.	*set of several sections* (subscribers)

List of principal translation files

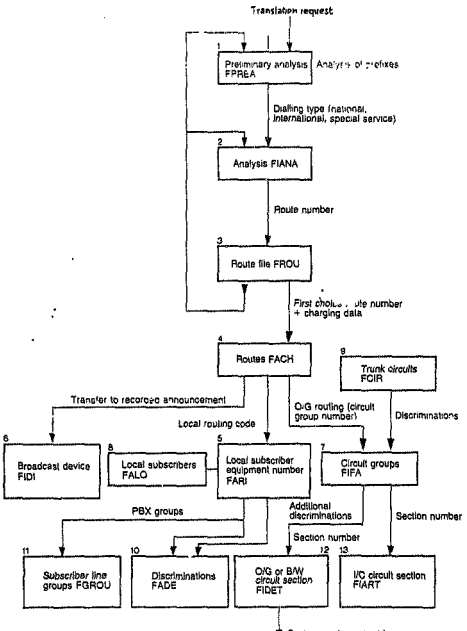


Figure 1.4 Example of Translator file use

etc.). It can also draw up detailed billing information for subscribers if required.

The operation of the Taxer is controlled by the Multiregisters as follows:
a) on receipt of the appropriate command from a Multiregister, the Taxer begins charging a call and stops at the end of the call;
b) receives metering pulses from distant exchanges over the trunk circuits;
c) sends out tones and signals eg. subscriber private metering;
d) sends subscriber accounts to the OMC on request;
e) sends detailed billing information to the magnetic tape unit on failure of the OMC.

1.2.4 The Marker

The Marker routes the main messages between the various units of the E10. The major functions of the Marker are :
a) receive off-hook and on-hook conditions from the Connection units;
b) pass commands, received from the Multi-registers, to the Switching network to set up connections, release connections and to send out tones;
c) interrogate the Switching network to obtain the addresses of the calling parties during call releases;
d) send messages to and receives messages from the OMC.
e) request the following from the connection units:
i. subscriber equipment tests,
ii. surveillance of subscriber and trunk circuits,
iii. interrogation of subscriber and trunk circuit

equipment.

1.3 The History and Development of Egide

Egide is a French acronym for :

Ensemble
pour la Generation
et l'Interrogation
des Donnees d'un
systeme E10,

which means the suite of programs for the generation and testing of the data for the E10 system.

The first French version of Egide was implemented on a Mitra minicomputer, with punched cards being used to input the data. This program was unfriendly and had a typical run-time of 24 hours. Local use of the program was ruled out by its unfriendliness and the necessity for the user to have an in-depth knowledge of the inner workings of Egide. Thus it was necessary initially to send the E10 configuration data to France, clarify queries with numerous telexes and phone calls, and hope to receive the magnetic tape in time for the commissioning stage of the site installation.

The lead time for such a magnetic tape was three months. SAPO can usually supply the complete information for an exchange two months before the commissioning date. Thus incomplete information had to be used in the database generation, with any further additions or modifications being done manually on site. This was a tedious and time

consuming process.

In addition any queries by SAPO in connection with database format changes also had to be forwarded to France, with unacceptably long periods between the requests and the implementation. These factors justified the development of a local Egide which would be faster and friendlier.

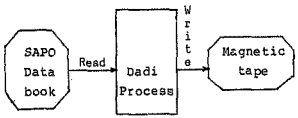
1.3.1 The Local Egide, Dadi

The local Egide is called Dadi, short for Data Dispatcher. The rationale behind this name was that the program essentially read data from the input forms, transformed it, and then dispatched it to the appropriate files.

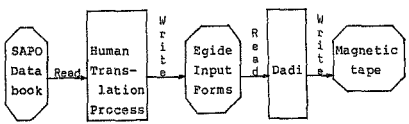
At the outset, the developers of Dadi were faced with the major choice illustrated in Figure 1.5. These options were either :
(a) Dadi could directly process the 5 different input forms as defined by SAPO, and which was used by them to specify the configuration of an exchange, or
(b) the 5 input forms in the SAPO Databook could manually be translated into the 33 different input forms used by Egide which would then be processed by Dadi.

Option (b) was chosen for the following reasons :
(i) The SAPO input forms were not rigorously defined and thus were not conducive to automation.
(ii) Option (a) would result in a manifold increase in complexity in Dadi. This was undesirable because Dadi had to be completed in the shortest possible time.

(iii) Dadi would be more logically decomposed into modules by associating each module with the corresponding Egide input form.



(a) Option for Dadi to process the SAPO databook directly.



(b) Option for Dadi to process the Egide Input Forms.

Figure 1.5 The options for the Dadi process.

1.4 The Egide Input Forms

Of the three databases generated for each E10 site the translator files are the largest and most varied with respect to the information contained. Since the scope of this chapter is not to discuss in detail all aspects of the E10 databases but only to bring into perspective the role of the input forms, only the most important translator input forms and the corresponding files will be discussed as this is sufficient to meet the goal.

1.4.1 Input Forms - Database Files Correspondence

Table 1.2 lists the most important translator files and the corresponding input forms which serve as the information sources for the Egide program during the generation of these files.

A full description of all the Egide input forms is not possible here because of the magnitude of the task. However a description of input form .ABD is given in Figure 1.6 and Table 1.3 as an example of the type of data contained in the input forms. A complete set of descriptions for the input forms can be found in Teltech (1985b).

1.5 The Requirements of IFE

Selecting option (b) in section 1.3.1 meant that the format of the input forms had to use the 80 column character format used on the punched cards input to Egide. Further constraints on the format of the input data to egide were added by the Dadi development team, resulting in the specification described in 1.5.1 for the Egide Input Forms Editor.

1.5.1 IFE Specification

a) Operator should be prompted to select one particular form to be edited.

b) New form definitions must be easily added to existing ones, and existing form definitions must

TABLE 1.2: Relationship Between *Input Forms* and Translator Files

Input Form	Contents	Files Generated
.DEF	File definitions	FIAF
.NSR	Number indices	FINUS
.URS	Connection units	FOUR, FIPIR, FLAC, FINUR, FEQL, FPRP, FICSA
.DIF	Multi-line transmission units	FIDIF, FARI
.ABS	Ordinary subscriber equipments	FIEQU, FARI, FOUR
.ABD	Discriminated subscriber equipments	FIEQU, FARI, FOUR, FADE, FENU
.GRP	Subscriber groups	FGROU, FIGA, FIGD, FIRGT, FARI
.FSC	Circuit groups	FIFAC, FIART, FIDET, FIRFT, FIEQU, FIFSC
.CEC	Circuit tests	FESC
.PAM	Multiregister parameters	FIPAM
.ACH	Routings	FACH, FITRAD
.AFH	Time-dependent routings	FAFH
.CAD	Timing rates calender	FIHAC
.TYJ	Type of day	FICAC
.CTX	Destination categories	FCDTX
.ANA	Analysis	FIANA, FROU, FRCT, FRZG
.PRE	Pre-analysis	FPREA, FCDST

TABLE 1.2 (contd.): Relationship Between Input *Forms* and Translator Files

Input Form	Contents	Files Generated
.SDA	Direct inward dialling	FGROU, FARI, FADE FROU, FIFAC, FIANA
.RNV	Call transfers	FIREN, FROU
.ERC	Alarms	FERC, FEMO, FACT
.ERS	Satellite alarms	FERS
.RPG	VDU alarm display	FPGV

be easily modified.

c) Desired editing features:
- full user control with up/down scrolling
- delete line
- insert line (optional)
- fast cursor advance/backup.

d) The sequential file created by IFE must have records which are all 80 bytes long, and must conform to the structure as shown in figure 1.7. IFE must have the possibility to select unused fields which will be skipped during data input and cursor movement. Unused field 'control-data' should be accessible with the Backspace command.

e) Facility to sort the data area of the resulting file according to user-specified fields.

f) Full validation of individual fields within records.

g) The program must be written in Fortran 77 since this was the main area of expertise of the development and maintenance personnel.

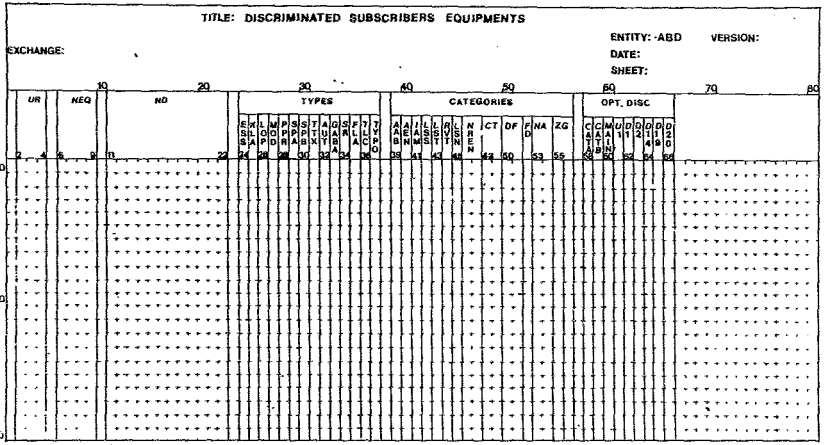


Figure 1.6 Input form .ABD

Table 1.3 ABD Field Descriptions

Field	Description	Option	Type
UR	Connection unit number	COM	DEC
NE	Equipment number	COM	DEC
ND	Calling sub local number	COM	CAR
ESS	Test equipment	OPT	BIN
KLA	Push button set	OPT	BIN
LOP	Operator line	OPT	BIN
MOD	Right to modification	OPT	BIN
PPR	Call office	OPT	BIN
SPA	Outgoing subscriber	OPT	BIN
SPB	Incoming subscriber	OPT	BIN
TTX	Remote metering	OPT	BIN
AUT	Special TYQ	OPT	HEX
GABA	Test parameters set	OPT	DEC
FLA	Flashing button	OPT	BIN
TLC	Concentrated line	OPT	BIN
TYPO	Operator type	OPT	DEC
AAB	Absent subscriber	OPT	BIN
AEN	Call recording	OPT	BIN
IAM	Identification of malicious calls	OPT	BIN
LSS	Essential line	OPT	BIN
LST	No charge line	OPT	BIN
RVT	Call forwarding	OPT	BIN
LSN	Hot line	OPT	BIN
NREN	Call transfer number	OPT	DEC
CT	Routing and charging category	OPT	DEC
DF	Reroute to recorded announcement	OPT	DEC
FD	Detailled billing	OPT	DEC
NA	Short code dialling	OPT	DEC
ZG	Area of calling subscriber	OPT	DEC
CATA	Called subscriber category	OPT	HEX
CATB	Caller subscriber category	OPT	HEX
MAIN	Test device equipment	OPT	HEX

Table 1.3 contd. ABD Field Descriptions

Field	Description	Option	Type
D15	Additional discrimination	OPT	HEX
D1	Additional discrimination	OPT	BIN
D2	Additional discrimination	OPT	HEX
D14	Additional discrimination	OPT	BIN
D19	Additional discrimination	OPT	BIN
D20	Additional discrimination	OPT	HEX

Abbreviations: OPT : Optional
COM : Compulsory
BIN : Binary
DEC : Decimal
HEX : Hexadecimal

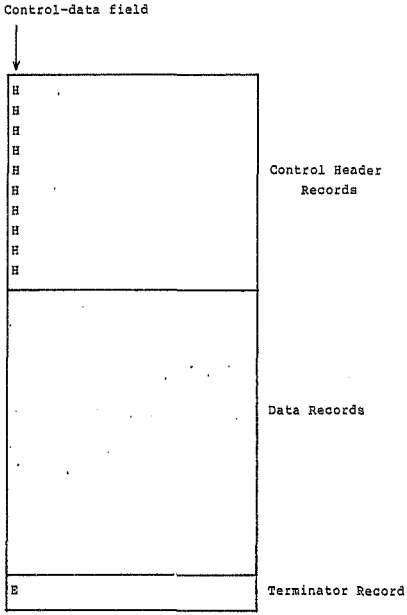


Figure 1.7 Required file structure for input form file

CHAPTER 2

THE DESIGN OVERVIEW OF IFE

2.0 Introduction

This chapter describes the design of the Egide Input Forms Editor to meet the specification of paragraph 1.5.

The major factors which influenced the design of IFE are described first. The software overview is described by means of a data flow diagram showing IFE in perspective in Figure 2.1, followed by the design of IFE using the process-resource method as described by Walker(1984). The result of this design is a set of processes which are then described in the Program Description Language (PDL) as described by Walker(1984) and Caine(1975). The behaviour of the resources used by these processes is also described.

2.1 Factors which influenced the design of IFE

2.1.1 Computer Workload

The computer on which the IFE was developed is extremely overloaded. Consequently free disc space is often limited to the extent that the VAX-VMS Editor aborts owing to insufficient working area in the case of large files. Thus it was necessary to minimise the working area of the program.

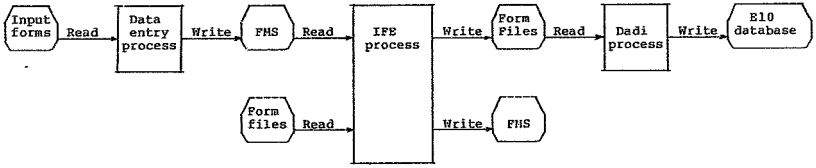


Figure 2.1 Data flow diagram showing IFE in perspective

2.1.2 VAX-11 Form Management System

This tool, described in paragraph 2.3.1 and Appendix A, was used because it immediately fulfilled the second requirement of the IFE specification, i.e. easily adding new form definitions and modifying existing ones. Furthermore development time had to be minimised as the IFE was already urgently required by the Egide development team. Thus the use of any existing tools was advisable.

2.1.3 Program Lifetime

At the time of development it was envisaged that IFE would be a temporary measure, the reason being that many man-hours (typically 120) are lost in transferring the information from the SAPO Databook to the computer. This process would eventually be automated provided that SAPO supplied the Databook on magnetic tape. The reasons this approach was not pursued initially were the lack of available personnel and tight time constraints. Therefore IFE was not required to be a very sophisticated editor.

2.1.4 *User Considerations*

The user of IFE would be a non-technical person such as a typist. Thus simplicity was necessary.

2.2 Process Resource Design of IFE

The process-resource method will not be described here in detail but will be illustrated by its

Application to IFE. The interested reader is referred to Walker(1984) for further reading.

2.2.1 First Level Design

Figure 2.2 shows the first level of the design of IFE. Here, the IFE process represents the entire systems activity, its interaction with the environment consisting of those with the resources shown.

2.2.2 Second Level Design of IFE

In this level of the process-resource method of design the major activities of the IFE process are identified. Each activity is functionally separate, and is associated only with those resources necessary to perform its function. The use of state diagrams later as a scheduling tool was found to be helpful in the separation of these major processes. This is due to the necessity of "fitting in" the processes between state transitions (D'Angelo, 1978). Table 2.1 shows the second level design of IFE.

Further process decomposition was unnecessary as each activity identified in the second level of design can be comfortably described by its own process-resource diagram, the associated Program Description Language (PDL) description of the process, and the resource behaviour description. These descriptions follow.

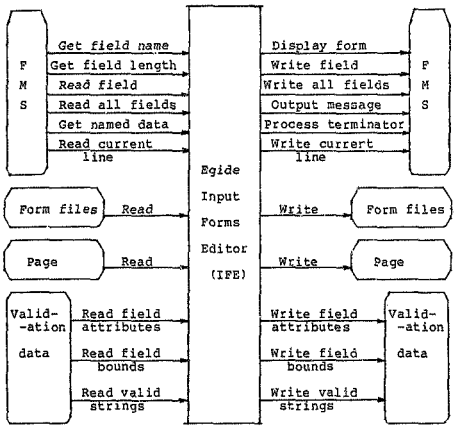


Figure 2.2 IFE: First level design

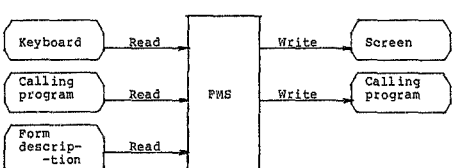


Figure 2.3 FMS behaviour

Table 2.1 Second Level Design of IFE - Major Activities of IFE Process

ACTIVITY	RESOURCES	OPERATIONS
P2:		
Select	FMS	Open form library.
Form		Display form.
		Get Named Data.
		Read all field values.
		Output message.
	Validation	Write.
	data	Read.
P3:		
Select	FMS	Open form library.
Unused		Display form.
Fields		Get Named Data.
P4:		
Edit	Form file	Open.
Form		Read.
Header		Write.
	FMS	Display form.
		Read all field values.
		Output message.
		Output all field values.
		Output data to current line of scrolled area.
P5:		
Edit	Form file	Open.
Egide		Read.
Form		Write.
	FMS	Display form.
		Get Named Data.
		Output message.
		Output data to current line of scrolled area.

Table 2.1 contd. *Second Level Design* of IFE - Major Activities of IFE Process

		Process terminator.
		Get current field name.
		Get specified field value
		Close form library.
	Page	Write.
		Read.
P6:		
Select Options	FMS	Open form library.
		Display form.
		Return all field values.
		Output message to screen.
P7:		
Get SORT Information	FMS	Open form library.
		Display form.
		Output message to screen.
		Get specified field value
P8:		
Delete Blank Records	Form Files	Read.
		Write.
P9:		
Sort Records	Form Files	Read.
		Write.
	VMS SORT Utility	Write.
		Read.

2.3 Resource Behaviour

2.3.1 FMS Behaviour

The FMS resource has 3 major components. The FMS

Form Editor component enables the user to create and modify computerised form descriptions to be used for form applications under program control. The FMS Form Utility allows the creation of form libraries, while the FMS Form Driver displays forms and data that users input in response to the forms.

The Form Driver is a library of routines that reduces programming effort by *manipulating* the screen, checking user reponses against the form description, *and displaying existing* help forms when requested. Thus the programmer is spared the *tedium of low level input-output* details. The Form Driver has 2 major interactions. The first is with the form decription, which is created with the Form Editor, and the second is with the user who fills in the fields in a displayed form.

Figure 2.3 summarises the behaviour of the FMS resource.

Operations on FMS

(i) Set Channel (channel)
This call sets the channel to be used to open or access a form library file.

(ii) Open Form Library (form library name)
The Form Driver opens the specified form library on the *current* form library channel.

(iii) Close Form Library
The Form Driver closes the form library which is open on the current channel.

(iv) Display Form (form name)
The Form Driver clears the entire screen and

displays the specified form. When first displayed the form includes all text and the default values for all fields.

(v) Get Named Data (label, value)
This call is used to retrieve by name data that has previously been associated with a form as named data. This data is not displayed with the form.

(vi) Get Field Name (current field name)
This routine does not call the Form Driver. It returns the current field name from the Form Driver argument block.

(vii) Get Field Length (field length, field name)
The Form Driver returns the length of the specified field.

(viii) Read Field (field value, terminator, field name)
The Form Driver places the cursor in the initial position of the specified field and accepts input by the user in that field. The input is checked against the legal data types for that field as specified in the form description. In the event of an error input is refused and an appropriate error message is displayed in the message area of the screen.

(ix) Write Field (field value, field name)
The Form Driver displays the value in the field specified.

(x) Read All Fields (field values)
This routine can only be used for areas of the form which are not scrolled. The Form Driver returns a concatenated string of the current values for all

fields in the form.

(xi) Write All Fields (field values)
This routine can only be used in areas of the form which are not scrolled. The Form Driver outputs the data specified to all fields in the form. The data must be specified as a concatenated string of all field values.

(xii) Get Current Line (line value, field name)
The Form Driver returns the contents of the scrolled line as a concatenated string of all field values in argument "line value". Argument "field name" can be the name of any field in the scrolled area.

(xiii) Write Current Line (field name, line value)
The Form Driver outputs the data specified to the current line of the scrolled area. The scrolled area is identified by specifying the name of any field in that area.

(xiv) Process Field Terminator (terminator)
The Form Driver processes the field terminator specified.

(xv) Output Message (line value)
The Form Driver clears the last line of the screen and displays the specified string on that line. This line is reserved for error messages and help text.

2.3.2 Page

The Page data structure is a 20 element array, each element being 80 characters long. Page is

effectively a "window" into the file being edited, and each record in array Page represents a line on the screen. Using a window into the file avoids the use of large working areas and the associated problems decribed in 2.1.1. Initially the page to be edited is read into Page and these records displayed on the screen. All changes by the operator are not immediately reflected in the file but in Page. When a line is scrolled off the screen, the corresponding record in Page is written to the file. At the end of the editing session all records in Page are written to the file.

When a field is written to by the user, the value is written to the appropriate position in Page. Thus it is necessary to calculate the record number and the field position within the record. To achieve this three pointers are associated with Page. These are Top and Bot for the first and last records in the window respectively, and Cur for the record currently occupied by the cursor.

Operations on Page

(i) Write (record, pointer 1, pointer 2, value)
The character string specified in value is written to the field whose position is specified by the two pointers, which field is in the element of page indexed by record.

(ii) Read (record, pointer 1, pointer 2) value
The character string in the specified record and in the field specified by the two pointers is returned in value.

2.3.3 The Validation Data

As described in Appendix A, the attributes of each field in the Egide input form has to be described in the Named Data area of the appropriate form description. When a particular Egide form is being processed, these field attribute are stored in a data structure called Validation Data. The elements of Validation Data are :

1. Starting position of field
2. Ending position of field
3. Type of validation
4. Lower bound
5. Upper bound
6. Valid character strings.

Note that in the actual implementation of this resource in Fortran 77 requires that the validation data be stored in two related data structures because it is not possible to have character and integer elements in the same data structure. However, since the concept of information hiding is encouraged by the resource behaviour description, these implementation specific details can be avoided by using the following operations on Validation data independent of the language of implementation.

Operations on Validation Data

(i) Write Field Attributes (Field no, Start position, End position, Type)

Stores the last 3 arguments for the specified field number.

(ii) Write Field Bounds (Field no, Upper bound, Lower bound)
Stores the last 2 arguments for the specified field number.

(iii) Write Valid Strings (Field no, Valid strings)
Stores the second argument for the specified field number.

(iv) Read Field Attributes (Field no) Start position, End position, Type
Returns the field attributes for the specified field number.

(v) Read Field Bounds (Field no) Upper bound, Lower bound
Returns the field bounds for the specified field number.

(vi) Read Valid Strings (Field no) Valid strings
Returns the valid character strings for the specified field number.

2.3.4 Fields used

This is a logical array of 40 elements which is used to record those fields which are selected as unused by the operator. No operation is performed on this resource besides that of reading and writing to an array. Indexing is by means of the field number. During the form editing process a field is skipped if it is unused.

2.3.5 Valid Form Names

This is an array of 40 elements, each element being 3 characters long. The operator is prompted to select a form by means of form Select, the Named Data of which contains all valid form names. This Named Data is read into Valid form names. The form selected by the operator is checked against the valid form names. Operations on Valid form names, like those on Fields used, are restricted to the simple read and write on an array, the indexing being done by the field number.

2.4 Process Descriptions

This section describes the major activities of IFE identified in the functional decomposition stage of the design using PDL. Each process is described by means of a process-resource diagram and a procedure in PDL.

For all processes the status is not returned to the calling program. Any exception is reported at the point at which it occurs by means of the Vax/Vms Exception Handler and, where necessary, IFE-specific error messages. The error reporting is such that the user is given the maximum possible information to enable him to easily understand and correct any errors.

The advantage of not returning the status to the calling program is that extensive passing of error related parameters is avoided. Thus a return of control to a calling program carries with it the implicit status of success.

2.4.1 Select Form Process

The Select Form process is described in Figure 2.4 and the associated procedure in Figure 2.5.

2.4.2 Display Header Process

The process-resource diagram for this process is shown in Figure 2.6 and the PDL description in Figure 2.7.

2.4.3 Unused Fields Selection Process

Figure 2.8 shows the Unused Fields Selection process-resource diagram and Figure 2.9 the PDL description for this process.

2.4.4 Edit Form Process

Figures 2.10 and 2.11 depict the process-resource diagram and the PDL description for this process respectively.

2.4.5 Select Options Process

The Select Options process is described in Figure 2.12 and the associated procedure in Figure 2.13

2.4.6 Delete Blank Records Process

This process is described in Figures 2.14 and 2.15.

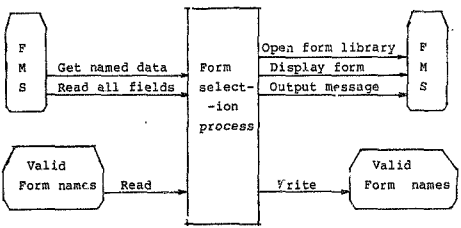


Figure 2.4 IPE: Form selection process-resource diagram

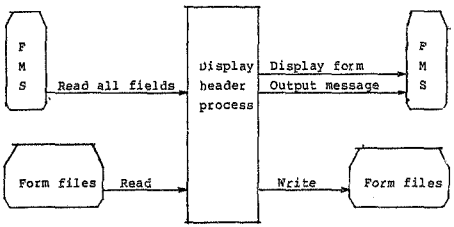


Figure 2.6 IPE: Display header process-resource diagram

```
Procedure  Form Selection
*************************************************
* A routine to prompt the user to specify the   *
* Egide form to be edited, whether certain fields *
* are to defined as unused, and whether the Egide *
* form header is also to be edited.             *
*************************************************

Inputs:     * None *
Outputs:    Form name

Variables:
  Char:
    Single:
      Global:Partition:A
            Form name

    Array:
      Local:
            Valid form names

Begin:
  Open Form Library (Formlib1)
  Display Form (Select)
  Get Named Data (Ndata,Valid names)
  * Store Valid names in Valid Form Names *
  While (* form is incomplete and selected
                form name is invalid *) Do
    Read All Fields (Field values)
    * Validate user-selected form name *
  End while
End:
End procedure:
```

Figure 2.5 Form Selection

```
Procedure      Display Form Header
*****************************************************
* A routine to display and edit the Header of an    *
* Egide form.                                       *
*****************************************************

Inputs:  Form name
Outputs: * None *

Constants
  Char:
    Single:
      Local:
        Ndata

Variables:
  Char:
    Single:
      Local:
        Header Data
  Char:
    Single:
      Global:Partition:A
            Form name

Begin:
  Display Form (Header)
  * Retrieve Header data from input form fil. *
  Write All Fields (Header data)
  While (* form is incomplete *) Do
    Read All Fields (Header data)
  End while
  * Save Header data in input form file *
End:
End Procedure:
```

Figure 2.7 Display Header

```
Procedure      Display Form Header
*****************************************************
* A routine to display and edit the Header of an    *
* Egide form.                                       *
*****************************************************

Inputs:  Form name
Outputs: * None *

Constants
  Char:
    Single:
      Local:
        Ndata

Variables:
  Char:
    Single:
      Local:
        Header Data
  Char:
    Single:
      Global-Partition:A
            Form name

Begin:
  Display Form (Header)
  * Retrieve Header data from input form file *
  Write All Fields (Header data)
  While (* form is incomplete *) Do
    Read All Fields (Header data)
  End while
  * Save Header data in input form file *
End:
End Procedure:
```

Figure 2.7 Display Header

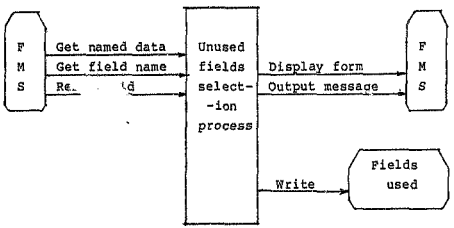


Figure 2.8 IFE: Unused fields selection process-resource diagram

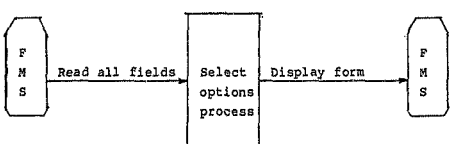


Figure 2.12 IFE: Select options process-resource diagram

```
Procedure Unused Fields Selection
*********************************************************
This routine enables the user to define certain *
* fields as unused, ie. these fields will be       *
* skipped during the editing of the Egide form.    *
*********************************************************

Inputs:  Form name
Outputs: Fields used

Constants
  Char:
    Single:
      Local:
        Fldno

  Integer:
    Single:
      Local:
        Used
        Unused
        Maximum fields
        Tab
        Backspace
        Return

Variables:
  Char:
    Single:
      Local:
        Unused fields
        Field name
        Field value
```

Figure 2.9 Unused Fields Selection

```
      Global:Partition:A
            Form name

  Integer:
    Single:
      Local:
        No of fields
        Field no

      Array:
        Global:Partition :B
          Fields used of size Maximum fields

Begin:
  * Use Form name to formulate name c    nused
                                  fields    *
  Display Form (Unused fields)
  Get Named Data (Fldno,No of fields)

  Field no:=1
  While (* form is incomplete *) Do
    Get Field Name (Field name)
    Read Field
        (Field value,Terminator,Field name)
    If (* field is used *) Then
      Field used:Field no:=Used
    Else
      Field used:Field no:=Unused
    End if

    If (Terminator = Tab) THEN
      Field no := Field no + 1
    Else if (Terminator = Backspace) Then
```

Figure 2.9 contd. Unused Fields Selection

```
      IF (* Field no exceeds 1 *) Then
        Field no := Field no - 1
      Else
        * Ring terminal bell to indicate error *
      End if
    Else if (Terminator = Return) Then
      * check if all fields defined as unused *
      If (* all fields are unused *) Then
        Output Message (Error:All fields defined as
                                              unused)
        Field no := 1
      End if
    End if
  End while
End:
End Procedure:
```

Figure 2.9 contd. Unused Fields Selection

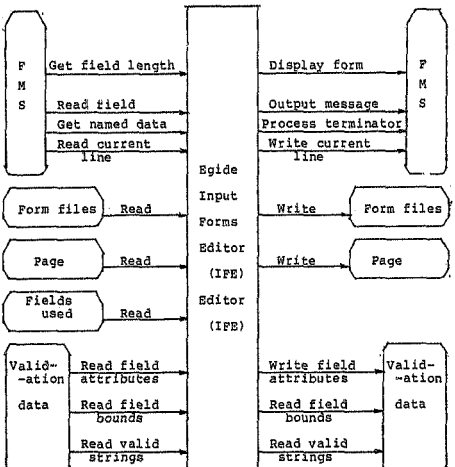


Figure 2.10 IFE: Edit form process-resource diagram

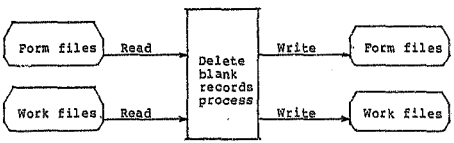


Figure 2.14 IFE: Delete blank records process-resource diagram

```
Procedure Edit Egide Form
*****************************************************
* This procedure provides the editing capability    *
* for any Egide input form.                         *
*****************************************************

Inputs:   Fields used
          Form name

Outputs:  * None *

Constants
  Integer:
    Single:
      Local:
            Tab
            Backspace
            Return
            Uparrow
            Downarrow
            First page
            Last page
            Page forward
            Page backward

Variables
  Integer:
    Single:
      Local:
            Terminator
            Field no
            Top
            Cur
            Bot
```

Figure 2.11 Edit Form

```
    Array:
      Globr :Partition :B
        Field used of size Maximum fields

  Char:
    Single:
      Local:
            Field name
            Field value

      Global:Partition:A
            Form name

Begin:
  * extract No of fields and Page size from
            Named Data of form description *
  * Initialise pointers Top,Cur,Bot *
  If (* new file *) Then
    * Clear screen data area and initialise Page *
  Else
    If (* error in opening file *) Then
      * Signal error *
    Else
      * Read first page from file into Page
                        and display it *
    End if
  End if

  * Read validation data from Named Data of form
            description into Validation Data *

  Terminator := Tab
  Field no := 1
  If (* current field is unused *) Then
    * move to next used field *
```

Figure 2.11 contd. Edit Form

```
  End if
  Do While (* editing session is not terminated *)
    * Read in current field value into FIELD *
    Get Field Name (Field name)
    Do While (* Field value is invalid *)
      Get the Value from the Specified Field
          (Field value, Terminator, Field name)
      * validate Field value *
      If (* Field value is invalid *) Then
        Output to last line
            of Screen (Illegal value)
        * ring terminal bell *
      End if
    End do
    * Update Page with Field value *

   Case Terminator of:
            Tab: * Move to next used field *
      Backspace: * Move to previous used field *
         Return: * Move to first used field
                     in next line *
      Downarrow: * Move to same field
                     in next line *
        Page up: * Display next page *
      Page down: * Display next page *
      Last page: * Display last page *
     First page: * Display first page *
    End case
  End do
End:
End Procedure:
```

Figure 2.11 contd. Edit Form.

```
Procedure Select Options
*****************************************************
* This routine enables the user to define whether *
* blank records are to be deleted, whether the    *
* records are to sorted, and if another Egide form*
* is to be subsequently edited.                   *
*****************************************************

Inputs:   * None *
Outputs:  * None *

Variables
  Integer:
    Single:
      Local:
        Terminator

Begin:
  Display Form (Nxtfrm)
  While (* form is incomplete *) Do
    Read All Fields (Field values,
                     Terminator)
  End while
End:
End Procedure:
```

Figure 2.13 Select Options

```
Procedure Delete Records
*************************************************
* This procedure deletes any blank records in the *
* input form file.                                *
*************************************************

Inputs:   Form name
Outputs:  * None *

Variables:
  Char:
    Single:
      Global:Partition:A
            Form name

  Integer:
    Single:
      Local:
            Blank

Begin:
  * Formulate file name form form name *
  Do While (not end of file)
    * Read record from record number N *
    If (record is blank) Then
      Blank:=Blank+1
    Else
      * Write record to record number N-Blank *
    End if
  End do
  If (no terminator record) Then
    * Write terminator record to
      record number N-Blank *
  End if
```

Figure 2.15 Delete Records

```
  * Copy original file to scratch file *
  * Delete original file *
  * Copy scratch file to original file *
  * Delete scratch file *
End:
End procedure:
```

Figure 2.15 contd. Delete Records

```
  * Copy original file to scratch file *
  * Delete original file *
  * Copy scratch file to original file *
  * Delete scratch file *
End:
End procedure:
```

Figure 2.15 contd. Delete Records

CHAPTER 3

THE IMPLEMENTATION OF IFE

3.0 Introduction

The result of the analysis and design of IFE in Chapter 2 is a set of processes described in PDL. This chapter describes the scheduling of these processes by means of a state machine, and the changes to these processes as demanded by the state machine.

3.1 Process Scheduling

A state transition diagram is used to describe the process scheduling. The aproach used here is similar to that used by Mendes(1978), who, together with Walker(1984) and others, has described the methodology and its advantages in sufficient detail for the reader who requires it.

However, the differences between the method used here and that used by Mendes will be explained. Firstly, Mendes uses an unconventional state diagram notation which is not used here. The usual conventions are used here; circles represent states, and arrows indicate allowable transitions between states. In addition each arrow's annotation describes first the condition that causes the state transition, and then the process(es) to be executed before the transition is made. The latter feature

of the state diagram makes it possible to derive the state table directly from the state transition diagram.

Another difference is that only "major" states and processes are included in the state diagram and state table here while Mendes includes all states and processes in a single state table. Section 3.2 contains the explanation of what are considered to be "major" states and the rationale behind including only these states in the main state diagram.

The processes to be scheduled have been described in Table 2.1. The major states related to these processes are listed in Table 3.1, and the conditions governing the transitions between these processes are described in Table 3.2. The

Table 3.1 The Major States in IPE

State	Description
S1	Start IPE
S2	Wait for input from 'Select' form
S3	Wait for input from form specifying unused fields
S4	Wait for input from 'Header' form
S5	Wait for input from Egide form
S6	Wait for input from 'Options' form.
S7	Wait for input from form used to get sort information
S8	End

of the state diagram makes it possible to derive the state table directly from the state transition diagram.

Another difference is that only "major" states and processes are included in the state diagram and state table here while Mendes includes all states and processes in a single state table. Section 3.2 contains the explanation of what are considered to be "major" states and the rationale behind including only these states in the main state diagram.

The processes to be scheduled have been described in Table 2.1. The major states related to these processes are listed in Table 3.1, and the conditions governing the transitions between these processes are described in Table 3.2. The

Table 3.1 The Major States in IFE

State	Description
S1	Start IFE
S2	Wait for input from 'Select' form
S3	Wait for input from form specifying unused fields
S4	Wait for input from 'Header' form
S5	Wait for input from Egide form
S6	Wait for input from 'Options' form.
S7	Wait for input from form used to get sort information
S8	End

relationship between the states, processes and governing conditions is illustrated in the state transition diagram of Figure 3.1. Table 3.3 is the state table derived directly from Figure 3.1.

3.2 Choice of Major States

The factors influencing the choice of the states in Table 3.1 are discussed in this section.

3.2.1 Major and Minor States

It should be noted that only major states are included in Table 3.1, while states considered to be minor are not. Minor states are those which are wholly contained within a major state, ie. there are no direct state transitions between these states and other major states. They are also easily managed and understood without the use of state tables. The use of state diagrams to schedule such states serve only to complicate rather than to clarify a program description. Therefore, processes which are clearly described and managed by the PDL method are exempted from the state diagram.

A case in point is the Egide Form Editing Process described in Figure 3.2. If the condition causing a state transition is such that the next state is the same as the current state, then this state transition is not shown in Figure 3.1, but is accomodated within the Form Edit process as shown in Fi 3.

Table 3.2 Conditions Governing State Transitions

Code	Description
C1	Option to select unused fields entered.
C2	No unused fields, display header.
C3	No unused fields, don't display header.
C4	At least one used field selected, display header.
C5	At least one used field selected, don't display header.
C6	Form terminated normally.
C7	Form terminated abnormally.
C8	Don't delete blank records, sort file
C9	Delete blank records, sort file
C10	Another Egide form is required.
C11	No other Egide form is required, delete blank records.
C12	No other Egide form is required, don't delete blank records.
C13	No other Egide form is required
C14	Another Egide form is required, no options selected.

Table 3.3 State table for IFE

Present State	Condition	Next State	Processes Executed
S1	—	S2	P1
			P2
S2	C1	S3	P3
	C2	S4	P4
	C3	S5	P5
	C7	S8	—
S3	C4	S4	P4
	C5	S5	P5
	C7	S8	—
S4	C6	S5	P5
	C7	S8	—
S5	C6	S6	P6
S6	C14	S2	P2
	C8	S7	P7
	C9	S7	P7
			P8
	C11	S8	P8
	C12	S8	—
	C7	S8	—
S7	C10	S2	P9
			P2
	C13	S8	P9
	C7	S8	—

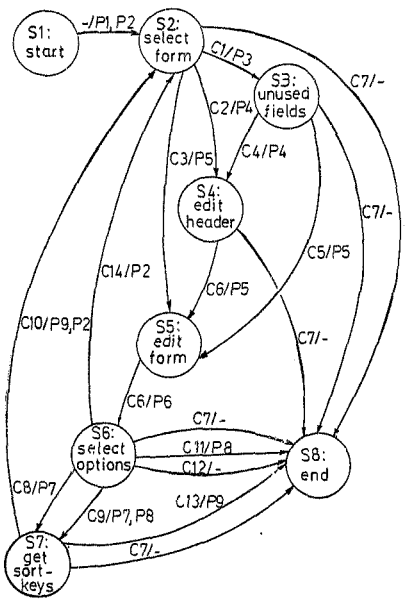


Figure 3.1. State transition diagram for I.F.E.

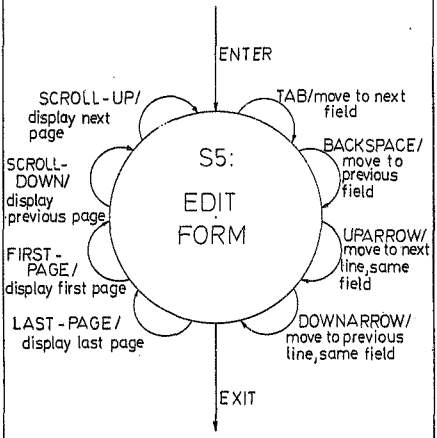


Figure 3.2. The edit form state with minor state transitions included

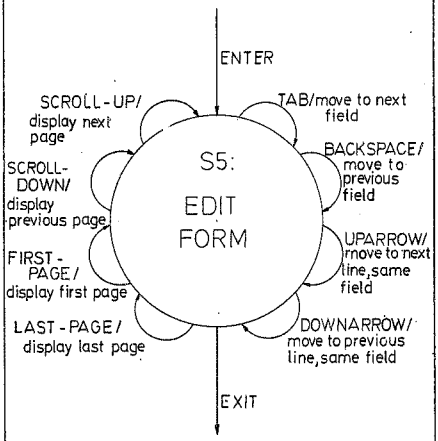


Figure 3.2. The edit form state with minor state transitions included

3.2.2 Modularity Retention

There is a real danger of a major process losing its "identity" by listing its subprocesses carte blanche in the state table. It is difficult for the maintenance programmer to appreciate the modularity easily in this case. However selective inclusion of processes in the state table can help in retaining the processes as obtained from the decomposition phase of the design.

3.2.3 Clarity of State Diagram

In addition the indiscriminate inclusion of every possible state and process regardless of size does not augur well for the clarity and ease of understanding of the state diagram. Rather a proliferation of state diagrams would result, making design, documentation and maintenance more difficult than necessary. The task of wading through numerous state diagrams to get an overview of the control and data flow in a system is both daunting and unlikely to succeed.

3.2.4 Program Response Time

Response time can also be detrimentally affected by including all the processes of a system in a single state table. For the purpose of illustration consider a 100-process system. If a particular state transition requires process 100 to be executed then the Execute Process procedure would be called with an argument equal to 100. In this case 100 checks will have to be made to determine which process to execute. Had there been only 10

major processes in the main state table, however, the number of checks would have been correspondingly less.

3.2.5 Hierarchy of State Tables

Furthermore, the use of state tables as a programming tool in preference to PDL as used by Caine(1975) is advocated by Walker(1984) to avoid long loops and repeated indentations of code in nested loops in large programs. Therefore if a process as produced in the decomposition phase of the design does not present these difficulties it is obvious that the PDL method can still be used in the design of that process. If the process is still too large so that the arguments against the use of the PDL tool only are justified, then the process can have its own state table. Such a method would result in a hierarchy of state tables where necessary, and would avoid excessively long state tables.

3.3 State Table Implementation

A major advantage of state table driven software is that the procedure to implement the state machine is similar for different programs. Therefore it is neccesary to write this procedure once only, and thereafter modify it where necessary for different programs. The format of the state table derived for IFE is similar to that used by Mendes(1984), the main difference being that Mendes calls state transition conditions and processes Lexical Items and Details respectively. The IFE main program and the procedure to execute the processes are

described in Figures 3.3 and 3.4.

3.4 Process Modifications

The behaviour of the processes as described in Chapter 2 will obviously have to be modified such that they adhere to the requirements of the state table driven method of process scheduling. For most processes this involves only the addition of more code at the end of the procedure to set the condition which governs the state transitions.
These modifications are illustrated by the new behaviour *of the Select Form and Display Header* processes described in Figures 3.5 and 3.6, which are derived from Figures 2.5 and 2.7 respectively.

This chapte ...cribed the state machine method of process ...lling as used in IFE, and the associated process modifications required. The next chapter assesses the performance of IFE.

```
Program IFE

Types
  State Table (ST) = Record:
                       Current State (CS) : integer
                       Condition     (CO) : integer
                       Next State    (NS) : integer
                       Process       (PR) : integer
                         array  of size (Maximum
                                         processes)
                     End Record:

    Array
      Local
            State Table:=of size (Maximum items)

Constants:
---------
  Integer
    Single
      Local
            Unconditional execution := 0
            End state := 8
            End of state table := -1
            Processes complete := -1
            Maximum processes := 3
Variables:
---------
  Integer
    Single
      External
            Initial state
            Initial Condition
```

Figure 3.3 IFE Main Program

```
      Local
            State table pointer     (STP)
            Process pointer         (PRP)
            Process                 (PR)
            Current state           (CS)
            Next state              (NS)
            Condition               (CO)
            State variable

      Array
        Local
              State table           (ST)
Data State Table

   Current state  Condition Next state  Process
         1            0          2        1,2,-1
         2            1          3        3,-1
         2            2          4        4,-1
         2            3          5        5,-1
         3            4          4        4,-1
         3            5          5        5,-1
         3            7          8        -1
         4            6          5        5,-1
         4            7          8        -1
         5            6          6        6,-1
         6            8          7        7,-1
         6            9          7        7,8,-1
         6            11         8        8,-1
         6            12         8        -1
         6            7          8        -1
         7            10         2        9,2,-1
         7            13         8        9,-1
         7            7          8        -1

End data
```

Figure 3.3 contd. IFE Main Program

```
Begin :
  State variable := Initial state
  Condition := Initial condition
  STP := 1
  PRP := 1
  While (State variable <> End state) Do
    If (ST.CS:STP = State variable) Then
      If (ST.CO:STP = Condition or
          ST.CO:STP = Unconditional execution) Then
        Next state := ST.NS:STP
        While (ST.PR:PRP:STP <> Processes complete
              and PRP <= Maximum processes) Do
          Call Processes (ST.PR:PRP:STP) Condition
          PRP := PRP + 1
        End While
        If (ST.PR:PRP:STP <> Processes complete)
          Then
          * Signal error: incorrect state table *
        End if
        State variable := Next state
        STP := 1
        PRP := 1
      Else
        STP := STP + 1
        If (ST.CS:STP = End of state table) then
          * Signal error: undefined state *
        Endif
      End if
    End if
  End While
End :
End Program :
```

Figure 3.3 contd. IPE Main Program

```
Procedure Processes
**************************************************
* This procedure calls the required process.     *
**************************************************

Inputs  : Process number
Outputs : Condition

Variables:
----------
  Integer
    Single
      External
            Process number
            Condition

Begin :
  Case Process Number of :
    1 : Call Initialisation
    2 : Call Select form () Condition
    3 : Call Select unused fields () Condition
    4 : Call Edit form header () Condition
    5 : Call Edit Egide form () Condition
    6 : Call Select options () Condition
    7 : Call Get sort information () Condition
    8 : Call Delete blank records () Condition
    9 : Call Sort records () Condition
  Else
    * Signal error: undefined process *
  Endcase
End :
End Procedure :
```

Figure 3.4 Procedure Processes

```
Procedure  Form Selection
*************************************************
* A routine to prompt the user to specify the   *
* Egide form to be edited, whether certain fields *
* are to defined as unused, and whether the Egide *
*   form   header   is   also   to   be   edited.
*
*************************************************

Inputs:      * None *
Outputs:     Form name,Condition

Variables:
  Char:
    Single:
      Global:Partition:A
            Form name

    Array:
      Local:
            Valid form names

  Integer:
    Single:
      Local:
        Condition

Begin:
  Open Form Library (Formlib1)
  Clear Entire Screen and Display Form (Select)
  Get Named Data (Ndata,Valid names)
  * Store Valid names in Valid Form Names *
```

Figure 3.5 Modified form selection process

```
  While (* form is incomplete and selected
                  form name is invalid *) Do
    Return Values for All Fields (Field values)
    * Validate user-selected form name *
  End while

  If (* normal exit from form *) Then
    Case option of:
      Select unused fields: Condition:=C1
      Display form header : Condition:=C2
    Else
      Condition:=C3
    End case:
  Else
    Condition:=C7
  End if
End:
End Procedure:
```

Figure 3.5 contd. Modified form selection process

```
Procedure     Display Form Header
*****************************************************
* A routine to display and edit the Header of an    *
* Egide form.                                       *
*****************************************************

Inputs:  Form name
Outputs: * None *

Constants
  Char:
    Single:
      Local:
        Ndata

Variables:
  Char:
    Single:
      Local:
        Header Data

  Char:
    Single:
      Global:Partition:A
           Form name

Begin:
  Clear Entire Screen and Display Form (Header)
  * Retrieve Header data from input form file *
  Output Values to All Fields (Header data)
  While (* form is incomplete *) Do
    Return Values for All Fields (Header data)
  End while
  * Save Header data in input form file *
```

Figure 3.6 Modified display header process

```
  If (* normal exit from form *) Then
    Condition:=C6
  Else
    Condition:=C7
  End if
End:
End Procedure:
```

Figure 3.6 contd. Modified display header process

```
  If (* normal exit from form *) Then
    Condition:=C6
  Else
    Condition:=C7
  End if
End:
End Procedure:
```

Figure 3.6 *contd.* **Modified display header process**

CHAPTER 4

PERFORMANCE OF IFE

4.0 *Introduction*

The analysis, design and implementation of IFE has been described thus far. This chapter evaluates the performance of IFE in the light of its specification.

4.1 Performance of the IFE Program

During normal use the response time of IFE is good. Normal use entails accessing a form and adding new records wihout using the options to sort or delete records. The actual elapsed time taken for each operation is obviously dependent on the total

TABLE 4.1 IFE Performance Figures.

OPERATION	CPU TIME (seconds)	
	100 records	25000 records
Display header	0,20	0,16
Display first page	0,96	0,92
Page forward	0,89	0,87
Page backward	0,88	0,81
Display last page	1,06	42,73
Delete blank records	1,94	139,31

workload of the VAX 11/750 system. CPU times for the important operations are given in Table 4.1.

Reading or saving a file is quick because only the "window" is read from or written to during these operations. Thus even for very large files (25000 records) the response time is not significantly different from that for very small files (100 records or less).

If the Delete-Records option is selected then the time taken to check for blank records is proportional to the file size as every record has to be read. The Sort option is not included in the comparison above because the maximum file size which can be sorted is dependent on the working area currently allocated by the VAX computer system.

4.2 User Assessment

IFE has been used extensively by six different people, of which one is a typist. All have found the system to be friendly, easy to use, and robust. As DeMarco (1978) advocates, the acceptance test of any system should and must be derived from the specification of that system. If this view is applied, then IFE meets every detail of its specification as given in paragraph 1.5.

However, the severity of the shortcomings experienced by users have been investigated. The complaints thus far are the lack of faster scrolling features and the delays when deleting records from long files.

Both these shortcomings can be traced back to inadequacies in the initial specification, where no response times for any feature of IFE was specified. However the delete records feature of IFE is not frequently used, thus reducing the severity of the associated shortcoming.

The lack of faster scrolling features is admittedly a desirable feature for modification of large files. IFE can be modified to include this feature at the expense of simplicity of use since more function keys will be required.

4.2.1 Modification and Addition of Forms

Occasions have already arisen for new forms to be added and existing forms to be modified to various degrees. These tasks have been easily performed not only by the author but also by a maintenance programmer who needed little guidance other than the description of the form structure and how to implement changes and additions of forms. The definition and addition of a very complex form can normally be completed within two hours. This attests to the ease of maintenance of the system.

CHAPTER 5

CONCLUSIONS

5.0 Introduction

The analysis, design and implementation of IPE using the process resource and state table methods have been described. This chapter discusses the major conclusions drawn from this project.

5.1 Advantages of the process-resource method

The Process-Resource method of analysis was found to have many advantages over the Structured Analysis and design method of DeMarco(1978). DeMarco(1978) advocates system analysis by means of data flow diagrams and structured charts independent of the machine on which the design is to be implemented. This implies that only resources common to all possible implementation machines can be used in the analysis, which is good when the machine is unknown in the analysis stage but not when it is. Often the machine, and sometimes even the language, is known in the analysis stage, as for IPE. In the latter case, many resources are available and can be seen in the analysis stage to be useful. In such a case analysis and design of functions which can be performed by these resources is wasteful and only serves to increase the complexity of the task at hand.

Another shortcoming of Demarco's method is that the only resource that is allowed is a file. Since prudent choice and definition of data structures for a particular system are resources which greatly affect the design of the system, it is important that the design methodology used allows their manifestation. It is probable that DeMarco's method is intended more for the commercial sector where traditionally the only resource used is a file, as compared to engineering and scientific applications where this is obviously not the case.

It was found that the resource description advocated by Walker (1984), viz. that of describing a resource in terms of its behaviour and restricting the interaction with that resource to high level operations on it, is conducive to hiding low level details. The information hiding benefit derived assists in delaying implementation specific details at the high level design stages, which is obviously a major benefit.

5.2 The Importance of the Specification

Most texts on software engineering stress that one of the most important functions of the analysis phase in the design cycle is the production of a correct specification. However, the extenuating circumstance in the case of IFE was the mistaken assumption that a specification received from a superior is correct.

The importance of not assuming that the initial user specification is correct and complete has been noted. In the case of IFE the maximum file size was not expected to exceed a hundred records in the

mind of the designer and the user because of a lack of experience and foresight in the usage of the EGIDE Input Forms. Admittedly, there are only two forms which normally exceed a hundred records. Nevertheless, the design of IFE would definitely have been influenced by these factors had they emerged during the analysis phase.

Fortunately IFE is still able to process these forms, albeit at a much degraded performance level. Thus thorough discussion and refinement of the initial user specification is necessary during the analysis stage. As Demarco(1978) notes, the performance of an efficient system is normally not better than its specification.

5.3 Functional Decomposition

The functional decomposition of a system is influenced significantly by the intuition and the experience of the designer, a conclusion which is supported by the Jackson Method (Yourdon et al, 1975; Lavigne, 1977). However, the experience of this project shows that the use of the state transition diagram helps in alleviating shortcomings which may occur in the functional decomposition. Thus using functional decomposition and state diagrams iteratively should result in a set of major processes which are tightly controlled.

Not all software engineering methodologies are easily applied to all systems to obtain the benefits therof. I think that there is an optimum tradeoff between rigidly adhering to a software engineering methodology and the ease of

understanding of a system. How and where the design of a system should adapt a particular methodology is still the intuitive choice of the designer.

5.4 Modularity Retention

The concept of using state-table-driven software in a manner that retains the inherent modularity of processes was advocated. The main reason for this view is to achieve a main state diagram which summarises the main control and data flows of a system in an easily digestible form. This is an effective overview of a system. In addition this concept improves the response time of a system, and retains the processes resulting from the process decomposition method. As can be seen from the PDL descriptions of the processes before and after the state table implementation, the only difference is the addition of a section in each process to set the state transition condition according to the state table.

The concept of modularity retention is intimately linked to that of top-down design. The well known merits of top-down decomposition (Freeman, 1980) justifies the attempt in this report to retain the various levels of the design in the implementation, and is a strong argument for a hierarchy of states for complex processes.

5.5 State Hiding

The concept of modularity retention is intimately linked with the concept of "state hiding". The relationship of "state hiding" to state diagrams

parallels that of "information hiding" to programs. It seeks to remove from the state diagram any state which is internal to a major process, and which, if added to the state diagram, contributes more to cluttering the system description than clarifying it.

APPENDIX A

The VAX-11 Form Management System (FMS)

Introduction

The FMS enables the user to easily define forms, and modify existing form definitions. Such a form definition enables the high level language programmer to do sophisticated input/output operations to a terminal without becoming engrossed in low level details.

FMS has numerous facilities but only those used by IFE will be discussed.

The Form Definition

The form is defined via the FMS Form Editor (FED). The text of a form is drawn exactly as it would appear on the screen when the form is displayed. The fields in the form must be described by drawing in the picture validation characters in the positions required. Picture validation characters determine what input from the terminal operator is valid for a specific character position in a field. In the event of an error, the input is rejected and an appropriate error message displayed at the bottom of the screen. The five picture validation characters and the input associated with each are :

A - Alphabetic
C - Alphanumeric
N - Signed Numeric
X - Any Character
9 - Numeric

Adjacent fields must be separated by at least one blank or one character of text. All fields read from the screen via FMS are returned as character strings.

Field Attributes

Only two of the many attributes are used. The first is used to associate the Help form with a particular field. For any one form, all fields are associated with a common Help form which provides information on the editing features of IPE and the corresponding keys.

The second feature is called the Supervisor Only attribute, and is used to satisfy the special requirements of the Control-character field. When the program turns on the Supervisor-only mode, the operator is barred access to all fields with the Supervisor-only attribute.

All fields are left-justified and blank-filled by default.

Form Definition Storage and Retrieval

Form definitions are stored in form libraries which can be accessed from programs. Creation and update operations are done via the FMS Form Utility (FUT) and the FMS Form Editor (FED).

Named Data

This section of the FMS Form Description consists of fifteen records, each of which consists of two fields. The first field is for a six-character name associated with the data stored in the second field. Each data field can contain a maximum of sixty characters, and can be used for whatever additional features the programmer desires to implement. IPE uses the Named Data section to store field-specific information as shown in Table A1.

TABLE A1 Format of Named Data for IPE

FLDNO	N,S
VLDT1	/Validation data for field 1/Validation data for field 2/....
.	
.	
.	
VLDTm	/Validation data for field N/

where N: is the number of non-Display-only fields in the form.
S: is the number of lines in the form
m: is less than or equal to 15

The validation data for each field must be in the format:

/P,L,V,Data/

where P: is the starting position of the field in characters counting from the left.
L: is the length of the field in characters.
V: is the type of validation and is :
0 if the field is not validated.
1 if upper and lower bounds test
2 if specific names test
3 if digit-by-digit bounds test.
Data: is the data to be used in the field validation as defined below.

V	Data
0	This field is not required and is followed immediately by the separator '/'.
1	$[a_1,b_1],[a_2,b_2],\ldots[a_k,b_k]$ where: a_i is the lower bound, b_i is the upper bound, $a_i > b_{i-1}$, i=2,3,...,k
2	$Name_1,Name_2,\ldots,Name_n$
3	$[a_1,b_1],[a_2,b_2],\ldots[a_k,b_k]$ where: a_i is the lower bound of the i^{th} digit, b_i is the upper bound of the i^{th} digit.

APPENDIX B

Functional Description of IPE

The keys, their functions and their usage are described in Tables B1 and B2. The layout of the keyboard is the standard QWERTY type for the VT-100. The keypad for the VT-100 terminal is shown in Figure B1.

TABLE B1 Keypad keys, function and usage for IPE.

KEYPAD KEY	FUNCTION	USAGE
PF1	Insert/ Overstrike	Switches from the INSERT mode to the OVERSTRIKE mode, or vice-versa.
PF2	Help	Displays the HELP form for the current form.
.	Exit	Terminates all entries in the form.
0	Page forward	Displays next page.
1	Page backward	Displays previous page.
2	Last page	Displays last page
3	First page	Displays first page

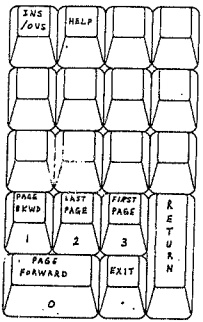


Figure B1 Keypad layout for VT100 terminal

TABLE B2 Keyboard keys ,functions and their usage for IFE.

KEYBOARD KEY	FUNCTION	USAGE
Left arrow	Cursor left	Moves the cursor to the preceding character position within the field.
Right arrow	Cursor right	Moves the cursor to the next character position within the field.
Delete	Erase character	In the INSERT mode erases the character to the left of the cursor and closes the space. In the OVERSTRIKE mode, moves the the cursor to the preceding character position within the field but erases it only when the character is the last one in a left-justified field.
Linefeed	Erase field	Erases the entire field

TABLE B2 contd. Keyboard keys ,functions and their usage for IFE.

KEYBOARD KEY	FUNCTION	USAGE
Ctrl W	Repaint screen	Repaint the screen with the current form, field values, and cursor location.
Most keyboard keys	Insertion	The keys for the printing characters on the keyboard insert their characters.
TAB	Next field	Moves the cursor to the initial position of the next used field.
BACKSPACE	Previous field	Moves the cursor to the initial position of the previous used field.
RETURN	New line	Moves the cursor to the first used field in the next line.
Down arrow	Scroll down one line	The cursor moves down one line and that line becomes

TABLE B2 contd. Keyboard keys ,functions and their usage for IFE.

KEYBOARD KEY	FUNCTION	USAGE
		the current line or the form is scrolled up one line if the if the cursor was at the last line on the screen.
Up arrow	Scroll up one line	The cursor moves up one line and that line becomes the current line or the form is scrolled down one line if the cursor was at the first line on the screen.

APPENDIX C

The IFE User Interface

The Login Procedure

To log in 'IFE' is entered in response to both the VAX/VMS Username and Password prompts. The Login command procedure is then used to prompt the user for the name of the site where the SA128 exchange is located. This name is used to determine the name of the subdirectory in which the input form files reside. The user is then asked whether the site is a new one or not. *If it is, a new subdirectory is* created by the command procedure and the name of the subdirectory assigned to a logical name used to communicate the name to the IFE program. If the site is an old one only the latter operation is performed.

Form Selection

After logging in successfully, the user is prompted with the panel shown in Figure C1.

The user is restricted to making entries in the enclosed areas, which appear in reverse video. The first entry is the form name, which must be one of the names contained in the list under Menu. The Display Header option allows the user to specify whether or not the header area of the file should be displayed, while the *Unused Fields* option enables the user to indicate that certain fields in the input form will not be used in this editing session.

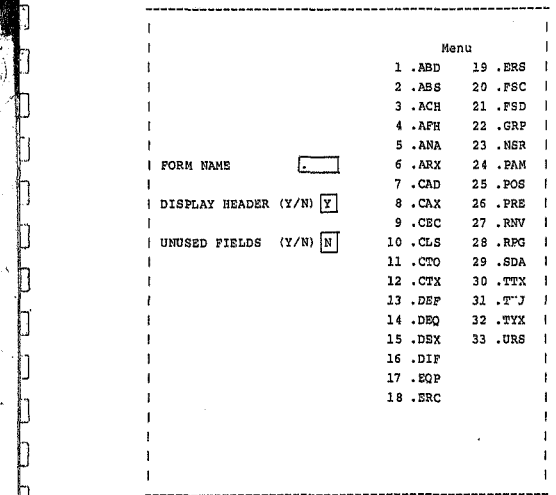


Figure C1 Form Selection Panel

The default values for the options are shown in Figure C1.
The 'TAB' key is used to jump to the next field, 'BACKSPACE' to jump to the previous field, and 'RETURN' to exit this panel.

The Header Area

The file header is displayed using the panel shown in Figure C2 if the Display Header option is active.

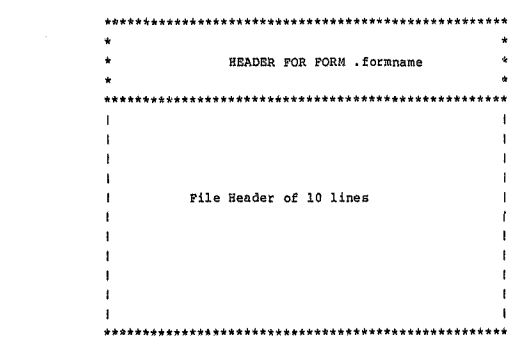


Figure C2 File Header Panel

The data area consists of 10 rows, each of 79 characters. Editing is the same as for the Form Selection panel with each row considered to be a field.

Selection of Unused Fields

If the Unused Fields option is active, then the user is prompted to select the unused fields with a typical panel as shown in Figure C3.

```
*****************************************************
*                                                   *
* Field   Field    Field   Field   Field    Field   *
* name1   name2    name3   name4   name5    name6   *
*****************************************************
|  N       N        N       N       N        N      |
*****************************************************
```

Type Y if the field is unused.

Figure C3 Panel to select unused fields.

The text area of this panel is the same as that of the form used to input the data. Below each field name is a single character in reverse video used to select the field as used or unused. Selection of the field as unused will result in the user being denied access to that field during editing of the input form. Editing of this panel is the same as for the Form Selection panel.

Input Form Panel

A typical Input Form Panel is shown in Figure C4. In order to minimise user confusion the input form panels were made to resemble as closely as possible those on paper from which the data is copied. Data fields are entered directly below the corresponding field names in the text header, and adjacent fields are clearly separated by blanks.

The maximum screen size is 24 rows by 132 column characters including the message area. However, not all 132 columns are used for all forms since the

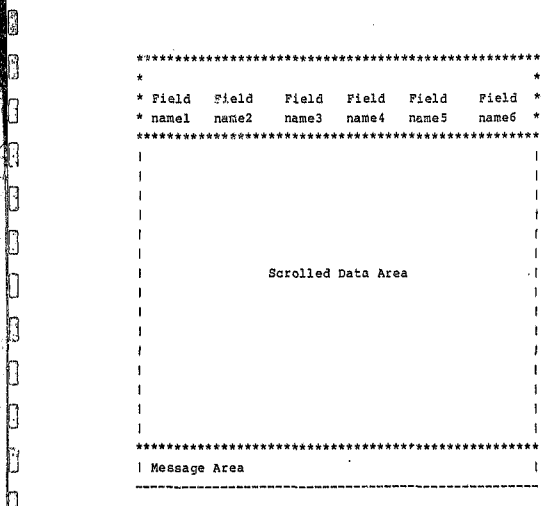


Figure C4 Screen Layout for Input Form Panel

width of the form is determined by the number of fields and field separators. At present the width of all input forms is the range 100 to 132 columns.

The length of the screen is fixed at 24 rows for all forms, and is made up of the lengths of the text header, the data area, and the message area. Since the length of the text header is determined by that of the field names, it differs for each form, varying from 3 to 7 rows. Since the length of the

message area is always 1, the length of the length of the data area in each form also varies from 15 to 19 rows. All lines in the data area are 80 characters long and are scrolled.

The message area is used to display informational and warning messages such as 'Illegal value', 'Top of buffer', 'End of buffer'.

Option Selection

On termination of the editing session the user is prompted with the panel shown in Figure C5.

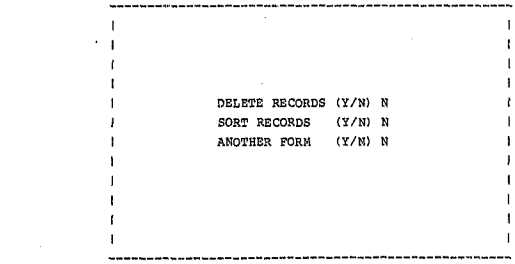


Figure C5 Option Selection Panel

Except for the options, the description of this panel is similar to the *Form Selection panel*. The Delete Records option is used if there are any blank records in the input form. If the records are to be

sorted then the Sort option is selected. The Another Form option enables the user to stay in *IFE* to edit another form, or to exit from IFE.

Sort-key Panel

If the Sort option is active, the user is prompted with a panel similar to that of Figure C6.

```
*******************************************************
*                                                     *
* Field  Field   Field  Field  Field   Field          *
* name1  name2   name3  name4  name5   name6          *
*******************************************************
!   0      0       0      0      0   SORT-KEY (0-9) !
!                                                     !
!   A      A       A      A      A   SORT ORDER(A/D)!
*******************************************************
```

Figure C6 Sort-key panel

The description of this panel is similar to that of the Unused Fields panel, the main difference being that for each field that is to be sorted 2 fields must be filled in, viz. the sort-key number and the order of the sort for that sort-key. The sort-keys determine the order of the sort, eg. a sort-key of 1 indicates the field is the primary sort-key while a sort-key of 2 indicates that the field is the secondary sort-key. The *sort-order specifies* whether the field should be sorted in ascending or descending order. Editing is as for the Form *Selection panel*.

If the user has indicated that he wants to edit

another form, then the *Form Selection* panel is now displayed. Otherwise IFE terminates.

BIBLIOGRAPHY

Caine, H.S., and Gordon, E.K (1975), PDL-A tool for software design, *Proceedings*, National Computer *Conference*, 1975.

De Marco, T. (1978) Structured Analysis and System Specification, 3rd ed. New York: Prentice Hall.

D'Angelo, P. (1983) State-table-driven code simplifies software development, EDN, Sept.

D'Angelo, P. (1985) Structured analysis aids in microcomputer system design, EDN, Mar.

DEC (1980), VAX-11 *FMS Software* Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982a) VAX-11 Fortran Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982b) VAX-11 Fortran User's *Guide*, Massachusetts: Digital Equipment Corporation.

DEC (1982c) VAX-11 Run-Time Library Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982d) VAX-11 Run-Time Library User's Guide, Massachusetts: Digital Equipment Corporation.

DEC (1982e) VAX-11 Record Management Services Tuning Guide, Massachusetts: Digital Equipment Corporation.

BIBLIOGRAPHY

Caine, H.S., and Gordon, E.K (1975), PDL-A tool for software design, *Proceedings*, National Computer *Conference*, 1973.

De Marco, T. (1978) Structured Analysis and System Specification, 3rd ed. New York: Prentice Hall.

D'Angelo, P. (1983) State-table-driven code simplifies software development, EDN, Sept.

D'Angelo, P. (1985) Structured analysis aids in microcomputer system design, EDN, Mar.

DEC (1980), VAX-11 *FMS Software* Reference Manual, *Massachusetts*: Digital Equipment Corporation.

DEC (1982a) VAX-11 Fortran Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982b) VAX-11 Fortran *User's Guide*, *Massachusetts*: Digital Equipment Corporation.

DEC (1982c) VAX-11 Run-Time Library Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982d) VAX-11 Run-Time Library User's Guide, *Massachusetts*: Digital Equipment Corporation.

DEC (1982e) VAX-11 Record Management Services Tuning Guide, Massachusetts: Digital Equipment Corporation.

DEC (1982f) VAX-11 System Services Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982g) VAX-11 SORT/MERGE Reference Manual, Massachusetts: Digital Equipment Corporation.

DEC (1982h) VAX-11 SORT/MERGE User's Guide, Massachusetts: Digital Equipment Corporation.

Freeman, P., and Wasserman, W.I. ed., (1980) Tutorial on Software Design Techniques, 3rd ed. California: IEEE Computer Society.

Mendes, J.F. (1984) The Design and Implementation of the Man-Machine Interface for a Supervisory Computer System, MSc Thesis, University of the Witwatersrand, Johannesburg.

Parnas, D.L (1972) On the criteria to be used in decomposing systems into modules, Communications of the ACM, Dec.

TelTech (1985a) E10B Digital Time Division Switching System - General Description, Boksburg: Telecommunication Technologies (Pty) Ltd.

TelTech (1985b) Egida Input Forms Description, Boksburg: Telecommunication Technologies (Pty) Ltd.

Walker, A.J (1983) Structured Information Processing Design, University of the Witwatersrand, Johannesburg.