PC BASED STORAGE AND PROCESSING OF ELECTROCARDIOGRAM TRACINGS RECORDED WITH A HP4745A PAGEWRITER II CARDIOGRAPH

JG WASSERMAN

PC BASED STORAGE AND PROCESSING OF ELECTROCARDIOGRAM TRACINGS RECORDED WITH A HP4745A PAGEWRITER II CARDIOGRAPH

JOHAN GEORGE WASSERMAN

Dissertation submitted in fulfilment of the requirements for the Degree

MAGISTER TECHNOLOGIAE:

INFORMATION TECHNOLOGY

in the

Faculty of Management
Department of Information Technology

at the

Technikon Free State

Supervisor: Mr DJ Kotzé, B.Sc. (Pharm), M.Sc. (IT), HBA
Co-supervisor: Dr P Jordaan, MB.Ch B., ECFMG (USA), M Med Internal Medicine

BLOEMFONTEIN
January 1998

DECLARATION OF INDEPENDENT WORK

I, JOHAN GEORGE WASSERMAN, do hereby declare that this research project submitted for the degree MAGISTER TECHNOLOGIAE: INFORMATION TECHNOLOGY, is my own independent work that has not been submitted before to any institution by me or anyone else as part of any qualification.

[illegible]
Signature of student

16 March 1998
Date

To my parents, for their faith, love and support. Thank you.

Without the following persons this research would not have been possible:

- Mr Dana Kotzé and Dr Pierre Jordaan, my supervisors, for their dedicated support, interest and guidance.
- Dr Josef Jacobs, cardiologist and friend, for his help to get this research started as well as for constant interest and support.
- Steve Weber and Craig Hamer (Hewlett Packard's McMinnville Division of Diagnostic Cardiology) for actually arranging a copy of Hewlett Packard's digital transmission protocol for me.
- Robert Vivrette for answering some questions about graphics programming and sending me information *via* e-mail.
- Tienie van Schalkwyk for directing me towards Hewlett Packard's McMinnville Division, as well as providing me with quotations and brochures.
- Dr AP Boezaart and Prof George Murray for their comments during the initial stages of this research.
- Prof Theo McDonald and Prof Charles Herbst for their valuable input during the initial stages of this research.
- Leona le Roux and Rika Pretorius, for helping with overseas telephone calls, making arrangements with courier companies and preparing numerous photocopies of articles.
- Piet van der Merwe for his valuable comments.
- Huibre Lombaard, Esté Louw and Tanya Rood for their friendly and efficient help to locate articles and books.
- Dr Pieter van der Wal for locating difficult-to-find articles in a university library in Amsterdam.
- Johan Pienaar for lending me his soldering iron and workshop and also for carrying PCs to and from the Department of Cardiology.
- Alex Hundt, for listening to ideas and also encouragement.
- Dr Norma van Niekerk for helping with the proof-reading of Chapter 3, lending me some of her medical text books as well as constant interest and support.
- Dr Sybrand Pretorius for lending me some of his medical text books.
- The ECG Ladies at the Department of Cardiology, Universitas Hospital, for testing and using software for digital capturing of ECG data.
- Neliëtte de la Rey for proof-reading of the dissertation. Thanks for a mammoth task. You have a very sharp eye and you are extremely thorough.
- Dr Linda Potgieter and Neliëtte de la Rey for their input on the statistical analysis of the data as described in Chapter 6.
- Cornea Venter for helping with some initial statistical analysis of the data in SAS.
- Jacques Venter for listening and answering a lot of questions about Delphi.
- Bernard van Niekerk for lending a hand with the graphics work.

- My friends Johan, Stoffel, Lukas, Bobby and Victor for bearing with me whilst I was researching and writing this dissertation.
- The Department of Cardiology, University of the Orange Free State, Universitas Hospital for the use of their computer network and electrocardiograph equipment whilst developing and testing the *Hearts 32* application.
- The Department of Pharmacology and the FARMOVS Research Centre for Clinical Pharmacology and Drug Development, University of the Orange Free State, for the use of their computer network, laser printers and time off to see my supervisors.

Summary

Currently the Department of Cardiology, Universitas Hospital, keeps paper copies of ECGs filed in large filing cabinets. Access to these files is tedious during office hours, and impossible after hours, when the filing room is locked and no filing personnel are available.

Commercially available systems for computerised storage of ECG data are available from a number of vendors. Some drawbacks of these systems include:

- Extremely expensive.
- Only a portion of the functions offered by these systems are really needed at the Department of Cardiology, Universitas Hospital. These systems are thus not economically justifiable by the Department of Cardiology, Universitas Hospital.
- Some require new/different ECG machines to be used.
- Some require an expensive computer system to be installed.
- Additional space is needed for additional equipment.
- Staff needs to be extensively trained to use the new equipment.

This dissertation describes the development of a dynamic link library (DLL) which is used to acquire and decode data from a Hewlet Packard HP4745A Cardiograph II PageWriter electrocardiograph. Furthermore, the database application using the HP4745A DLL can also be expanded to accept data from other ECG machines. The acquisition and decoding DLL must be developed to produce a decoded data file conforming to the format described in this dissertation.

By storing these decoded data in a database such as *Hearts 32*, the data can be reprocessed (drawing of ECG traces on screen or on printer). Selected leads from different ECGs can also be plotted on the same screen. Fast access to previous ECGs will help the cardiologists at the Universitas Hospital in Bloemfontein to improve patient care. The cardiac patients of the Free State community as well as the staff at the Department of Cardiology, Universitas Hospital, Bloemfontein can benefit from the results of this research.

Opsomming

Huidiglik berg die Departement Kardiologie, Universitas Hospitaal, papierkopieë van EKGs in groot liasseerkabinette. Toegang na die lêers in hierdie kabinette is binne kantoorure moeilik, en na ure feitlik onmoontlik wanneer die liasseerkantoor gesluit en liasseerpersoneel weg is.

Daar is 'n verskeidenheid kommersieel-beskikbare stelsels vir die rekenaarmatige berging van EKG data op die mark. Probleme met die gebruik van hierdie stelsels sluit in:

- Uiters duur.
- Slegs 'n klein gedeelte van die funksionaliteit wat deur hierdie stelsels aangebied word sal regtig deur die Departement Kardiologie, Universitas Hospitaal gebruik word. Die aankoop van so 'n stelsel kan dus nie ekonomies geregverdig word deur die Departement Kardiologie nie.
- Sommige van hierdie stelsels vereis dat nuwe/ander EKG masjiene gebruik moet word.
- Sommige van hierdie stelsels vereis dat duur rekenaartoerusting installeer moet word.
- Addisionele ruimte word benodig vir addisionele toerusting.
- Werkers moet intensief opgelei word om die nuwe toerusting te kan gebruik.

Hierdie verhandeling beskryf die ontwikkeling van 'n "Dynamic Link Library (DLL)" wat gebruik kan word om data van 'n Hewlett Packard HP4745A PageWriter II Cardiograph EKG masjien te ontvang en dekodeer. Verder word hierdie data in 'n databasis gestoor, vanwaar dit later opgeroep en grafies vertoon kan word. Interaksie met EKG maak dit moontlik om sekere metings op die rekenaarskerm te doen. Die databasis toepassing (*Hearts 32*) wat hierdie "DLL" gebruik kan verder uitgebrei word om data van ander EKG masjiene te aanvaar. Die uitruil en dekodering wat deur die "DLL" gedoen word, moet voldoen aan die formaat wat in hierdie verhandeling beskryf is.

Verdere verwerking van die EKG data word moontlik gemaak deurdat hierdie gedekodeerde data in 'n databasis soos *Hearts 32* gestoor word. Dit sluit in die stip van EKG grafieke op 'n rekenaarskerm of drukker. Geselekteerde afleidings van verskeie EKGs kan ook vertoon word sodat mens dit langs mekaar kan sien. Vinnige toegang na vorige EKGs sal die kardioloë van die Universitas Hospitaal in Bloemfontein help om pasiëntsorg te verbeter. Die kardiologie-pasiënte van die Vrystaatse gemeenskap, sowel as die personeel van die Kardiologie Departement, Universitas Hospitaal, Bloemfontein kan baat vind by die resultate van hierdie navorsing.

Table of contents

List of tables

List of figures

List of code snippets

Chapter 1

Introduction

The Department of Cardiology, University of the Free State, Universitas Hospital in Bloemfontein treats patients with cardiac problems. Due to the increase in the incidence of cardiac disease, the Department of Cardiology has experienced tremendous growth over the past few years. Roughly 13,000 patients are seen *per annum*.

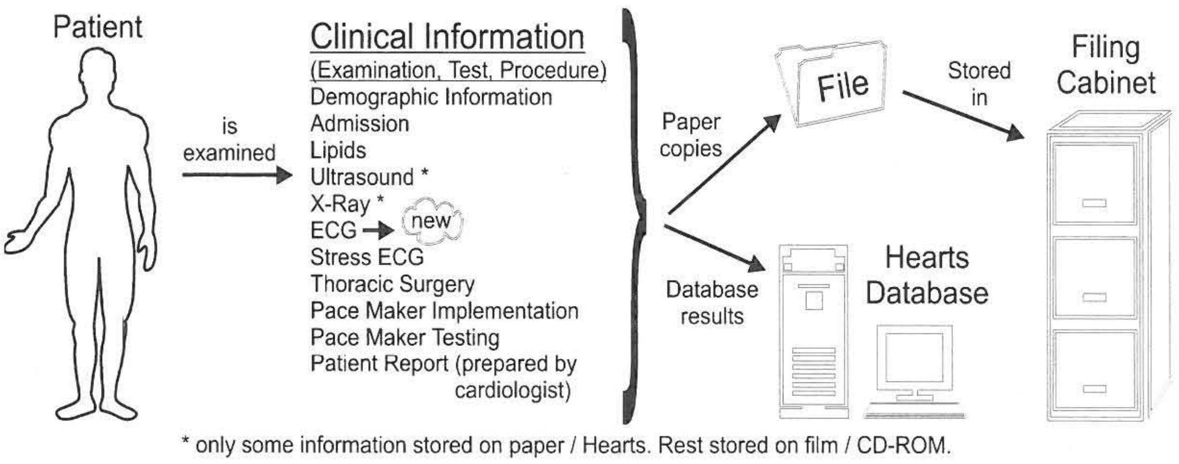


Figure 1-1: Data collected from the patient and stored on file

As can be seen from Figure 1-1, data are collected from the patient as a result of examinations, tests and/or procedures performed. These results were traditionally recorded on paper.[1]

In order to make information useful for the cardiologists, it has to be managed. During November 1993, the Department of Cardiology actively started to work towards computerised storage of patient data. The application developed was called the *Hearts* database.[2]

[1] Information such as the patient report (written by the cardiologist) was prepared using a word processor. These word processor files were not tightly integrated with the rest of the patient information: a printed copy of the report was placed on file. Some of the results are stored on other media, such as X-Ray film (X-Ray examination), magnetic tape (Holter ECG), photographic film and lately CD-ROM (Coronary Angiography).

[2] *Hearts* was developed in Clipper 5.2 and is a MS-DOS® based application. It fully supports multi-user access on a Novell® network.

Goals for *Hearts* included:

- Better organisation of information.
- Fast and easy retrieval of information.
- Increased usefulness of information (for example, to easily create periodic statistics).
- Increased security (regular backups performed, only authorised personnel have access to database).

During the development of the first version of *Hearts* it became clear that it would not be possible to include all data in a digital form in the *Hearts* database (Figure 2-1 on page 10). It was also apparent that the capabilities of the *Hearts* database would grow (and change) with the needs of the Department of Cardiology.

The electrocardiogram is a report of one of the examinations routinely performed. Roughly 20,000 electrocardiogram reports are produced *per annum*. Up to the time of writing this thesis the *Hearts* database did not have the capability to capture and store electrocardiogram data (Figure 2-2 on page 11). Since electrocardiograms are routinely produced (thus largely contributing to the clinical information gathered from a patient) it makes good sense to have these data digitally available.

1. Background

The Department of Cardiology at the Universitas Hospital in Bloemfontein currently (December 1997) has six HP4745A PageWriter II Cardiographs in operation. (A total of 10 electrocardiographs are being used; three are HP4700A Cardiographs and one is a Marquette MAC VU electrocardiograph.[3]) Each of the HP4745A Cardiographs is equipped with a RS-232 communications port. These ports have, however, up to the present, not been used at all.

[3] The HP4700A does not inherently have the capability to transmit ECG data. According to the Hewlett-Packard Company (Hewlett-Packard Company, 1983 : 3) a specialised ECG transmitter module is needed. Such transmitter modules have not been purchased for the three HP4700A ECG machines mentioned. Since this study is aimed at the HP4745A Cardiograph II PageWriter electrocardiograph, any other ECG machines such as the HP4700A and Marquette MAC VU are excluded from this research.

The reasons for this are:

- The cost of a centralised management software package offered by the Hewlett-Packard Company is too excessive (≈R500,000.00) to be economically viable.

- Additional expenses would have to be incurred to house such expensive equipment. New cabling would have to be installed in the hospital. Additional staff would be needed to operate the system. Expensive training would be needed to enable the staff to handle the system efficiently.

The MAC VU is used in the Coronary Intensive Care Unit to monitor acute changes in patient ECGs.[4] ECGs recorded with the MAC VU in the Intensive Care Unit (ICU) do not reflect long term changes (due to the condition of the patients in the ICU). It is planned to place one of the six HP4745A Cardiographs in the ICU to allow the recording of the patient's ECG at admission to the ICU. The HP4700A electrocardiographs are extremely old and are currently being scrapped from normal day to day use at the Department of Cardiology, Universitas Hospital. These machines will be used in other wards where *ad hoc* ECGs are recorded (not by the Department of Cardiology, Universitas Hospital).

The envisaged system (that is, a version of the *Hearts* application, capable of digital capturing and storing of electrocardiogram data), will be economically justifiable, since Hewlett Packard made the *HP Diagnostic Cardiology Digital Transmission Protocol* (see Appendix A) available under an agreement which does not permit the use of the protocol in a commercial product. This means that the data acquisition module developed for the Department of Cardiology (as a result of this research) will be free of charge.

Staff will need to be trained in using the new *Hearts* software. The learning curve, however, is expected to be less steep than with a commercial product, since the staff

[4] The MAC VU can be used to continuously record rhythm strips in real time for as long as needed, since the machine uses continuous paper. The HP4745A does not have this functionality; paper needs to be loaded manually.

already have experience with the existing *Hearts* software. Since the development of *Hearts* was a team effort, staff members also feel personally involved with the system.

The Department of Cardiology currently utilises a file server running Novell® Netware® 4.1 (50 user license). The *Hearts* database resides on this Novell® Netware® 4.1 file server.

The need for digital storage of documents (ECG reports) was greatly increased by three facts:

1. The Department of Cardiology consults approximately 13,000 patients *per annum*. They produce roughly 20,000 electrocardiogram reports per year. Access to a paper-based file system is highly inefficient, due to misfiling that leads to lost documents. Since cardiologists have to request the retrieval of a document in advance (and rely on a filing clerk for the retrieval), access to these patient files are too tedious to be practical. Another danger is that files can be lost or misplaced once they have been retrieved from the filing system. There is further no guarantee that some documents will not be accidentally removed from a patient file!

2. The filing personnel only work normal office hours. It is impossible for a cardiologist to retrieve a patient's file after hours or during an emergency.

3. Storage space for the large number of filing cabinets is no longer available. The weight of the existing cabinets poses a threat to the safety of the building.

Since the inception of the *Hearts* database system, work flow has improved tremendously and documents are retrieved with greater ease. The fact that electrocardiogram information could, up to now, not be digitally stored shows a clear place for improvement in the current system.

2. The rationale behind a new system

Excellent systems for digital storage and manipulation of electrocardiogram data are commercially available (as discussed in Chapter 2). The biggest problem with these systems is their economic viability. Although the Hewlett Packard and Marquette Electrocardiogram Management Systems are the Rolls Royce of this type of system, the purchase price of such a system is simply too high for the Department of Cardiology, Universitas Hospital, Bloemfontein.

The Cardio Perfect system is compact and very easy to use and should ideally be within the financial reach of every general practitioner. However, due to the relatively high price involved, one cannot dispose of all existing electrocardiographs to have them replaced by the Cardio Perfect equipment. Although some of the existing Hewlett Packard electrocardiographs in use are quite old (about 10 years) the machines still function well and cannot be discarded at will.[5]

2.1 Hypothesis

The Department of Cardiology has a need for digital storage of electrocardiogram tracings for easy retrieval and duplication. Having the electrocardiogram tracings digitally available will also facilitate the superimposing (and easy comparison) of selected leads for a specific electrocardiogram. Such recognition of trends will improve patient care. (Some patients have cardiac ECG abnormalities due to previous cardiac disease. It is of utmost importance that the cardiologist is aware of the fact that these abnormalities are in actual fact “normal” for the specific patient. Having the ECG data available in a central repository will provide a means to access these historic ECG data in a timely manner.)

In the light of this, it makes sense to capture electrocardiogram tracings and store them in the existing *Hearts* database used by the Department of Cardiology. If an

[5] It is important to point out at this stage that the HP4745A Cardiograph II PageWriter electrocardiograph does not support dates past 1999 (only the year digits are used, millennium and century digits are omitted, resulting in years with only two digits).

application could be developed (at low or no cost) to suit the needs of the Department, a large amount of money could be saved.

3. Problem definition

This study forms part of the work currently being undertaken at the Department of Cardiology, Universitas Hospital in Bloemfontein to computerise their patient records. A new version of the *Hearts* database, which will be a 32-bit Windows® application, is planned.[6]

The addition of the electrocardiogram data to this database would be an additional advantage as a large number of patients is seen daily by the Department of Cardiology. The digital storage of as much patient information as possible would lead to less dependence on clumsy paper-based filing systems and would allow access from any personal computer connected to the local area network.

This study will lead to the development of the following applications:

- A data acquisition and decoding module for the HP4745A PageWriter II Cardiograph.
- A method for database storage and retrieval of these digitally captured electrocardiogram tracings.
- A set of specifications to allow other developers, creating data acquisition and decoding modules for specific electrocardiograph equipment, to create data sets compatible for inclusion in the *Hearts 32* database.
- A graphic browser used for interactive examination of stored electrocardiograms. Interactive tools will include callipers for quick and easy determination of voltation

6 This version of *Hearts* will be referred to as *Hearts 32* for the remainder of this thesis.

and lapsed time, as well as a zoom tool for more detailed insight into selected tracings. The browser will allow:

- Simultaneous views of different electrocardiograms *via* the implementation of a Multiple Document Interface (MDI).[7]

- Superimposing of selected leads of an electrocardiogram, on screen, allowing the cardiologist to quickly and easily identify trends when examining selected leads of an electrocardiogram. Monitoring of electrocardiogram changes over time, as therapy is adjusted, is an important aspect in improving patient care.

- Printing of the stored electrocardiogram tracing.

Specialised modules such as the data acquisition and graphic modules will be implemented as Dynamic Link Libraries (DLLs), thus protecting the host application (*Hearts 32*) from the complexities of data acquisition and decoding.[8] This approach will ensure that *Hearts 32* is open ended; by the addition of a (specialised) data acquisition module (delivering a data set conforming to the aforementioned specifications), data from other ECG machines can be stored in the *Hearts 32* database.

The successful implementation of the proposed *Hearts 32* database will allow immediate access to previously recorded electrocardiograms. In an emergency, this is very important, as changes in the electrocardiogram tracing have important therapeutic implications for the patient.

This study will not necessarily produce a brand new product in terms of digital capturing, storage and manipulation of the electrocardiogram, but it should provide an economic and cost effective solution to some of the existing information technology

7 Examples of such systems include Microsoft® WinWord and Microsoft® Excel, where the user is allowed to have more than one word processor document or spreadsheet open simultaneously, each in a different window, and then has the ability to switch between different windows.

8 The data acquisition and decoding process differs between manufacturers and models of ECG machines, and are proprietary in nature.

problems at the Department of Cardiology, Universitas Hospital in Bloemfontein, thus improving patient care to the benefit of the Free State community.

4. The layout of this thesis

Chapter 2 defines the problem in more detail. A solution to the problem is then outlined. Chapter 3 formally introduces the electrocardiogram. A basic understanding of the anatomy of the heart as well as the different components of the electrocardiogram are discussed. The data acquisition process will be discussed in Chapter 4, while some basic database concepts will be discussed in Chapter 5. The reasons for choosing a specific storage method will be discussed in Chapter 6. A high level discussion of the software developed for graphic reconstruction of an ECG can be found in Chapter 7. Chapter 8 constitutes a technical discussion of the program code developed for data acquisition and graphic reconstruction of an ECG. The results of this research conclude in Chapter 9.

Appendix A serves as a technical reference where the portion of the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol* relevant to this study, is discussed. Appendix B is aimed at developers who need to create an ECG data file in a *Hearts 32* compatible format, since the ECG data storage format used in *Hearts 32* is described in detail here. A list of terms and abbreviations used throughout this thesis can be found in Appendix C.

Chapter 2

Problem Definition

1. Introduction

The Department of Cardiology, Universitas Hospital, Bloemfontein had a need for digital storage of ECG data (as outlined in Chapter 1). The current *Hearts* database did not support this capability.[1]

Hearts had, in fact, a number of drawbacks that had to be addressed:

- The user interface was still character based (Figure 2-1), and did not keep up with more modern user interface options such as the graphical user interface (GUI) found in Windows® (Figure 2-2).
- The language of the user interface was Afrikaans (Figure 2-1). Since not all of the doctors at the Department of Cardiology, Universitas Hospital, Bloemfontein can read Afrikaans, this is a serious problem.
- *Hearts* could not store the digital ECG data (Figure 2-1).
- Even if the digital ECG data could be stored, *Hearts* could not display these data graphically (Figure 2-1).
- A weak connection between MS Word 5.0 for DOS® and *Hearts* existed. Basically, documents were named with the computer number used to identify the patient in *Hearts*. A myriad of small document files cluttered the hard disk of the file server.

In order to solve the problems and satisfy the needs as identified above, a new version of *Hearts* needed to be developed. This version of *Hearts* would:

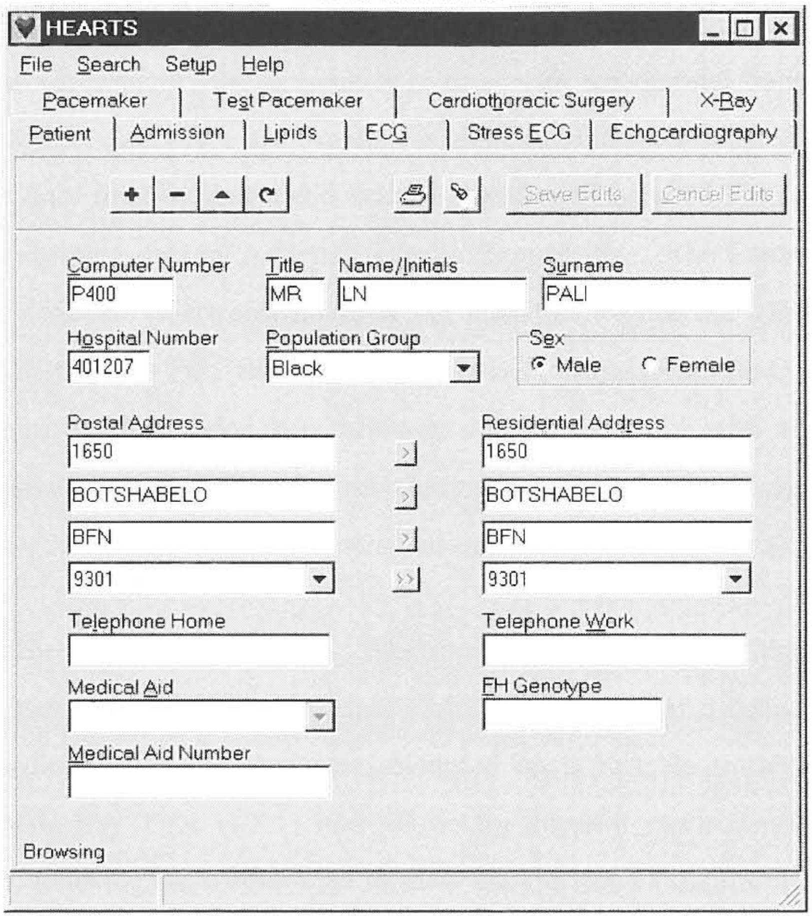
[1] This is not 100% true. An augmented version of *Hearts* was developed (as part of this research) to allow digital capture and storage of ECG data in DOS® files on the Novell® Netware® 4.1 file server. The main reasons for this step was to start the acquisition of ECG files, and also to have data available for statistical analysis. The result of such analysis would help to determine the optimal storage format for the data sets in the database.

- Have a graphical user interface (GUI) (Figure 2-2).
- Interact with users in English (Figure 2-2).
- Be capable of storing binary data streams of an arbitrary length. (Used for storing ECG data as well as word processor documents.)
- Allow a connection to MS Word for Windows® *via* Object Linking and Embedding (OLE).

```
+--------------------------------------------------------------------------------+
¦ Hearts                         Datum:  Son 28 Des 1997         Tyd: 23:34:17 ¦
¦ Weer. 2.01    Verander pasiënt/opname/prosedure detail.  <Esc> om te stop.   ¦
¦-------------------------------------------------------------------------------¦
¦                                                                               ¦
¦ Rekenaar Nr    [P400    ]              Hospitaal Nr          [401207]         ¦
¦ Ras     [S]    (Blank/Swart/Kleurling/Indier/Chinees/Taiwanees/Ander)         ¦
¦ Van     [PALI                  ]      Naam         [LN               ]        ¦
¦ Titel   [MR    ]                      Geslag       [M]  (M/V)                 ¦
¦ Familiële Hipercholesterolomie Status [         ]                             ¦
¦         Woonadres                             Posadres                        ¦
¦-------------------------------------------------------------------------------¦
¦ Reël 1   [1650                   ]    Reël 1   [1650                    ]     ¦
¦      2   [BOTSHABELO             ]         2   [BOTSHABELO  +----------------+ ¦
¦      3   [BFN                    ]         3   [BFN         ¦A. Lipiede      ¦ ¦
¦ Poskode                    [9301]*    Poskode               ¦B. Opnames      ¦ ¦
¦                                                             ¦C. Pasaangeërs  ¦ ¦
¦ Telefoon Tuis     [              ]    Telefoon Werk     [   ¦D. Sonars       ¦ ¦
¦                                                             ¦E. Toets Pasaan¦ ¦
¦ Mediese fonds              [      ]*  M/F Nr   [            ¦F. Toraks Chir  ¦ ¦
¦                                                             ¦G. Trapmeul EKG¦ ¦
¦                                                             ¦H. X-Strale     ¦ ¦
¦                                                             ¦X. Klaar        ¦ ¦
¦                                                             +----------------+ ¦
+---Caps-Num-------------------------------------------¦ Druk F1 vir Hulp ¦----+
```

Figure 2-1: The old character based user interface of Hearts

In all fairness, it must be mentioned at this point that the character based interface of *Hearts* is not quite as appalling in real life as it is portrayed in Figure 2-1 above. The character based interface allows colour display, as well as inverse and blinking display. Line and box drawing capabilities also exist. The translation between the extended ASCII characters in DOS® and Windows® does not seem to work 100%, hence the poor replica in Figure 2-1.



Computer Number: P400	Title: MR	Name/Initials: LN	Surname: PALI
Hospital Number: 401207	Population Group: Black	Sex: [✓ Male] Female	

Postal Address	Residential Address
1650	1650
BOTSHABELO	BOTSHABELO
BFN	BFN
9301	9301
Telephone Home:	Telephone Work:
Medical Aid:	FH Genotype:
Medical Aid Number:	

Browsing

Figure 2-2: The new graphic user interface of Hearts 32

Since this research specifically addresses the issue of digital acquisition of ECG data for the HP4745A, (together with digital storage of ECG data and the graphic reconstruction of these data) the main focus of the remainder of the thesis will be this topic.

2. Related systems currently in use

Cardio Control BV (Cardio Control BV) has developed a product called Cardio Perfect. This innovative system uses a small, portable electrocardiograph that can directly interface with any IBM compatible PC equipped with a serial port. It is an extremely user friendly system, allowing for real-time display (of the electrocardiogram being recorded) on screen, as well as browsing, storing, comparison, printing and analysis of the electrocardiogram tracing. The high price of thermal paper *versus* normal paper is discussed in the introduction of Chapter 3.

Since reports can be printed using laser, ink jet or dot matrix printers, the price per printed electrocardiogram tracing is cheaper than with the thermal transfer process. Electrocardiogram data files are small; a minimal electrocardiogram (that is a recording of only three leads for 2.5 seconds) will need only 10 KB, an important issue for transmitting data by modem. (Cardio Control BV). The Cardio Perfect kit includes the hardware (electrocardiograph and leads) as well as the software needed for viewing ECGs on a PC. Since the software only works with the Cardio Perfect electrocardiograph, this option becomes less viable for a user who already has a number of existing, non-Cardio Perfect cardiographs in use. In order to use the Cardio Perfect system, the existing electrocardiograph machines need to be replaced.

Marquette Electronics (Marquette Electronics, 1992) offers the MAC VU (Microprocessor Augmented Cardiograph, Virtually Unmatched), a system combining an electrocardiograph and a computer, mounted on a portable trolley. The high quality Cathode Ray Tube (CRT) unit allows for superior electrocardiogram wave form display, allowing the cardiologist to view the electrocardiogram in real time, as it is recorded. The CRT unit is also used to display instructions and on-line help information to the user. The printer provides electrocardiogram reports that are hard copies of the information displayed on the CRT, at a very high resolution of 1000 lines per inch. These hard copies are identified by bar-codes, for easy retrieval when using Marquette's MUSE® Network System. Roughly 200 electrocardiogram tracings can be stored on a normal 1.44 MB stiffy disk which can be read by an IBM compatible computer. (The data contained in the files do not make sense on their own - one would need specialised software to interpret them.) The system has extensive communication capabilities, allowing transmission of ECGs to and from other Marquette electrocardiographs and electrocardiogram management systems *via* the built-in RS-232 communications port. Computer analysis of electrocardiograms is performed based on a still-growing database of over 5 million clinically correlated electrocardiograms (Marquette electronics, 1992).

The HP4745A cardiograph has the capability to store ECGs in main memory and allows transmission to other HP cardiographs as well as to HP's electrocardiogram

Management System *via* a serial interface (RS-232 port). (Hewlett-Packard Company, 1988 : 5.1 - 5.14, 6.1 - 6.19). The electrocardiogram Management System aims to automate filing, retrieval and transcription of HP PageWriter electrocardiogram tracings, which are easily identifiable using bar-codes. The system provides for high speed on-line storage of electrocardiogram tracings which allows rapid retrieval and computer analysis for comparison of analysis statements with previous electrocardiograms. Electrocardiogram morphology changes can also be viewed easily with superimposed electrocardiogram tracings. Electrocardiogram tracings are printed onto normal printer paper that does not fade (laser printouts), whilst costing less per page than a page printed with the thermal transfer method. The system is already integrated with Novell® Netware®, making it easy to connect with a large installed base of Local Area Networks. Unattended backup to Digital Audio Tape is an integral part of the system, enhancing data integrity and relieving the user of the burden of daily backup duties. (Hewlett-Packard Company, 1995).

These solutions are not economically viable, as already discussed in section 2, page 5 in Chapter 1.

3. Digital Storage of ECG Data

The electrocardiogram is an instrument that receives its input in the form of voltages. This means that the instrument is by its very nature an analogue device. Since we are interested in digital capture of the ECG data, this poses an interesting problem: how will the analogue information be converted into digital information? Most modern ECG machines perform this conversion internally.

According to Kennedy (Kennedy, Ratcliff, 1987 : 186), ambulatory (Holter) electrocardiograph instrumentation and computer technology coexisted for more than two decades before integration allowed practical advantages in both clinical practice and clinical research. (Ambulatory electrocardiography is performed when a special electrocardiogram recording device is connected to a patient for an extended period of time, typically 24 hours.)

"A 24-hour ambulatory electrocardiogram examination usually results in the identification and classification of more than 100,000 cardiac cycles. Because of diverse changes and forms of cardiac rhythm occurring throughout a 24-hour diurnal cycle, computer data formatting and presentation techniques greatly aid and facilitate the clinical understanding of the ambulatory electrocardiographic data in a practical sense." (Kennedy, Ratcliff, 1987 : 187). The sheer volume of such a data set makes it impractical to evaluate by hand!

Other important issues include storage and retrieval of ambulatory electrocardiogram data, prompt access to patient records, duplication of reports, computer aided analysis and comparison of previous ambulatory electrocardiogram records. These issues do not only apply to ambulatory electrocardiogram tracings, but also to normal, resting electrocardiogram tracings.

In an article by Mustard (Mustard *et al*, 1990 : 65) it is mentioned that most researchers measure physiological variables using analogue devices, and record results manually. These results are then fed into computers for storage and analysis. One would be able to enhance the quality of the data and speed up work flow by directly storing the data on a computer system.

Quite a few researchers experimented with analogue to digital conversion of signals obtained from various physiological measurements: Axenborg (Axenborg, 1989 : 75 - 85), Brodie (Brodie, Mann, 1982), Farrell (Farrell, 1987 : 151 - 159), Herbst (Herbst *et al*, 1991 : 407 - 415), Jossinet (Jossinet *et al*, 1990 : 253 - 260), Mustard (Mustard *et al*, 1990 : 65 - 74), Piper (Piper *et al*, 1987 : 279 - 291), Van Vliet (Van Vliet, West, Road, 1987 : 143 - 150). Although not all these authors investigated the capturing of electrocardiogram data, it is interesting to take note of their work, as this aids in understanding the concept of converting and capturing results.

The amount of ECG data that is recorded tends to become quite large very quickly, as a sample rate of between 100 Hz and 1000 Hz is used. Due to the finite size of the memory found on electrocardiographs, data cannot be stored as is. Different compression schemes have been developed to overcome this problem. The schemes used are lossy compression schemes (in contrast with lossless compression schemes). With lossy compression, compressed data cannot be reconstructed to exactly match the input data. Normally, this type of compression is used for audio and video signals, where it does not matter too much if slight distortion occurs. The human organs of sense can normally not detect these changes. With information like the electrocardiogram, however, care has to be taken that a clinically acceptable level of distortion is maintained. It would serve no purpose to have the data reduced by 80%, but to have the compression resulting in inaccurate presentation and interpretation.

It is therefore necessary to use a lossy compression technique which provides a good compression ratio whilst maintaining a clinically acceptable level of distortion. Examples of such lossy compression schemes used for compressing electrocardiogram tracings include CORTES (Coordinate-Reduction-Time-Encoding System algorithm) (Abenstein, Tompkins, 1982 : 46 - 47, Jalaleddine *et al*, 1990 : 334), TP (Turning Point algorithm) (Abenstein, Tompkins, 1982 : 44, Jalaleddine *et al*, 1990 : 334), AZTEC (Amplitude-Zone-Time-Epoch-Coding algorithm) (Abenstein, Tompkins, 1982 : 44 - 46, Jalaleddine *et al*, 1990 : 333 - 334), SLOPE (Tai, 1991 : 176 - 179), CORNER (Tai, 1992 : 585 - 589) and AZTDIS (Tai, 1993 : 511 - 515).

Hewlett Packard uses their own data smoothing routines and data compression scheme in the HP4745A. If the data of certain leads are smoothed before compression, the compression process yields better results. These algorithms are documented in HP's digital communications protocol. (Hewlett-Packard Company, 1985).

Another factor which will influence the size of the data set (the captured electrocardiogram tracing) is the number of bits that is used to represent each value in the tracing. There is a direct relationship between the number of bits and the size of the data set. Berson (Berson, Wojick, Pipberger, 1977 : 382) found that although the American Heart Association suggested a precision level of 9 bits, 8-bit data are sufficient for representing electrocardiogram data. This has important implications for storage, digital data transmission and use of microcomputers with a typical word size of 8 or 16 bits (Berson, Wojick, Pipberger, 1977 : 382), as a smaller number of bits will result in smaller data sets and faster transmission over telephone lines and computer networks.

4. Problem solution

To solve the problem as outlined in the hypothesis in Chapter 1 (page 5), the following sub-problems will need to be solved successfully:

- Extracting stored electrocardiogram tracings from a HP4745A PageWriter II Cardiograph.
- Selecting the optimal storage method available for storage of electrocardiogram tracings in a database on a Personal Computer.
- Successful integration of the digital electrocardiogram tracing information with the *Hearts* database in use at the Department of Cardiology, Universitas Hospital, Bloemfontein.
- Recreating a complete electrocardiogram tracing from a PC-based database on a computer screen or on paper. This includes superimposing of electrocardiograms for recognition of trends as therapy progresses.
- Creating callipers to allow easy measurement of the different components of an electrocardiogram tracing on screen.
- Implementing all of the above in a layered manner when developing the software product, to protect the software product from changes introduced when new electrocardiograph equipment is used.

5. Research method

The research was to be carried out in five phases:

1. Obtain technical documentation on the HP4745A PageWriter II Cardiograph from the Hewlett-Packard Company.
2. Develop an interface between the HP4745A PageWriter II Cardiograph and a Personal Computer in order to establish communication for data transfer.

3. Select the most optimal storage method available for storage of electrocardiogram tracings from a HP4745A PageWriter II Cardiograph on the hard disk of a Personal Computer.
4. Develop a graphical interface to reconstruct the saved electrocardiogram on a computer screen as well as on any Windows®-compatible printer.
5. Integrate the electrocardiogram into the existing *Hearts* database in use at the Department of Cardiology.

6. Summary

The existing *Hearts* database has some serious shortcomings. Some additional functionality is needed to solve the problems. A new version of the *Hearts* database needs to be developed. If ECG data are to be stored in the *Hearts* database, cognisance of digital storage of ECG data needs to be taken.

A system that will allow handling of digital ECG data will need to acquire, decode, store, manage, retrieve and manipulate these data to succeed.

Chapter 3

The Electrocardiogram

1. Introduction

With every heartbeat blood is pumped through the body, helping to sustain life. Without this organ we cannot survive. Cardiac disease has become common, killing large numbers of people every day. Doctors are continuously trying to gather more information about this condition, in order to save lives.

Every heartbeat is caused by the contraction of the myocardial cells (muscle cells of the heart). The electrical stimulus originating in the SA node and conducted to the ventricles *via* the AV node causes depolarisation of the myocardial cells which leads to their contraction. During their resting stage, the myocardial cells are polarised (in this case polarisation means that the inside of the cells become negatively charged). (Dubin, 1989 : 7). As the wave of depolarisation spreads through the myocardium (heart muscle), it contracts.

The electrocardiograph machine is a medical diagnostic tool which is used to record the electrical activity of the heart. It produces a permanent record (the electrocardiogram, or ECG[1]) of the heart's electrical activity. The ECG is interpreted by a cardiologist in order to diagnose the condition of the heart.

Although initial experiments and observations started round about 1855, the "electrokardiogram" evolved only around 1901, thanks to work done by Einthoven. (Dubin, 1989 : 4).

The electric activity that passes through the heart causes electrical potentials, which can be detected on the skin. Electrodes that are connected to the body at specific points are sensitive enough to detect the skin potentials which are recorded as the ECG.

[1] Please refer to Figure 3-20 (page 48) for an example of an ECG.

What makes the ECG so useful, is that it is a non-invasive diagnostic technique, enabling a cardiologist to examine cardiac conditions without exposing the heart.

Put into simple terms, the various components[2] of an electrocardiograph have to perform the following tasks:

- Amplification of signals. "In order to detect body potentials, electrodes must convert the ionic currents in the body into electron current in the wire." (Tompkins, Webster, 1981 : 6).

- Screening of the electrocardiograph from high voltages induced by electro-surgical and defibrillation units. (Tompkins, Webster, 1981 : 11).

- The amplifier should selectively amplify the electrocardiogram signal and reject electrical interference. (Tompkins, Webster, 1981 : 11).

- Bandpass filtering is needed to allow for high gain of the electrocardiogram (Tompkins, Webster, 1981 : 11). Interference should be reduced as much as possible. Electrical power lines are a major source of interference as they radiate electrical and magnetic fields which can have an adverse influence on the electrocardiogram machine.

- Analogue to digital conversion. The measured quantities are of an analogue nature. In order to present the values on a computer, a specialised piece of hardware is needed to convert these analogue signals into their digital equivalents.

- Display and recording. Different ways of communicating the electrocardiogram tracing include paper copies, cathode-ray-tube display and digital storage. Paper copies of electrocardiograms can be produced using a pen recorder. A moving pen is used to record the data on graph paper. Another method uses a heated stylus on thermal paper. These thermal transfer systems have a high running cost, since

2 Please refer to Figure 3-19 (page 40) for a schematic representation of these components.

thermal transfer paper costs more than normal paper. When using plotter pens with flowing ink and a ball inside the tip of the pen, clogging can occur as the ink dries out. Some systems rely on pressurised-ink systems to overcome clogged ink pens. (Tompkins, Webster, 1981 : 21). Many systems in use employ a dot matrix printing system. Such a printing mechanism is not well suited to the application, since the resolution of the dot matrix printer does not allow for a continuous graph. This makes detailed reading of the graph difficult for the cardiologist.

- It is also possible to display an electrocardiogram on a cathode-ray-tube display (CRT). Examining complete physiological wave forms using a CRT can be very difficult. Non-fade displays address this problem by holding and displaying complete wave forms. (Tompkins, Webster, 1981 : 21). The CRT display does not provide a permanent copy of the electrocardiogram as in the case of a printed copy.

- Distribution. Electrocardiogram data can be distributed over computer networks, telephone links and by radio telemetry (wireless links). (Tompkins, Webster, 1981 : 22).

Before understanding the functioning of the electrocardiograph and the resulting electrocardiogram, it is necessary to have a basic understanding of the anatomy of the heart.

1.1. Brief overview of the anatomy of the heart

Put into simple terms, the heart consists of two pumps, a left and a right side pump that cause blood to circulate through the lungs and the rest of the body. (Meyer *et al*, 1988 : 30.1).

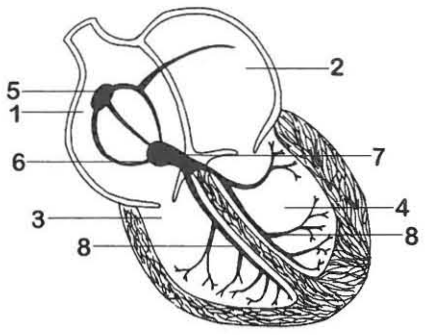


Figure 3-1: Schematic representation of the heart (frontal view)

(Meyer *et al*, 1988 : 30.5)

Table 3-1: Different parts of the heart

Number	Item	Description
1	Right Atrium	According to Guyton (Guyton, 1966 : 234) blood flows from the great veins into the atria. About 70% of this flows directly into the ventricles before atrial contraction. Atrial contraction is responsible for the remaining 30% filling of the ventricles. The right atrium transfers blood from the superior and inferior vena cava to the right ventricle.
2	Left Atrium	The left atrium transfers blood from the pulmonary veins to the left ventricle.
3	Right Ventricle	The right ventricle circulates blood to the pulmonary circulation.
4	Left Ventricle	The left ventricle transfers blood to the systemic circulation.
5	SA Node	The Sinoatrial Node is located in the upper part of the wall of the Right Atrium and consists of a concentration of P-cells which are responsible for the rate of heart contraction. Impulses which cause the heart to contract normally, start here. Contractions are rhythmic and spontaneous in a normal, resting person and occur at an average rate of 70/minute. Thus the SA Node is the primary pacemaker of the heart. (Meyer *et al*, 1988 : 30.4 - 30.5).
6	AV Node	The AV Node is located in the lower part of the wall of the Right Atrium. The cellular structure resembles that of the SA Node. (Meyer *et al*, 1988 : 30.5).

Table 3-1: Different parts of the heart (continued)

Number	Item	Description
7	His Bundle	The AV Node forms the head of an elongated fibre bundle of conduction tissue which extends to the ventricles. This is called the His Bundle and consists of Purkinje cells. Purkinje cells conduct electrical impulses at an accelerated rate. (Meyer *et al*, 1988 : 30.5 - 30.6).
8	Right and Left Bundle Branches	The His Bundle forks into a left and right branch. Each of these branches forms smaller branches in the sub-endocardial tissue. This forms the Purkinje Network. (Meyer *et al*, 1988 : 30.5 - 30.6).

Although the cells of the SA Node, AV Node, His Bundle, Bundle Branches and Purkinje fibres differ anatomically and physiologically, they form an integrated pacemaker conducting system (Figure 3-1, page 21). The AV Node is the only electrical connection switch between the atria and the ventricles. It is not possible for impulses generated in the atria to reach the ventricles if the AV Node is damaged. Such a condition is called "heart block" and can either be total heart block or partial heart block. (Meyer *et al*, 1988 : 30.6).

1.2. Position of the heart

As indicated by Figure 3-2, the position of the heart within the thorax is that of an upside down cone. (Meyer *et al*, 1988 : 30.1).

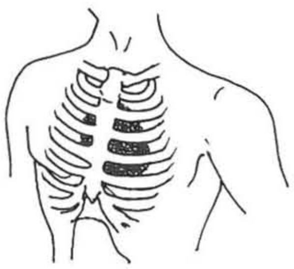


Figure 3-2: The position of the heart in the thorax

(Meyer *et al*, 1988 : 32.4)

1.3. A brief description of the pump action of the heart

The function of the heart, as already stated, is to circulate blood through the body. Oxygen (O_2) rich blood must be transported from the lungs to the rest of the body. Blood containing carbon dioxide (CO_2) needs to be pumped to the lungs, where the CO_2 is removed and O_2 is added again.

The pump action of the heart is caused by the contraction of the heart muscle (myocardium). This is illustrated in Figures 3-1, 3-3 and 3-4. Such contractions are the result of electrical activity in the myocardium. In 1858 Kollicker and Müller proved that contraction of the myocardium is accompanied by electrical activity (Figures 3-15, 3-16, 3-17 and 3-18). (Meyer *et al*, 1988 : 32.2).

During the resting state of the heart, muscle cells (myocardial cells) are polarised. This means that the inside of the cells is negatively charged. (Dubin, 1989 : 7). Myocardial cells are stimulated to contract as the charge within each cell changes to positive. This process is called depolarisation and is illustrated in Figures 3-15 and 3-16. (Dubin, 1989 : 8). Progressive contraction is achieved as the wave of positive charges advances through the myocardial cells.

During repolarisation, the negative charge within the myocardial cells is restored. As can be seen from Figure 3-17, myocardial cells do not respond to repolarisation. (Dubin, 1989 : 9).

The electric impulse for cardiac stimulation is initiated in the Sinus Node. This wave of depolarisation proceeds outward from the Sinus node and causes both atria to contract (Figure 3-3). (Dubin, 1989 : 13). When the AV Node receives the impulse, a brief pause occurs. This allows the blood contained in the atria to enter the ventricles. The electrical stimulus passes rapidly down the His Bundle, Left and Right Bundle Branches and finally through the terminal Purkinje fibres, causing the distribution of depolarisation to the ventricular myocardial cells. (Dubin, 1989 : 18). The result of this is ventricular contraction, which causes the blood to be expelled from the ventricles (Figure 3-4).

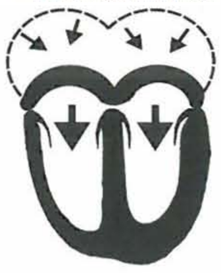


Figure 3-3: Atrial contraction

(Dubin, 1989 : 14)

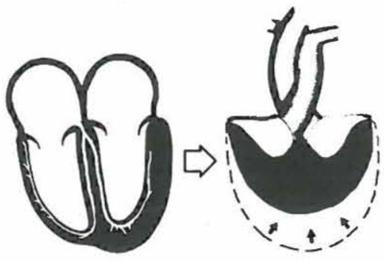


Figure 3-4: Ventricular contraction

(Dubin, 1989 : 18)

2. The Electrocardiograph

2.1. History

Ludwig and Waller showed in 1887 that the electrical activity of the myocardium could be monitored from a person's skin. (Meyer *et al*, 1988 : 32.2, Dubin, 1989 : 2). Their "capillary electrometer" was interesting, but of little use since it did not allow for permanent recording of the findings. (Dubin, 1989 : 2).

It was only in 1903 that Einthoven (now seen as the father of electrocardiography) managed to produce a permanent record of the heart's electrical activity by projecting a light beam across a moving silvered wire which was suspended through two holes drilled in a large permanent magnet. Two skin sensors were placed on a man's body and attached to this silvered wire. The movements of the wire represented the man's heartbeat. These movements were recorded on a scroll of moving photographic paper. (Dubin, 1989 : 3 - 4).

The electrocardiograph is the instrument used to record the electrical activity of the heart. The result of such a recording is called an electrocardiogram (ECG). An example of an ECG can be seen in Figure 3-20 on page 48.

2.2. Electrodes

In order to measure electrical potential, a complete circuit must exist between the tissue acting as the conducting medium and the measuring device. In simple terms, two electrodes are needed to measure an electrical potential (Figure 3-5). (Meyer *et al*, 1988 : 32.2).

Electrodes are made of corrosion proof metal. Since the skin is a relatively poor conductor of electricity, some preparation is needed:

- Electrodes must be clean (free of corrosion).
- The skin should be briskly rubbed with the edge of the electrode until it is slightly red.
- An electrolyte cream (such as Redux® cream) should be applied to the prepared areas of the skin.

Electrodes are kept in place by fastening them with rubber straps (wrist and ankles). Rubber suction cups are used for applying electrodes to the chest area. A modern technique employs the use of disposable electrodes, which are kept in place by an adhesive substance.

2.3. Leads

The relative positions of the two electrodes connected to the body (for measuring electrical activity of the heart) are called leads. The standard ECG consists of 12 separate leads. (Dubin, 1989 : 30, Meyer *et al*, 1988 : 32.3). This means that electrodes are placed in 12 different ways on the body. These 12 leads consists of 6 limb leads and 6 chest leads. An enumeration of these leads can be found in Table 3-2 below.

Table 3-2: The 12 different leads of a standard ECG

Limb Leads	Chest Leads
I	V1
II	V2
III	V3
aVR	V4
aVL	V5
aVF	V6

The limb leads reflect electrical activity in the vertical axis, whereas the precordial leads (V1 - V6) reflect the electrical activity in the horizontal axis.

(Dubin, 1989 : 30). The 12 leads produce 12 different "views" of the simultaneous electrical activity of the heart at a given time. An example of these 12 leads can be found in Figure 3-20 (page 48), where the leads are identified by items B, C, D and E.

Three of these 12 leads are bi-polar (this means that both electrodes are connected to an electrical potential). One of the electrodes is always positive, and the other is always negative. The remaining 9 leads are strictly speaking also bi-polar, since two electrodes are used per lead. Since only one of the electrodes in each lead is connected to an electrical potential, these leads are termed uni-polar leads. (The other electrode is artificially kept at a constant potential.) The uni-polar leads thus register changes in electrical potential in respect of a constant electrical potential at another point. (Meyer *et al*, 1988 : 32.3).

2.4. Bi-polar Limb Leads

Einthoven used the 3 bi-polar limb leads for the first time, forming Einthoven's triangle. In this scheme, electrodes are connected to the right and left arm, as well as to the left leg. These electrodes form the limb leads and the placement of these electrodes forms a triangle, as shown in Figure 3-5. Leads I, II and II are also known as the standard leads.

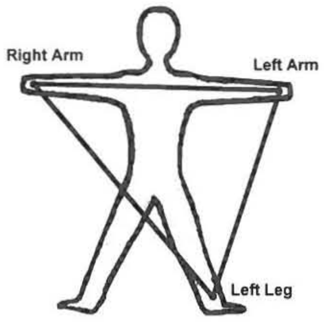


Figure 3-5: Einthoven's triangle and the limb leads

(Dubin, 1989 : 31)

In this model it is assumed that the heart is contained within an imaginary equilateral triangle, the corners of which are formed by the basis of the right and left arm as well as that of the left leg (Figure 3-6). The limbs thus act as electrodes conducting electricity away from the heart. This enables us to measure the electrical activity of the heart by using electrodes connected to the left and right wrist, as well as to the left ankle. The standard leads are named I, II and III, respectively.

In Lead I the negative electrode is connected to the right wrist. The positive electrode is connected to the left wrist. In Lead II the negative electrode is connected to the right wrist and the positive electrode is connected to the left ankle. In Lead III the negative electrode is connected to the left wrist, while the positive electrode is connected to the left ankle (Figure 3-6).

Lead I measures the electrical potential between the right and left arm. Lead II measures the electrical potential between the left leg and the right arm, while Lead III measures the electrical potential between the left leg and the left arm. (Meyer *et al*, 1988 : 32.3).

Since there is a close correlation between the standard leads, it does not really matter which lead is used for the identification of the heart's rhythm. When identifying the type, location and extent of lesions, it is particularly important to choose the correct lead since the different leads are affected in different ways.

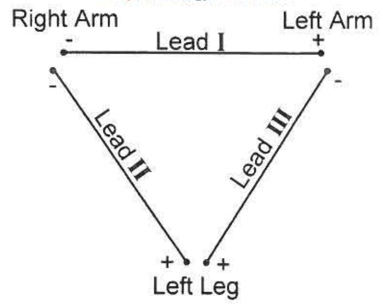


Figure 3-6: Einthoven's equilateral triangle

(Dubin, 1989 : 32)

The standard leads fall short on the following points:

1. Both electrodes are connected at roughly the same distance from the heart, on the limbs.
2. Both electrodes are subjected to an electrical potential and the tracing only represents the difference between the two electrical potentials at two points (Lead I = Left Arm - Right Arm, Lead II = Left Leg - Right Arm, Lead III = Left Leg - Left Arm). The electrical potential cannot be measured at any given point.
3. The electrodes in the standard leads are located on the same horizontal plane of the body.

2.5. Einthoven's Law

According to Einthoven, the electrical potentials (height and depth) recorded in Lead II equal the sum of the electrical potentials of Lead I and III. Thus:

Lead II = Lead I + Lead III

From this law it can be seen that the potential of a complex in a third lead can be calculated if the potential of the same complex in the other two leads are known. (Meyer *et al*, 1988 : 32.3 - 4).

2.6. *Uni-polar Chest Leads (Precordial Leads)*

The six chest leads are obtained by placing six positive electrodes (one electrode per lead) at six progressively different positions around the chest, as illustrated in Figure 3-7. The second "electrode" is formed by the connection of three electrodes (left arm, right arm and left leg). This central terminal is connected to the electrocardiograph and the potential is kept at 0 through the use of a resistor. The second electrode is called a neutral electrode, since it is not subjected to an electrical potential. (Meyer *et al*, 1988 : 32.3 - 4). The chest leads are numbered from V1 to V6 and move successively from the person's right to left side.

The names of these leads are formed by numbering the anatomical positions where each lead is placed. (See Figure 3-7.) Each number is prefixed by the letter V. This is short for Vector.

Since the electrode sensor for the chest leads is positive, a depolarisation wave moving towards a skin sensor produces a positive (or upward) deflection on the ECG. The positive wave of ventricular depolarisation moves progressively towards the positive electrode of Lead V6.

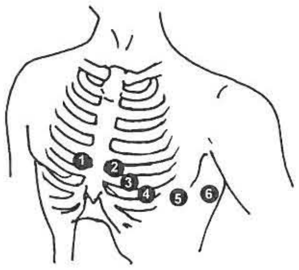


Figure 3-7: The six chest leads

(Meyer *et al*, 1988 : 32.4)

"If leads V1 through V6 are assumed to be the spokes of a wheel, the centre of the wheel is the AV Node." (Dubin, 1989 : 42). As can be seen from Figure 3-8, the body is cut into top and bottom halves by the plane of the chest leads. This is called the horizontal plane.

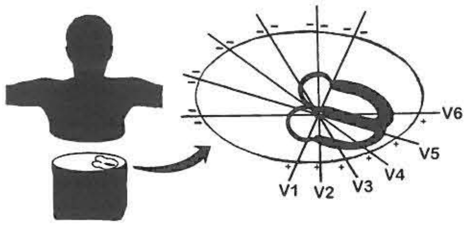


Figure 3-8: The horizontal plane

(Dubin, 1989 : 42)

From Figure 3-8 it can be seen that Leads V1 and V2 provide more information about the electrical activity in the right ventricle, while Leads V5 and V6 mainly provide information about the electrical activity in the left ventricle.

2.7. Uni-polar Limb Leads

The electrodes used for the chest leads are connected nearly directly over the heart. After these leads proved to be clinically useful, the uni-polar limb leads were introduced using the same design. The electrode is placed on each of the three limbs, and these leads are called VR (right arm), VL (left arm) and VF (left foot), respectively (Figures 3-11 and 3-13).

The electrical activity registered in these leads is small, due to the small electrical potential in the limbs. It was empirically determined that the omission of the resistor and the inclusion of the potential of the other two limb leads (together with their resistors) produced larger potentials. This is the reason why the leads are called augmented uni-polar limb leads.

Using augmented leads also changes the name of the leads as follows: aVR, aVL and aVF. (Meyer *et al*, 1988 : 32.3 - 4).

2.8. The electrical axis of the heart

By convention, a vector is represented by an arrow. A vector consists of both size and direction. The length of the arrow indicates the size of the vector, while the direction of the arrow indicates the direction of the vector.

The depolarisation of the normal heart progresses in an orderly fashion, and follows a fixed pattern. Quite a few vectors are simultaneously present, since different parts of the heart (which are not necessarily in close proximity of each other) depolarise simultaneously. The sum of these vectors can be seen as one vector, called the immediate electrical vector. The size and direction of this vector change continuously as depolarisation progresses. This changing in direction corresponds with a three dimensional rotation movement around a central point. The size of the immediate vector starts at zero (start of depolarisation), increases up to a maximum and then gradually recedes back to zero.

If the arrows of immediate electrical vectors are connected, the resulting line (loop) represents a spatial vector cardiogram. The longest arrow in the figure represents the dominant heart vector and the direction represents the mean electrical axis of the heart, as depicted in Figure 3-9.

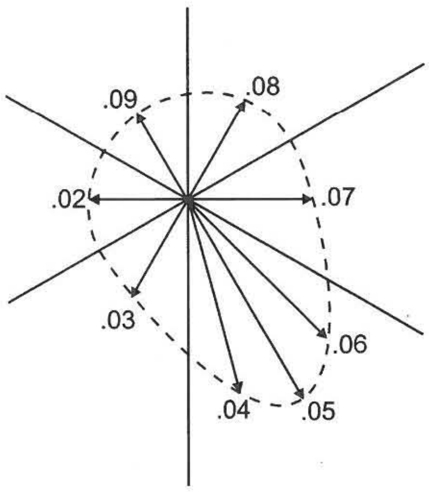


Figure 3-9: The three dimensional vector cardiogram and the mean electrical axis of the heart

(Meyer *et al*, 1988 : 32.8)

The electrical axis of the heart is of clinical relevance, since the position of the heart is not exactly the same in all people. Heart lesions (such as myocardial infarction) influence the direction of the axis. It is not sufficient to observe that the axis has been displaced. The displacement can be expressed in terms of degrees, and the polarity can be expressed in terms of a negative or positive polarity.

The hypothetical line connecting the two electrodes of a given lead is called the lead axis. In this way, the horizontal line connecting the left arm and right arm forms the axis of Lead I. The axes of 2, 3 or all 6 of the limb leads can be used to numerically calculate the mean electrical axis.

In order to create 6 intersecting lines of reference, the sides of Einthoven's triangle are rearranged so that they cross one another at an angle of 60° at a central point. The lines may have been moved, but they still remain at the same angle as can be seen from Figure 3-10.

First start by rearranging the 3 standard leads, I, II and III.

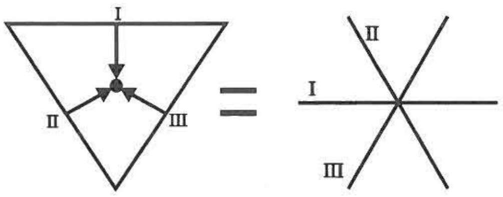


Figure 3-10: 3 Intersecting lines of reference for Leads I, II and III.

(Dubin, 1989 : 33)

Now proceed by following the same procedure for Leads aVR, aVL and aVF. Note that these limb leads will intersect at different angles to produce three other lines of reference.

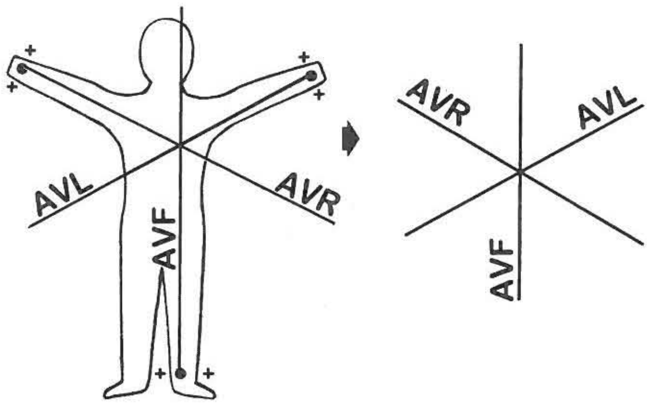


Figure 3-11: 3 Intersecting lines of reference for Leads aVR, aVL and aVF

(Dubin, 1989 : 36)

The 6 intersecting limb leads are spaced 30° from one another. These limb leads may be visualised as lying in a flat plane over the person's chest. This is referred to as the frontal plane (Figure 3-12, Figure 3-13).

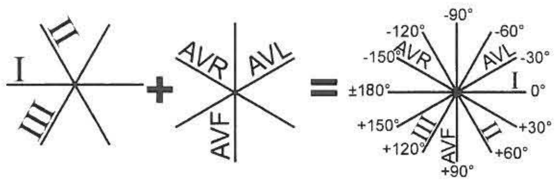


Figure 3-12: 6 Intersecting lines of reference (I, II, III, aVR, aVL and aVF)

(Dubin, 1989 : 37)

The same cardiac activity is recorded in each of the leads. Since the electrical activity is monitored from a different angle for each lead (Figure 3-13), the waves in the various leads differ.

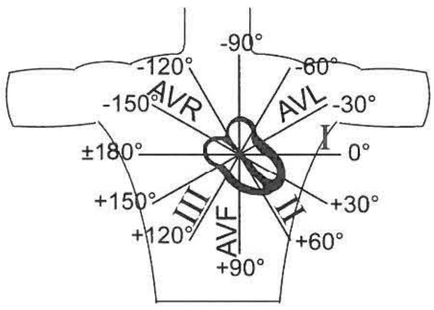


Figure 3-13: Different views of the same cardiac activity

(Dubin, 1989 : 38)

The precordial leads are placed in the 2nd interspace on the right side of the sternum (V1), on the left side (V2), over the apex of the heart (V4), and in the 5th interspace in the anterior axiallary line and in the midaxiallary line. V3 is located between V2 and V4. These leads start predominantly negative as they point towards the cavity of the heart (V1), and become progressively positive towards leads V5 and V2. These leads reflect electrical activity in the frontal plane of the heart.

3. The Electrocardiogram

Dubin (Dubin, 1989 : 5) states that the electrocardiogram provides a valuable, permanent record of the heart's function. It is important to note that the electrocardiogram reflects electrical activity in the heart, not contractile or pump functions. (There may even be complete dissociation between these functions in some clinical conditions, for example electromechanical dissociation.) The electrical changes that occur during a cardiac cycle normally cause 5 to 6 deviations, which are identified from left to right by the letters P, Q, R, S, T and U (if present). The periods where an isoelectric condition exists in the heart are called segments. These segments normally occur between P and Q (PQ-segment) and between S and T (ST-segment). In normal electrocardiograms, recorded on standard electrocardiograph equipment, P, R, T and U are normally positive (upward deflection). Q and S are negative (downward deflection). (Meyer *et al*, 1988 : 32.3 - 4).

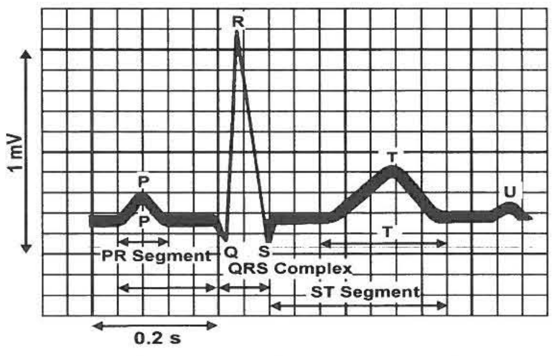


Figure 3-14: A conventional Electrocardiogram

(Meyer *et al*, 1988 : 32.5)

The electrocardiogram is inscribed on ruled paper. Such a page of graph paper is divided into squares. This design permits direct determination of the electrical activity, duration of different components of the ECG, as well as the heart rate. The smallest divisions measure one millimetre in height and one millimetre in width. For every five small divisions (blocks) a heavy line is drawn (Figure 3-14).

Deflections in the wave (both upwards and downwards) are measured in millimetres and represent a measure of voltage. Each millimetre represents a potential of 0.1 mV. A potential of 0.5 mV is represented by each heavy horizontal line. This voltage is present as a result of the electrical activity of the heart. Upward deflections are also called positive deflections. Likewise, downward deflections are called negative deflections. (Dubin, 1989 : 27). By measuring along the vertical axis, we can find a measure of the electrical activity of any part of the cardiac cycle.

The horizontal axis represents time. Each of the small divisions represents 0.04 seconds. Since there are five of these divisions for every heavy line, the distance between heavy lines represents 0.2 seconds. The duration of any part of the cardiac cycle can be found by measuring along the horizontal axis. (Dubin, 1989 : 29).

3.1. P-wave

The P-wave represents atrial depolarisation and the accompanying atrial contraction (both atria) (Figures 3-14, 3-15 and 3-18). It is worth noting that contraction and depolarisation do not occur simultaneously, but for the purpose of this discussion, these events are deemed to occur simultaneously.

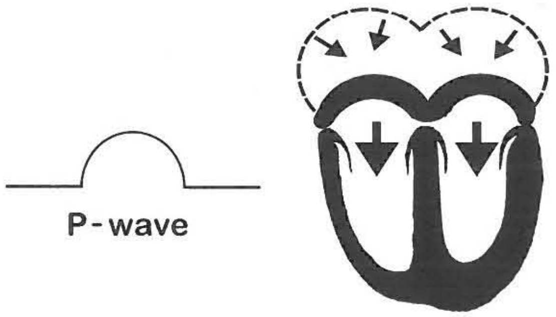


Figure 3-15: Atrial contraction and the P-Wave

(Dubin, 1989 : 14)

The pause that is present towards the right of the P wave is caused by the fact that the stimulus of depolarisation slows down as it enters the AV node. (This is necessary to allow blood from the atria to pass through the AV valves.)

3.2. PR-Segment

The PR-segment is represented by the distance between the beginning of the P-wave and the first deflection of the QRS-wave (Figure 3-14). It would be more correct to talk about the PQ-segment, but since the Q-wave is often absent, it is common practice to talk of the PR-segment. Depolarisation of a part of the AV-Node, the His Bundle as well as the Bundle Branches, occurs during the PR-segment. The electrical activity caused by these tissues is very small. This is reflected by the flat PR-segment. Atrial repolarisation does not produce any visual deflections, since ventricular depolarisation and atrial repolarisation coincide, obscuring any deflections caused by atrial repolarisation. (Meyer *et al*, 1988 : 32.3 - 6).

3.3. QRS-Complex

The depolarisation of the myocardial cells produces the QRS-complex. The QRS-complex, in turn, represents the initiation of ventricular contraction. Although mechanical contraction extends beyond the QRS-complex, we will consider the QRS-complex to represent ventricular contraction.

The downward Q-wave (when present) indicates the start of the QRS-complex (Figures 3-14, 3-16 and 3-18). A Q-wave is not present in all tracings. A positive R-wave follows the Q-wave. By definition, the first negative wave of the QRS-complex is the Q-wave. Any upward (positive) deflection in a QRS-complex appearing before a "Q"-wave is NOT a Q-wave. It is actually the R-wave! The upward R-wave is followed by a downward S-wave. Thus, a negative deflection preceded by a positive deflection is a S-wave. (Dubin, 1989 : 20).

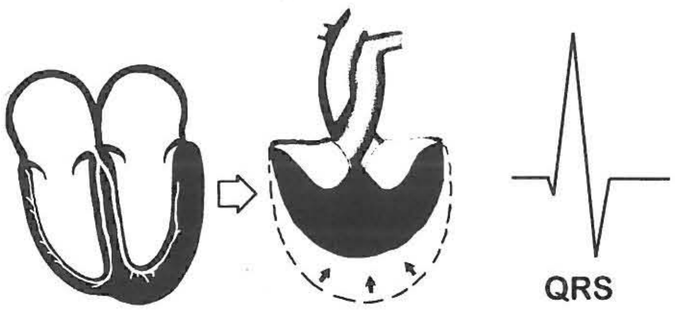


Figure 3-16: Ventricular contraction and the QRS Complex

(Dubin, 1989 : 18)

3.4. J-Point and the ST-Segment

The point where the QRS-complex ends and the ST-segment starts is called the J-point. The ST-segment starts at the J-point and continues up to the start of the T-wave. Normally the ST-segment is a flat piece of baseline. This indicates that no electrical activity takes place for the duration of the ST-segment (Figure 3-14). (Meyer *et al*, 1988 : 32.3 - 7).

3.5. T-Wave

The T-wave follows the ST-segment and is caused by the repolarisation of the ventricles. It is also known as the second ventricular complex (Figures 3-14, 3-17, 3-18). The ventricles have no physical response to repolarisation. The T-wave is strictly an electrical phenomenon recorded on the ECG. (Meyer *et al*, 1988 : 32.3 - 7).

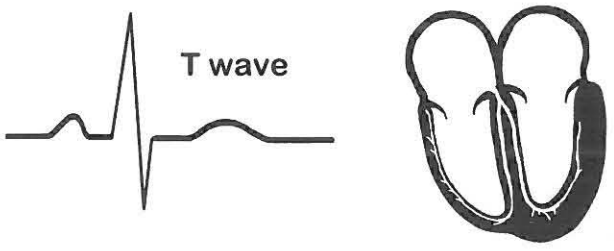


Figure 3-17: T-wave indicates no cardiac response

(Dubin, 1989 : 23)

3.6. U-Wave

This deflection closely follows the T-Wave and is normally positive (Figure 3-14). According to Meyer (Meyer *et al*, 1988 : 32.7) the exact origin of this wave is not known, but Meyer speculates that slowed and uneven repolarisation could be a likely cause.

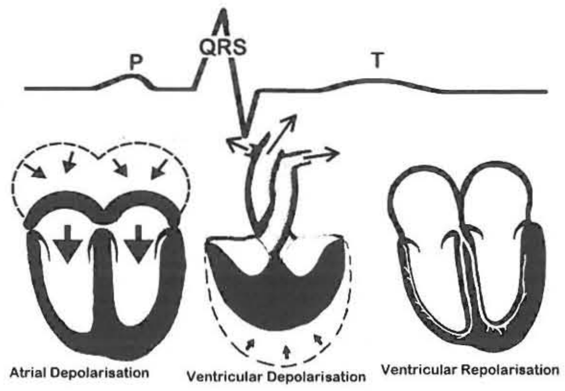


Figure 3-18: The cardiac cycle

(Dubin, 1989 : 24)

3.7. Time intervals in the ECG

The PR (or PQ) time is the elapsed time from the start of the P-wave up to the start of the QRS-complex (Figure 3-14). The PR-time represents the time needed for the electrical impulse to pass from the SA Node through the atria, AV-Node, His Bundle, Bundle Branches and Purkinje fibres (Figure 3-1).

The QT-time starts at the first deflection of the QRS-complex and continues to the end of the T-wave (Figure 3-14). QT-time represents the duration of ventricular systoly and diastoly.

Table 3-3 summarises the normal time span and amplitude of the different ECG components.

3.8. Summary of ECG components

Table 3-3: Summary of the normal time span and amplitude of the different ECG components

ECG Component	Normal duration (seconds)	Normal amplitude (mV)	Relevant cardiac activity
P-Wave	0.11	0.3	Atrial depolarisation
PR-Segment	0.14		
PR-Time	0.12 - 0.20		Depolarisation of atria, AV-Node, His Bundle and Bundle Branches

Table 3-3: Summary of the normal time span and amplitude of the different ECG components (continued)

ECG Component	Normal duration (seconds)	Normal amplitude (mV)	Relevant cardiac activity
Q-Wave	< 0.04		Start of ventricular depolarisation
QRS-Complex	0.08 - 0.11	0.5 - 3.0	Ventricular depolarisation
QT-Time	0.35 - 0.45		Ventricular depolarisation plus ventricular repolarisation
T-Wave	0.10 - 0.25	0.4	Ventricular repolarisation
U-Wave	0.10	0.1	

(Meyer *et al*, 1988 : 32.3 - 7). Please refer to Figure 3-14 as well.

3.9. Interpretation of the ECG

In the actual reading of the ECG, the following 5 general areas are checked:

1. Rate
2. Rhythm
3. Axis
4. Hypertrophy
5. Infarction

Text books dedicated to the reading and interpretation of ECGs are available. Suffice it to mention these areas. Interpretation of the ECG is normally done by a cardiologist.

4. The HP4745A PageWriter II Cardiograph

Since the Department of Cardiology, Universitas Hospital, Bloemfontein uses the HP4745A PageWriter II Cardiograph machines extensively, this equipment was used for this research.

4.1. Basic description of operation

The operation of the HP4745A ECG machine can be summarised as follows:

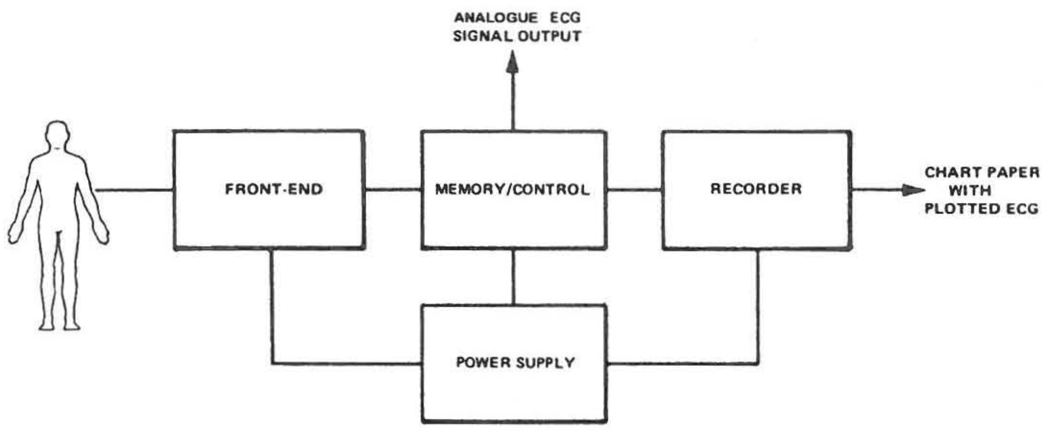


Figure 3-19: HP4745A Block Diagram

(Hewlett-Packard Company, 1989 : 3-14)

The Front End connects the patient to the HP4745A. It isolates and protects the patient whilst acquiring the ECG. The ECG data are applied from the Front End to the Memory/Control Board for buffering, after which the data are applied to the Recorder Assembly where ECG signals are drawn on chart paper. A modem can be connected to the Memory/Control Board for transmission of ECG data. AC power is applied to the Power Supply Board, which, in turn, supplies the DC voltages required for cardiograph operation. (Hewlett-Packard Company, 1989 : 3-1 to 3-2). A summary of the configuration settings can be found in Tables 3-4, 3-5, 3-6 and 3-7.

4.2. Configuration

The Department of Cardiology, Universitas Hospital, Bloemfontein, has a standardised setting that is used for the recording of all ECGs. These settings were taken into account for the purpose of creating of the data acquisition and graphics modules described in Chapters 4, 7 and 8, as well as the design of the ECG storage format described in Chapter 6. The information presented next serves to illuminate the choices available to the user.

Certain cardiograph responses can be preset on the HP4745A. These include:

- ECG Formats
- Characteristics of the printed record
- Patient ID and administrative information
- Filter and frequency settings
- Transmission guidelines
- Time and date

It is thus possible to determine which ECG format will be used by default. One can further determine which fields are mandatory and require input, and which fields can be ignored. Once a parameter has been configured, the setting is stored until changed again.

4.3. Summary of configurable functions

Displayed in the following 4 tables is a summary of configurable functions. Values entered in the column 'User Selections' indicate the actual configuration settings as found on the HP4745A ECG machines for the purpose of this study. (The ECG rhythm speed setting for the standard ECG programme is 25 mm/sec and the calibration 10 mm = 1 mV. Lead V1 is selected for the rhythm strip.)

Table 3-4: Format Configuration Options

Function	Parameter	User Selections
1.0 Record type?	Auto, Auto/RS, Manual, Rhythm	Auto/RS•
2.0 Auto speed?	25, 50 mm/sec	25 mm/sec
2.1 Auto sensitivity?	0.5 cm/mV, 0.5 (½ V), 1.0 cm/mV, 1.0 (½ V) 2.0 cm/mV, 2.0 (½ V)	1.0 cm/mV (10 mm/mV)
2.2 Auto lead length?	2.5, 5.0 sec	2.5 sec
2.3 /RS speed?	2.5, 5.0, 10.0, 12.5, 25.0, 50.0, 100.0 mm/sec	25 mm/sec•
2.4 /RS sensitivity?	0.25, 0.50, 1.00, 2.00 cm/mV	1.00 cm/mV•

• Figure 3-20 (page 48), Table 3-9 (page 46).

Table 3-4: Format Configuration Options (continued)

Function	Parameter	User Selections
2.5 /RS lead?	I, II, III aVR, aVL, aVF V1, V2, V3 V4, V5, V6 II(SP1), aVF(SP2), V5(SP3)	V1●
3.0 Manual speed?[3]	25, 50, 100, 200 mm/sec	25 mm/sec
3.1 Manual sensitivity?	0.25, 0.50, 1.00, 2.00 cm/mV	1.00 cm/mV
3.2 Manual lead group?	I, II, III aVR, aVL, aVF V1, V2, V3 V4, V5, V6 SP1, SP2, SP3	SP1, SP2, SP3
3.3 Manual placement?	Whole page, Upper page, Lower page	Whole page●
4.0 Rhythm speed?[4]	2.0, 5.0, 10.0, 12.5, 25.0, 50.0 mm/sec	12.5 mm/sec
4.1 Rhythm sensitivity?	0.25, 0.50, 1.00, 2.00 cm/mV	0.5 cm/mV
4.2 Rhythm lead?	I, II, III aVR, aVL, aVF V1, V2, V3 V4, V5, V6 SP1, SP2, SP3	SP1
4.3 Rhythm Save data?	1 chan / 3 chan	1 chan
4.4 Rhythm Save length?	1.5, 5.0, 10.0 sec	5.0 sec
5.0_a Frequency response? (Auto only)	0.05 - 40 Hz, 0.05 - 100 Hz	0.05 - 40 Hz●
5.0_b Frequency response? (Manual and Rhythm)	0.05 - 40 Hz, 0.05 - 100 Hz 0.5 - 40 Hz, 0.5 - 100 Hz	0.5 - 40 Hz

[3] Items 3.0 - 3.3 and 5.0_b are only used for the MANUAL recording mode, Formats 5 & 6. The user settings are listed here for the sake of completeness. The MANUAL recording mode, however, was not used for this research.

[4] Items 4.0 - 4.4 and 5.0_b are only used for the RHYTHM recording mode, Formats 7 - 9. The user settings are listed here for the sake of completeness. The RHYTHM recording mode, however, was not used for this research.

Table 3-4: Format Configuration Options (continued)

Function	Parameter	User Selections
5.2 Record header width?	Wide, Narrow	Narrow●
5.4 Patient ID required?	Yes, No	No
5.6 ID Header type?	Full, Minimum	Full●

(Hewlett-Packard Company, 1988 : 7-8)

Table 3-5: Global parameter Options

Function	Maximum Characters Accepted	User Selections
6.0 Location code?	5	00000●
6.1 Cart ID?	4	0001●
Function	**Parameter**	**User Selections**
6.6 Automatic ECG Storage?	Yes, No, Choice	No
6.7 Language?	English, French, German, Dutch, Spanish, Italian	English
7.0 Power-on Format?	Enter value 0-9	2
7.3 Power line frequency?	50 Hz, 60 Hz	60 Hz
7.4 Power on artifact filter?	Off, On	Off
7.7 ID field units?	English, Metric	English
7.9 Age ID field enabled?	Yes, No	Yes
7.10 Sex ID field enabled?	Yes, No	Yes
7.11 Height ID field enabled?	Yes, No	Yes
7.12 Weight ID field enabled?	Yes, No	Yes
7.13 BP fields enabled?	Yes, No	Yes
7.14 Race ID fields enabled?	Yes, No	Yes
7.15 Medication ID fields enabled?	Yes, No	Yes

Table 3-5: Global parameter Options (continued)

Function	Parameter	User Selections
7.16 Diagnosis ID fields enabled?	Yes, No	Yes
7.17 Criteria ID fields enabled?	Yes, No	Yes
7.18 Operator ID fields enabled?	Yes, No	Yes
7.19 Department ID field enabled?	Yes, No	Yes
7.20 Room ID field enabled?	Yes, No	Yes
7.21 Requested by ID field enabled?	Yes, No	Yes
7.22 User A ID field enabled?	Yes, No	Yes
7.23 User B ID field enabled?	Yes, No	Yes
7.24 Stat ECG ID Field enabled?	Yes, No	Yes

(Hewlett-Packard Company, 1988 : 7-15)

Table 3-6: Transmission Parameter Options

Function	Maximum Characters Accepted	User Selections
8.0 Phone #1[5]	22	
8.1 Phone #2	22	
8.2 Phone #3	22	
8.3 Phone #4	22	

[5] Fields 8.0 - 8.3 are blank since the HP4745A is not connected to any modem and thus not used for dialling into other systems.

Table 3-6: Transmission Parameter Options (continued)

Function	Parameter	User Selections
8.4 Printback enabled?	Yes, No	No
8.5 Autodial after transmit, selection?	Yes, No	Yes
8.6 Dialing type?	Pulse, Tone	Tone
8.7 Dial pause length?	2, 4, 6, 8, 10 sec	2 sec
8.8 Phone baud rate?	300, 600, 1200, 2400, 4800, 9600, 19200	1200
8.9 System baud rate?	300, 600, 1200, 2400, 4800, 9600, 19200	19200

(Hewlett-Packard Company, 1988 : 7-20)

Table 3-7: Time and Date Options

Function	Parameter	User Selections
9.0 Clock Mode (12/24)	12, 24	24
9.1 Date (dd/mm/yy)[6]	dd/mm/yy	
9.2 Time (hh:mm)[7]	hh:mm	

(Hewlett-Packard Company, 1988 : 7-22)

4.4. The ECG recording mode and format used for this study

The HP4745A is very versatile and allows for 9 preset, factory configured, recording modes (as well as user-defined recording mode) to be used. (These choices are summarised in Table 3-4.) A format is a preset combination of leads, recorded at a given speed and sensitivity. Since nearly all ECGs taken at the Department of Cardiology, Universitas Hospital are recorded using the same (standard) format, only the following recording mode was used:

[6] The current date in the memory of the HP4745A is displayed by default when the user is prompted for a new value for the date field.

[7] The current time in the memory of the HP4745A is displayed by default when the user is prompted for a new value for the time field.

Table 3-8: Report format in use at Cardiology, Universitas Hospital, Bloemfontein

Recording Mode	Format	Factory Setting
AUTO/RS	2	12-Lead + Rhythm Strip (10 seconds). The rhythm strip represents 10 seconds of acquired data. Speed for the rhythm strip is 25 mm/sec.

Please refer to Table 3-9 and Figure 3-20 for an example of an ECG recorded using the recording format described above.

Table 3-9: Items found on the ECG report (AUTO/RS Mode, Format 2)

Item	Description
A	Patient ID and demographic information.[8]
B, C, D	Auto ECGs contain three channels of data. Each channel consists of four leads.
E	Auto/RS ECGs add a fourth channel as a rhythm strip. These data are acquired immediately after the 12-lead data are obtained.
F	A 1 mV calibration pulse is printed.
G	Identification is printed above each lead.
H	A split vertical line indicates when the recording switched to the next lead.
I	Lead, speed and sensitivity for the rhythm strip are printed directly above it.
J	Location code (CART ID).
K	Frequency response (Front end filter) defaults to 40 Hz.
L	Unique sequence number assigned to ECG by cardiograph.

[8] Depending on the configuration of the HP4745A Cardiograph, different information is printed. Information includes Patient ID Number, Age and Sex, Height and Weight, Systolic and Diastolic Blood Pressure, Race, Medications, Diagnoses, two User Fields, ECG Requested By, Operator and Room Number as well as a Department Identifier. (Hewlett-Packard Company, 1988 : 4-6).

5. Summary

Myocardial activity is initiated by electrical stimulus. An electrocardiograph produces a permanent record of such myocardial electrical activity, and not of the contraction, as is often assumed. In order to perform this task, the ECG machine needs to amplify, filter and convert electric signals. It also needs to display and distribute recorded signals.

Electrodes connected to the skin provides information on the electrical activity of the various chambers of the heart. Twelve standard leads provide a simultaneous view of the myocardial electrical activity from different perspectives (angles). The resulting traces consist of various waves and segments.

A standard configuration for the HP4745A PageWriter II Cardiographs used at the Department of Cardiology, University of the Orange Free State, Bloemfontein, has been set up. These settings are needed to enable proper interpretation of the digitally acquired ECG Data.

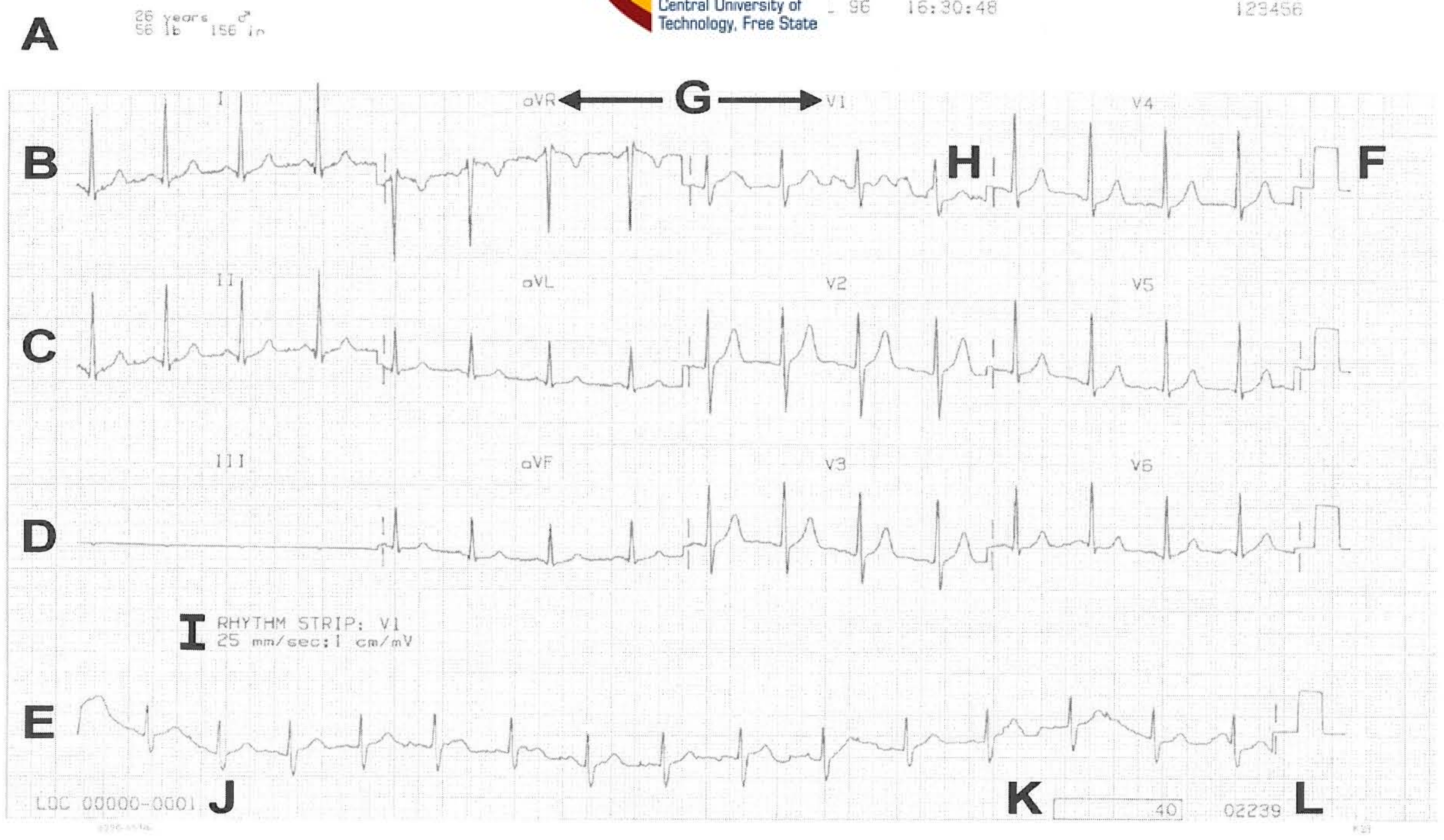


Figure 3-20: A Sample ECG

Chapter 4

Data Acquisition

1. Introduction

The ultimate goal of this study was to develop (amongst others) a software product that would allow digital storage of electrocardiogram data captured from a HP4745A PageWriter II Cardiograph. Before data can be stored, however, it needs to exist. Data were captured from the source (HP4745A) by means of some data capture procedure, and stored in the *Hearts 32* database.

When contemplating the data capture mechanism, it is often easier to model such a mechanism as a "black box" (Figure 4-1 below). Such an approach is not unique to data capture between the HP4745A PageWriter II Cardiograph and a PC, but can also be applied to other situations where data are exchanged between any other instrument and a PC.

If an interface between the instrument and the outside world is provided by the manufacturer of the instrument, it is possible to develop an interface between the PC and the instrument (connecting to the provided instrument-interface). The "black box" mentioned in the previous paragraph can thus be seen as the interface between the PC and the instrument, connecting to the interface of the instrument, as shown in Figure 4-1 below.

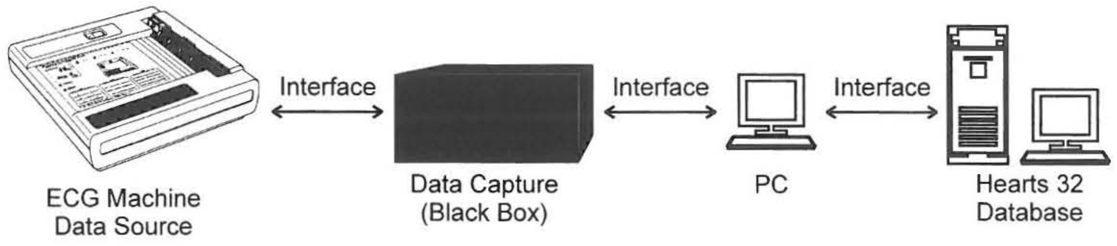


Figure 4-1: Schematic representation of data acquisition

The data acquisition module can only succeed to interface the instrument and the PC when it interfaces at both a hardware and software level, as depicted in Figure 4-2 below.

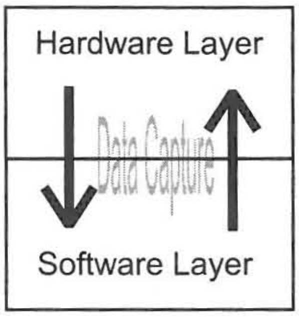


Figure 4-2: Two major components of data acquisition

The hardware level is responsible for providing the electrical connection between the interfaced equipment (instrument and PC). It is this connection that constitutes the communications link (in terms of voltages, as defined by the RS-232 serial communications protocol) between the interfaced equipment. Data transfer utilising said link is in part controlled by the processes running in the software level. (A discussion of the details of the RS-232 protocol is outside the scope of this thesis. Suffice it to mention that RS-232 communications is utilised for hardware communications. The settings of the RS-232 communications port parameters are discussed in Appendix A.)

The software level utilises the services provided by the hardware level in order to move data between the interfaced equipment, and also to manage the link. The communications protocol developed by the vendor of the instrument is implemented in the software layer as well. A detailed discussion of the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol* can be found in Appendix A.

Such a data acquisition module as mentioned above will only become truly useful when integrated with a software product such as *Hearts 32*. This chapter provides a high level discussion on the design issues of the data acquisition module developed to download and decode data from a HP4745A PageWriter II Cardiograph.

2. Design issues

During the conceptualisation of the data acquisition module, the effects of each of the following components had to be taken into account:

- RS-232 communications settings (baud rate, data bits, stop bits, parity)
- Digital transmission protocol used for downloading of data
- Internal data storage format

Although roughly the same, the communications settings used for RS-232 data communications differ for different models and makes of ECG machines. The digital transmission protocols contain proprietary information and are closely guarded by each manufacturer of ECG machines. Internal data storage formats also differ for different models and makes of ECG machines.

If a database, capable of storing digitally captured ECG data from different ECG machines is to be developed, great care should be taken to ensure that the interface between the database and the ECG machine is open ended.

In the case of the *Hearts 32* database and the HP4745A, provision has to be made to enable the *Hearts 32* database to cope with a changing environment. Preparing the HP4745A data acquisition module in such an open ended manner will:

- Allow data acquisition from the HP4745A.
- Immediately enable *Hearts 32* to communicate with future data acquisition modules that conform to the specifications developed during this research.

2.1. *Hardware independence*

In a dynamic environment, it cannot be expected that the systems in use will stay static. Systems become outdated and are replaced with others. Database applications such as *Hearts 32* should be designed in such a way that they are robust enough to handle different sources of ECG data.

It should not be necessary to change the *Hearts 32* application itself each time a different ECG machine is introduced into the system. Configuring *Hearts 32* for data acquisition from different ECG machines should be as easy as selecting the desired machine from a list.

2.2. Configurability

Just as the ECG machines that are used to acquire data are subject to change, different models of ECG machines will use different communications configuration settings. The *Hearts 32* application must make provision for setting the communications configuration of the data acquisition module.

3. Addressing the issues

The issues mentioned above have been addressed during the development of the data acquisition module. No attempt was made to recreate Hewlett Packard's Digital Transmission Protocol. Instead, a program was written to handle data from the HP4745A, supplying the correct responses (as defined by the protocol) at the correct time during a transfer. A detailed discussion of this program can be found in Chapter 8.

3.1. Hardware Independence

It is unrealistic to expect that a database application such as *Hearts 32* should inherently be capable of acquiring data from any ECG machine on the market. For this research, a data acquisition module for the HP4745A was developed. To plan for hardware independence, the data acquisition module is implemented as a Dynamic Link Library (DLL). This idea can graphically be illustrated as in Figure 4-3:

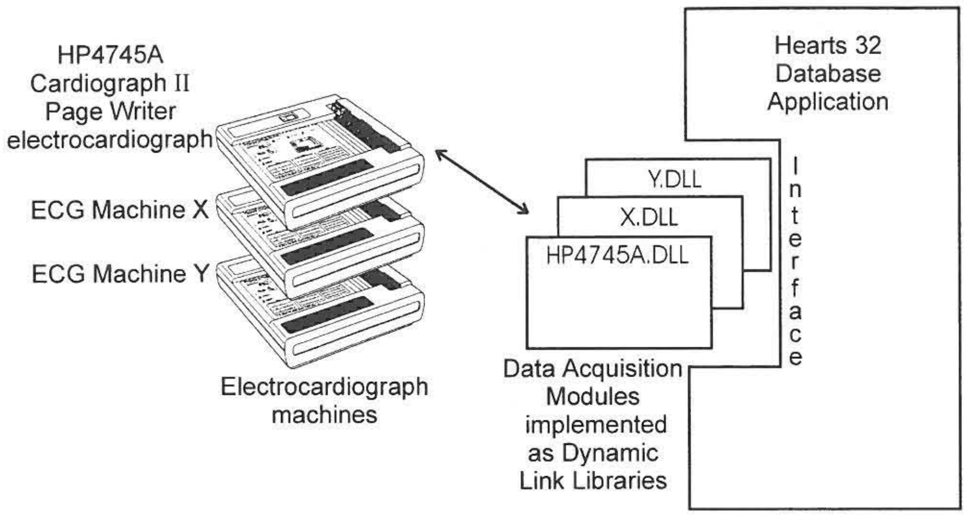


Figure 4-3: Hearts 32 and the Data Acquisition DLL

The rationale behind using DLL technology is that the data acquisition module should not be a fixed part of the *Hearts 32* database (as in Figure 4-4). The situation depicted in Figure 4-4 will lead to the following problems:

- With every new ECG machine used for digital data acquisition, the *Hearts 32* application code itself will have to be updated.
- Since it is possible that data acquisition modules for different ECG machines may be developed by parties other than the original creators of the *Hearts 32* application, this is a severely limiting factor.
- The *Hearts 32* application will become unnecessarily large (wasting critical resources such as RAM, and time to load the application) when it has to contain code for data acquisition of a variety of ECG machines (some of which may no longer be in use).

If and when another ECG machine is connected to *Hearts 32*, an appropriate data acquisition DLL for the ECG machine must be provided. The data acquisition DLL then acts as a driver for the application program (Figure 4-3). This does not differ from any other Windows® program, where the complexities of driving peripherals, for example, have been abstracted from the application programs using the peripherals.

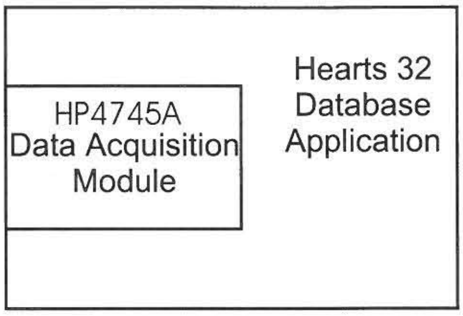


Figure 4-4: The incorrect approach (integrated data acquisition)

The function of such a data acquisition DLL would be to deliver a data set in a standard, pre-defined format[1] for inclusion in the *Hearts 32* database (Figure 4-3). This is needed because *Hearts 32* (or any other application) needs to be able to interpret the data for processing.

3.1.1.Dynamic Link Libraries

According to Pacheco & Teixeira (Pacheco & Teixeira, 1996 : 656) a dynamic link library is a program module that contains code, data or resources that can be shared among Windows® applications, as shown in Figure 4-5 below. Dynamic link libraries can also share code, data or resources among one another.

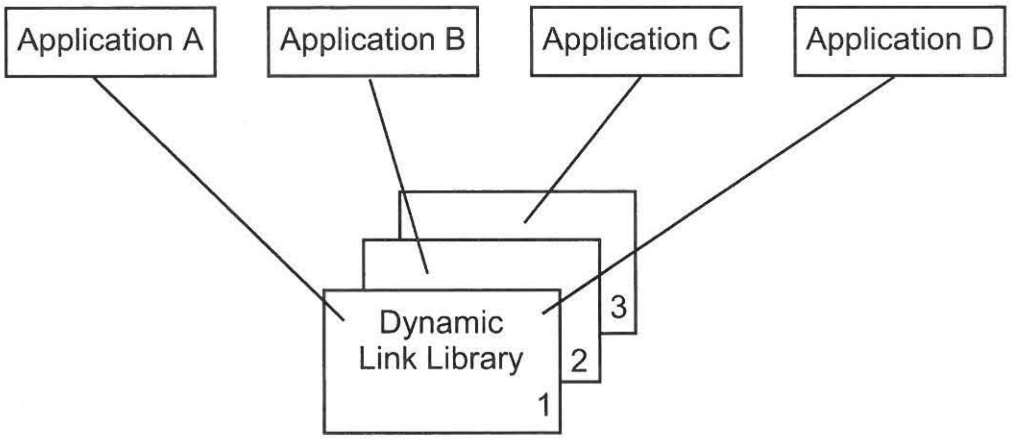


Figure 4-5: Code sharing with Dynamic Link Libraries

[1] Besides a Data Acquisition Module for the HP4745A Cardiograph II PageWriter electrocardiograph, a set of specifications enabling data acquisition from other electrocardiograph machines into the *Hearts 32* database also resulted from this study. Details of the proposed format can be found in Appendix B.

Some very real advantages can be found from implementing dynamic link libraries. These are summarised in Table 4-1:

Table 4-1: Advantages of using DLLs

1	Applications can load code to be executed at runtime, rather than having code statically linked into the executable file.
2	Multiple applications can use the same code, provided by the DLL, simultaneously.[2]
3	Because of re-use of DLLs, the programs using them require less disk space.
4	Applications become modular, allowing maintenance to be performed on sections (the DLLs) of the project without affecting the rest of the project. For example, when new peripherals are created, a set of drivers is supplied. In the case of Windows®, these are normally .DRV files (device drivers). These .DRV files are nothing else than dynamic link libraries.
5	Hiding of implementation details. To the user (application program) of the exported functions of a DLL, only the function name is visible. Exactly how the results are achieved is of no importance when using the function.

(Pacheco & Teixeira, 1996 : 656).

As with all good things, there is a downside and the disadvantages of using dynamic link libraries can be summarised as follows:

[2] An example of this is Win32®, which, according to Pacheco & Teixeira (Pacheco & Teixeira, 1996: 656), relies heavily on the files KERNEL32.DLL, USER32.DLL and GDI32.DLL. Services supplied by these dynamic link libraries include memory, process and thread management as well as graphics and user interface functions in Windows 95®.

Table 4-2: Disadvantages of using DLLs

1	If the DLL file is not available, the application using it will fail since it cannot provide the necessary services without the DLL.
2	Distribution of the application is somewhat more complicated, since supplying the executable file alone is not enough to ensure that the application will function correctly.
3	Repetitive loading of the DLL might have an impact on the application's performance.

3.1.1.1.Static linking versus dynamic linking

When writing program code, the programmer uses the keywords and operators of the programming language, together with the procedures and functions provided by the vendor of the programming language in run time library (RTL) files. Normally, the programmer also creates his/her own procedures and functions in order to augment the procedures and functions found in the RTL.

A list of references to the procedures and functions used in the program is built by the compiler during the compilation phase.

The linker has to resolve the procedures and functions used in the program, as identified during the compilation phase. Resolving means that the linker must ensure that the executable code for each procedure/function is available when the program executes.

In the case of static linking, the linker places a copy of each procedure/function into the executable program file. For example, suppose that a user-defined procedure *Power* has been created to raise *x* to the power of *y*, and placed in an Object Pascal unit file called *PowerU*. Every Object Pascal program using the *Power* procedure will contain a copy of the *Power* procedure code, statically linked into the executable file, as illustrated in Figure 4-6 below.

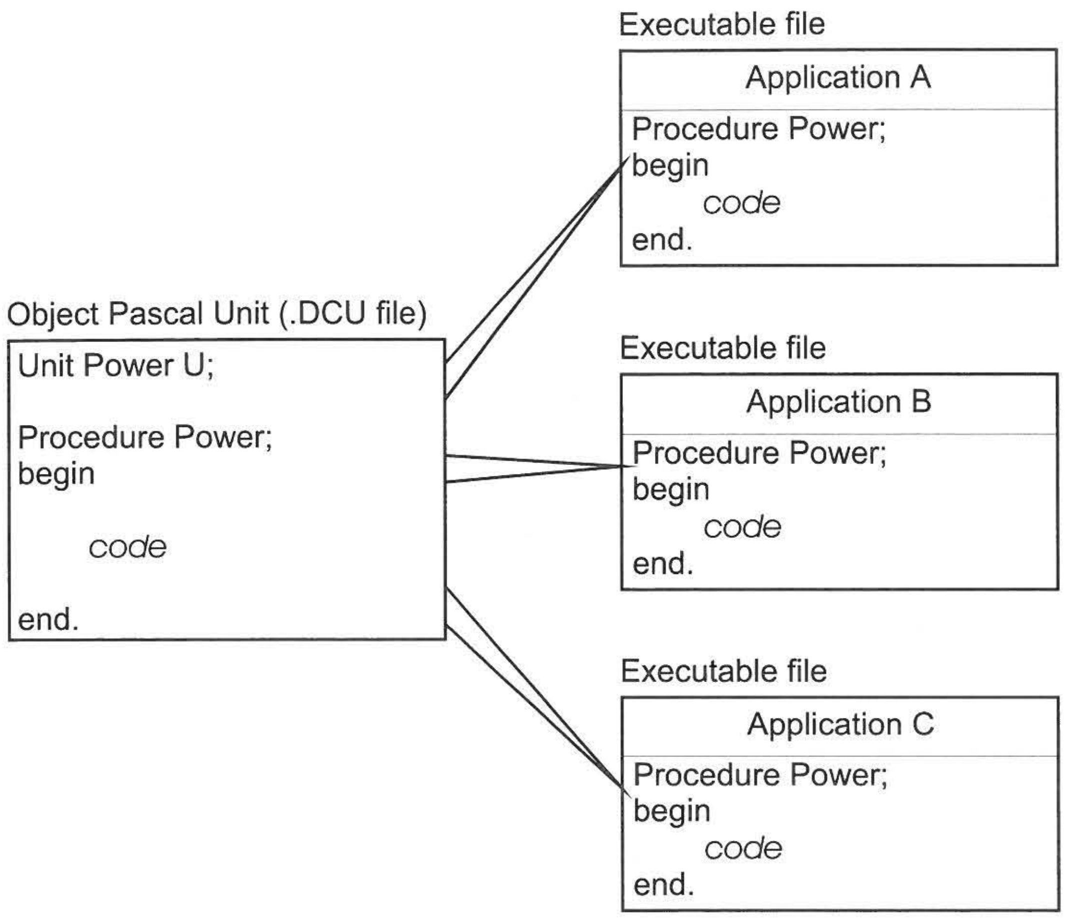


Figure 4-6: Static linking

Some problems that can be foreseen with static linking include:

- If the code contained in the procedure/function changes, all programs containing a copy of the code need to be re-compiled and re-linked. This makes maintenance more difficult. In some cases this represents a serious problem, especially if a large number of users use the product, and the executable programs are large. Imagine what would have happened if the entire Windows® software were developed as one big executable file (if this could be done!). Every time a new piece of hardware came on the market, the entire Windows® software package would need to be updated, and re-distributed!

- Bloating of the executable file (increased file size) by including a copy of the code in each and every program. This could waste a lot of disk space and load time if a large number of programs using the same procedures and functions reside on the same computer system and are very dynamically loaded and un-loaded.

With dynamic linking, the link between a call to the procedure *Power* (in our example) and the *Power* procedure itself would be resolved at runtime by using an external reference to the *Power* procedure resident in the relevant DLL. This can be illustrated as in Figure 4-7 below:

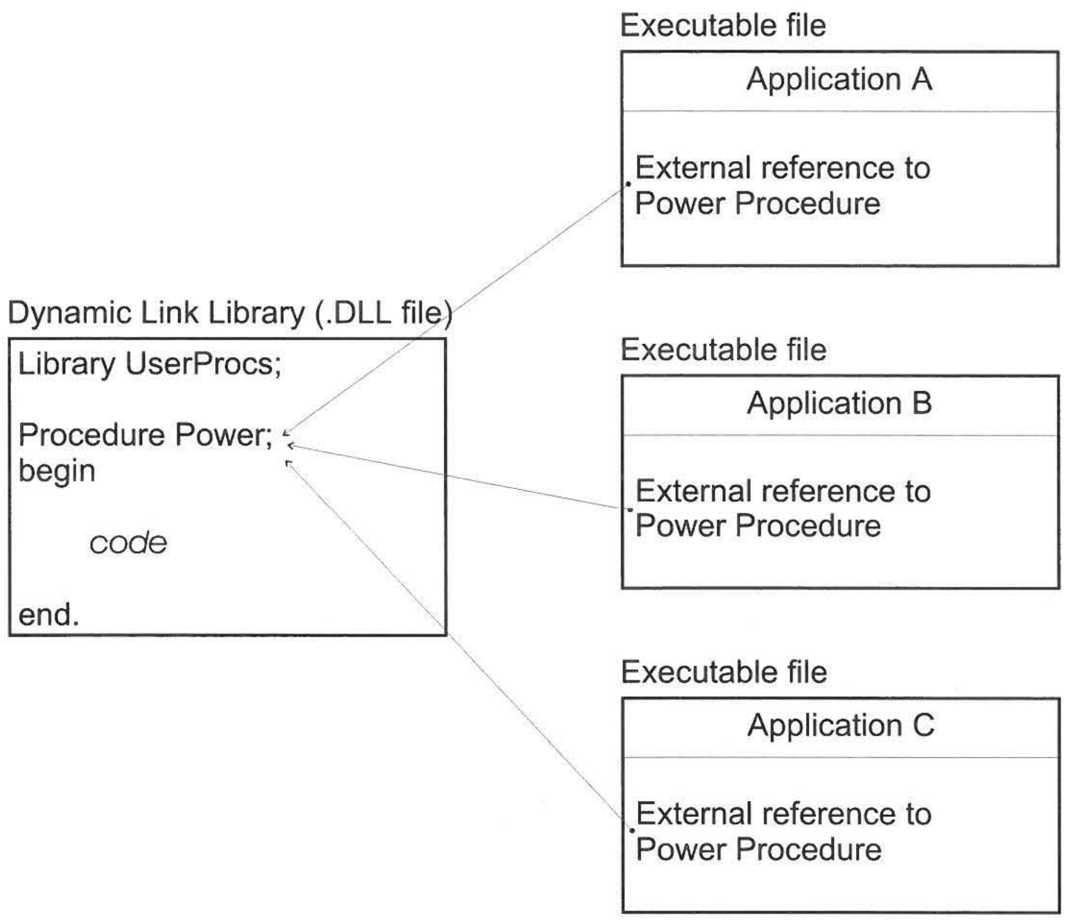

Executable file
Application A
External reference to Power Procedure
Dynamic Link Library (.DLL file)
Library UserProcs;
Procedure Power;
begin
code
end.
Executable file
Application B
External reference to Power Procedure
Executable file
Application C
External reference to Power Procedure

Figure 4-7: Dynamic linking

3.1.1.2.Implicit Loading versus Explicit Loading

When some of the functions and procedures required by the application reside in a DLL, the DLL is automatically loaded. This process is called Implicit Loading and it is implemented in Object Pascal in the following manner:

In the interface part of the unit, the procedure/function declaration is made in the normal manner. The use of the *StdCall* reserved word is necessitated to ensure that parameters are passed from right to left, enabling the Object Pascal program to interface with DLLs written in other languages such as C.[3] Since Windows® is written in C, this also enables interfacing with the Windows® DLLs.

In the implementation part of the unit, the procedure/function declaration is given, but without reference to parameters and/or return values. The *External* keyword indicates that the procedure/function is imported from an external source.

Consider the code example shown below:

Code Snippet 4-1: Implicit loading of a procedure/function from a DLL

```
  unit HeartsMain;

  interface
    function ReadDecodeCalcStoreECG( ComputerNumber : PChar ) : Integer; StdCall;

  implementation
function ReadDecodeCalcStoreECG; external 'HP4745A.DLL' name 'ReadDecodeCalcStoreECG';

end.
```

The fact that the function name (ReadDecodeCalcStoreECG) is entered in the program code as a string constant does not pose a serious problem, since it can be documented in the data acquisition module specification documentation. However, the hard coding of the DLL name (HP4745A.DLL) poses a serious problem for the application program using the DLL, when the application program needs to invoke functions from different (even unknown, as in the case of *Hearts 32*) DLLs. How can *Hearts 32* be informed about the existence of other data acquisition DLLs, and how will it load and execute such DLLs?

It is not always desirable to use implicit loading of a DLL. The following reasons illustrate the problems:

[3] It is extremely important to note that the default parameter passing convention of Object Pascal and that of C does not correspond to each other. The *register* and *pascal* conventions found in Object Pascal pass parameters from left to right, that is the leftmost parameter is evaluated and passed first and the rightmost parameter is evaluated and passed last. The *cdecl*, *stdcall* and *safecall* conventions pass parameters from right to left.

Table 4-3: Contra-indications for use of implicit DLL loading

1	If a DLL is implicitly loaded but never used during a particular execution of the application, time and memory is be wasted.
2	A DLL with a large number of routines can be quite large, occupying a lot of memory. In this instance it is better to load the DLL on demand.
3	An application loading multiple DLLs possibly does not need access to all the DLLs simultaneously. It would be better to load these DLLs as needed, thus conserving memory requirements whilst improving performance through reducing the initial load time.
4	The existence of DLLs may not be known to the application at the time of creating the application.

Sharing of the resources contained in the DLL is achieved, but implicit loading of the DLL still means that there is a strong static coupling between the calling application and the DLL itself. What is needed for the *Hearts 32* application to succeed in its quest for data acquisition, is DLL technology, but with a loose coupling between *Hearts 32* and the data acquisition DLL.

Explicit Loading refers to the loading of a specific DLL (as identified by the application) on request of that application. Such "loading on demand" offers more freedom, but also necessitates greater care. The application must test that the desired DLL exists (is accessible), has been loaded, and once loaded, determine whether the desired function or procedure can be located in the DLL.

Since explicit DLL loading does not access the DLL or the procedures and functions that the DLL contain, using literal identifiers (such as literal text strings in the program code), extra work has to be performed. A global procedure pointer[4] is declared as follows:

Code Snippet 4-2: Declaring a global procedure pointer type

```
TReadECGFunc = Function( hAppHandle : THandle; sDescription : PChar; nComPort :
Integer; nBaudRate : Integer; sBinaryFile : PChar; sASCIIFile : PChar; sECGTime :
TAr20; sECGDate : TAr20 ) : Integer : StdCall;
```

The global procedure pointer allows access to a procedure or function residing in the DLL, but from the calling module.

A variable of type *TReadECGFunc* (declared in Code Snippet 4-2 above) now needs to be created. This variable will store the address of the *ReadDecodeCalcStoreECG* function. Another variable that will access the DLL (by obtaining a handle to the DLL) needs to be declared. The declaration of these variables can be illustrated as follows:

Code Snippet 4-3: Variable Declaration for explicit DLL Loading

```
var
     ReadECGFunc  : TReadECGFunc;
     LibHandle    : Thandle;
     LoadThisDLL  : String;
```

The next step is to load the DLL. The name of the DLL in this example is contained in a string variable called *LoadThisDLL*, and the value of this variable is assumed to be assigned previously. A handle into the loaded DLL is returned, through which the contents (code and resources) of the DLL can be accessed.

Code Snippet 4-4: Explicit Loading of a DLL

```
     LibHandle := LoadLibrary( LoadThisDLL );
```

[4] The Borland® Object Pascal Language Guide describes global procedure pointers as follows: "A procedural type declared without the **of object** clause is called a global procedure pointer. A global procedure pointer can reference a global procedure or function, and is encoded as a pointer that stores the address of a global procedure or function." (Borland, 1997 : 4-18).

After successful loading of the DLL, the address of the desired procedure/function residing in the DLL must be determined. This address will be assigned to the global procedure pointer variable declared in Code Snippet 4-3, and can be illustrated as follows:

Code Snippet 4-5: Linking with the DLLs Exported functions/procedures

```
    @ReadECGFunc := GetProcAddress( LibHandle, 'ReadDecodeCalcStoreECG' );
```

The *GetProcAddress* Windows® API function takes as parameters a handle to the DLL and a string containing the name of the function/procedure of which the address must be determined. It returns this address. Since we do not want the variable *ReadECGFunc* to contain the address, but rather to point to the address, we need the @ operator.[5]

The *ReadDecodeCalcStoreECG* function is indirectly invoked *via* the *ReadECGFunc* pointer as follows:

Code Snippet 4-6: Invoking a Function via a Pointer

```
    ResValue := ReadECGFunc( Application.Handle, PChar( DLLDescription ),
                       ComPort, BaudRate, PChar( FileNameE ),
                       PChar( FileNameA ), TimeECGTaken, DateECGTaken );
```

The library handle (assigned in Code Snippet 4-4 above) needs to be freed before the code module in the main program terminates. This is done as follows:

[5] The Borland® Object Pascal Language Guide describes procedural types and the use of the @ operator as follows: "The @ operator is often used when assigning an untyped pointer value to a procedural variable. For example, the *GetProcAddress* function defined by Windows (in the *WinProcs* unit) returns the address of an exported function in a DLL as an untyped pointer value. Using the @ operator, the result of a call to *GetProcAddress* can be assigned to a procedural variable." (Borland, 1997 : 6-13).

Code Snippet 4-7: Freeing the Library Handle

```
FreeLibrary( LibHandle );
```

3.1.1.3.Making the connection

For an application to use the functions or procedures contained in a DLL, the DLL must export (make available) the names of these modules. This is accomplished by using the *exports* clause in Object Pascal. *Exports* entries can include the name of the module to be exported (Code Snippet 4-4), or it can include the word *index* followed by an integer constant (Code Snippet 4-3). "This method is called *importing by ordinal*". (Pacheco & Teixeira, 1996 : 663).

Code Snippet 4-8: Importing by ordinal

```
function ReadDecodeCalcStoreECG; external 'HP4745A.DLL' index 1;
```

Code Snippet 4-9: Importing by name

```
function ReadDecodeCalcStoreECG; external 'HP4745A.DLL' name 'ReadDecodeCalcStoreECG';
```

The user (application) of the DLL can invoke the desired module by using the exported name or the index entry. It is interesting to note that when using the name, there is a slight performance penalty to be paid, since the module's name has to be looked up in the DLL's name table. Pacheco & Teixeira (Pacheco & Teixeira, 1996 : 664) suggest that the "importing by name" method be used, since the "importing by ordinal method" is too cumbersome to work with.

The "importing by name" method has been chosen for the *Hearts 32* application.

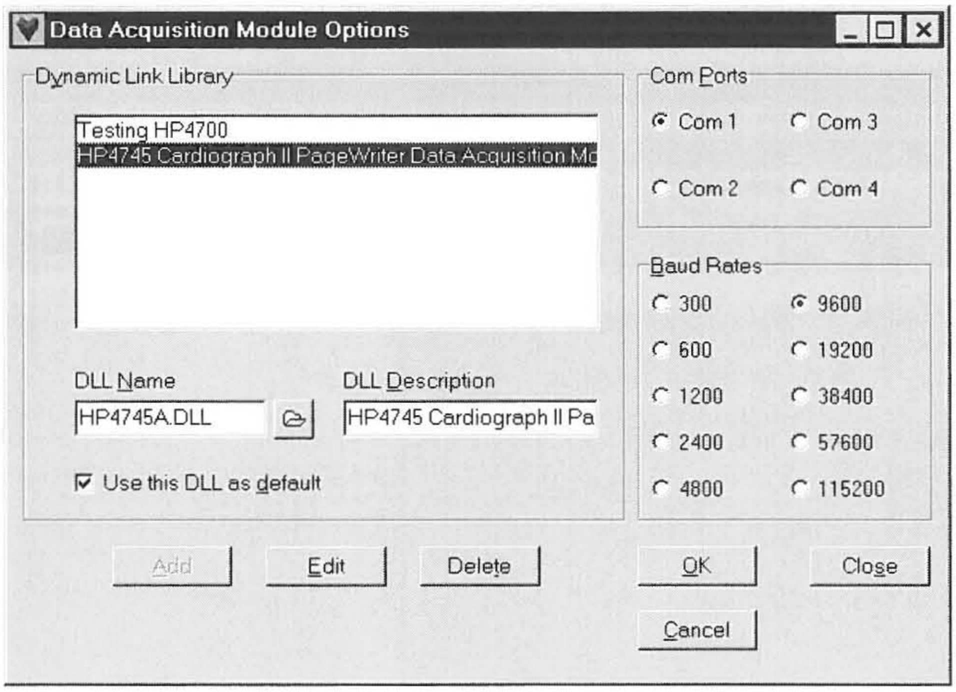
3.2. Configurability

It is all good and well to have the main application (*Hearts 32* in this case) on the one side, and the different, separate data acquisition modules implemented as dynamic link libraries on the other side, but some mechanism is needed in order to connect these different pieces of software.

Simply placing the DLL files in a certain location and having the application search for them, is in itself not enough. The reason for this is that different ECG machines may have different communications port settings.

The following basic items (as found in Figure 4-8) are recorded for each data acquisition module:

- DLL Name.
- DLL Description.
- Whether this DLL is to be used as default or not.
- Communications port where ECG machine is connected to the PC.
- Baud rate at which the link between the ECG machine and the PC operates.



Dynamic Link Library
Testing HP4700
HP4745 Cardiograph II PageWriter Data Acquisition Mo

DLL Name: HP4745A.DLL
DLL Description: HP4745 Cardiograph II Pa
[✓] Use this DLL as default

Com Ports: [✓ Com 1] Com 2 Com 3 Com 4

Baud Rates: 300 600 1200 2400 4800 [✓ 9600] 19200 38400 57600 115200

Figure 4-8 Data Acquisition Module Configuration Dialogue Box

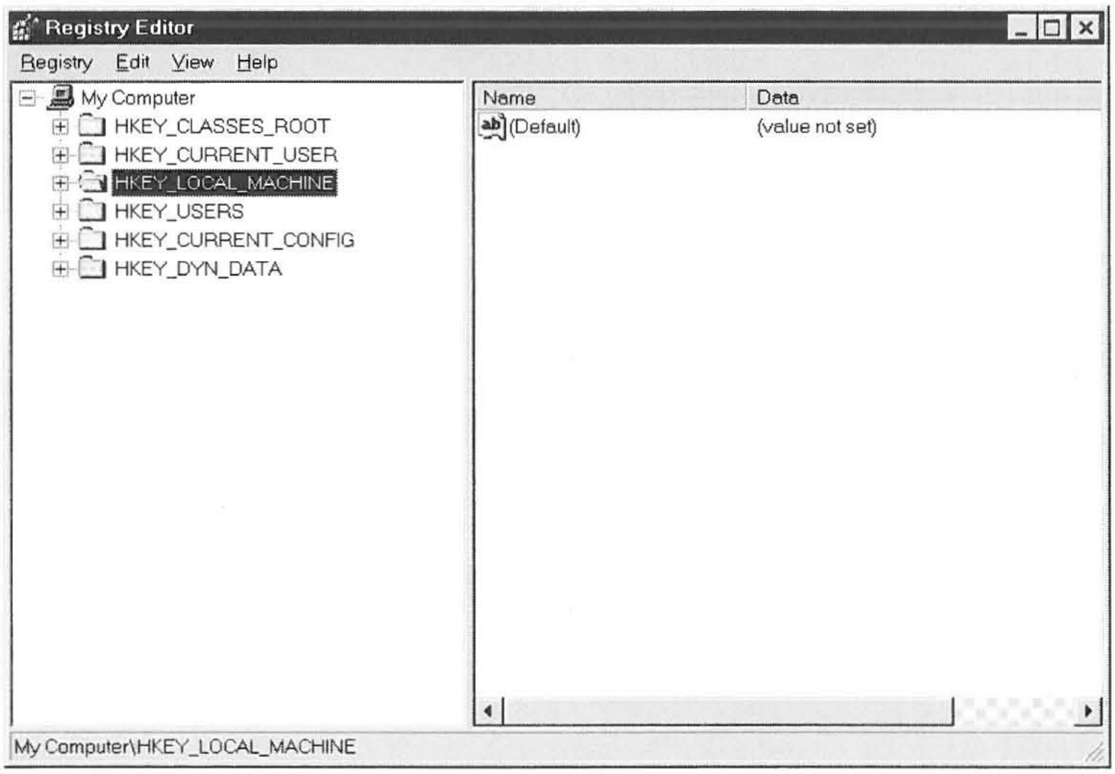
The idea behind this kind of link is that the *Hearts 32* application can easily be informed of available data acquisition modules.[6] It is also possible to make some very basic (but critical) changes to the communications settings *via* this interface. Other

[6] By simply clicking with the mouse on the folder speed button (located next to the edit dialogue box for DLL Name), the user may navigate the hard disk in order to easily locate the dynamic link library searched for.

communications settings (such as buffer sizes and flow control options) should be set in the data acquisition module itself and not be exposed to tampering by operators.

The question now arises: Where is this information stored? In the Registry. Cantù describes the Registry as "a hierarchical database of information about the computer and software configuration, and the user preferences." (Cantù, 1997 : 1173). Thurrot *et al* describes the Registry in much the same way. (Thurrot *et al*, 1997 : 57). Thurrot *et al* also mentions that the function of the Registry has previously haphazardly been performed by INI files in earlier versions of Windows®. The Registry is more sophisticated than INI files due to its organisational structure. (Thurrot *et al*, 1997 : 57).

The Windows® 95 Registry is based on six top-level keys, as shown in Figure 4-9:



My Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG
HKEY_DYN_DATA

(Default) (value not set)

My Computer\HKEY_LOCAL_MACHINE

Figure 4-9: The Windows® 95 Registry

It is possible to examine (view and modify!) the Registry by using a program such as RegEdit.EXE, or other custom designed software. An important word of warning at this point: Be extremely careful. According to Cantù, "The importance of the registry should not be underestimated. The Registry holds crucial information about the system hardware configuration, Control Panel settings, OLE servers, and even statistics about the machine." (Cantù, 1997 : 1174). If the Registry is damaged, a complete reload of Windows® 95 might be needed.

Windows® provides a set of API functions allowing interaction with the Registry. In order to use the Registry, the correct Key (folder) has to be opened. The correct Subkey (subfolder) then has to be opened, allowing access to values (items). Delphi™ supplies an interface to the Registry through the two Visual Component Library (VCL) classes *TRegistry* and *TRegIniFile*. This facilitates working with the Registry dramatically.

When adding entries to the Registry, it is recommended that the items be placed under the *Software* subkey, and even under a company name, software product and version number (Cantù, 1997 : 1175). This is the default behaviour of the InstallShield Express software installation program which is supplied with Delphi™.[7] In the *Hearts 32* application the complete key for application information is defined as follows:

Code Snippet 4-10: Complete Registry Key used in the Hearts 32 application

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Double Precision Computing Services CC\Hearts 32\1.0\Data
Acquisition Modules
```

This is graphically illustrated in Figure 4-10:[8]

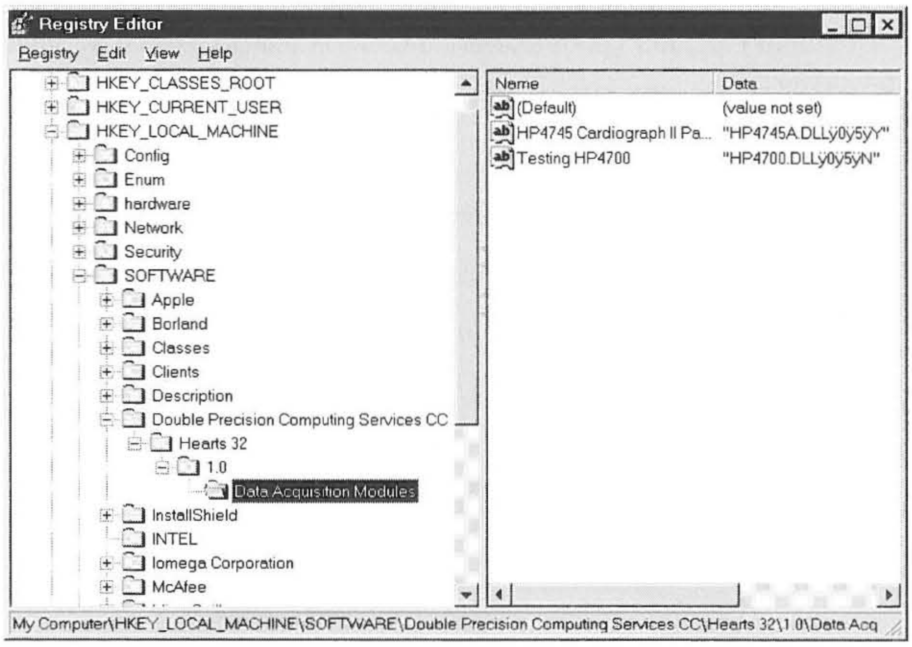
From Figure 4-10 it can be seen that the individual items as listed in Figure 4-8 (page 64) are stored as values under the *Data Acquisition Module* subkey. The item

[7] The InstallShield Express installation software is a product in its own right, and seems to have become the *de facto* standard for installing of programs in the Windows® environment. A cut down version of InstallShield Express is supplied with Borland® Delphi™.

[8] Although Figure 4-10 displays the key as starting with "My Computer", this part is not supplied when gaining programmatic access to the Registry. The root of the key, therefore, is shown as \.

DLL Description serves as the value name, while the value itself is comprised of the rest of the data as shown in the dialogue box in Figure 4-8.[9]

A detailed discussion of the programming needed for access to the Registry, can be found in Chapter 8.



```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
  Config
  Enum
  hardware
  Network
  Security
  SOFTWARE
    Apple
    Borland
    Classes
    Clients
    Description
    Double Precision Computing Services CC
      Hearts 32
        1.0
          Data Acquisition Modules
    InstallShield
    INTEL
    Iomega Corporation
    McAfee
```

Name	Data
(Default)	(value not set)
HP4745 Cardiograph II Pa...	"HP4745A.DLLÿ0ÿ5ÿY"
Testing HP4700	"HP4700.DLLÿ0ÿ5ÿN"

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Double Precision Computing Services CC\Hearts 32\1.0\Data Acq

Figure 4-10: An example of the Registry entries for Hearts 32

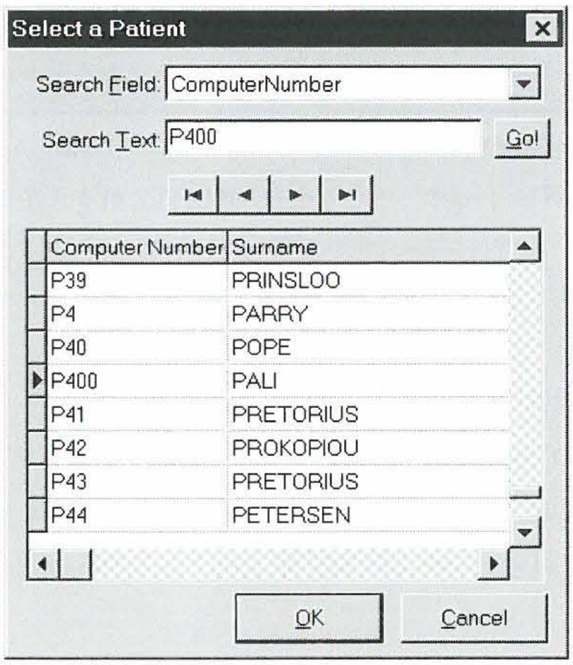
3.3. Data acquisition

The data acquisition process can be outlined as follows:

1. The correct patient record is identified and selected in the *Hearts 32* database (Figure 4-11).
2. The ECG tab is selected, opening the dialogue box for ECG data acquisition (Figure 4-12).

[9] The value for each of the communications configuration items has been concatenated into one long string in order to ease the management of these items in the Registry. Each value in the string has been separated from the other by using ASCII 255 (represented as a ÿ in Figure 4-10). This makes tokenising of the string possible when the individual values need to be retrieved again.

3. When the user activates the data acquisition procedure, *Hearts 32* scans through the Registry entries, searching for the data acquisition DLL that has been set as the default DLL.
3. *Hearts 32* now proceeds by attempting to load the default data acquisition DLL, and invoking the data acquisition function located in the DLL.
4. The name of the temporary file in which the data are stored during processing is supplied by *Hearts 32*, and is based on the computer number (an internal identifier which is used at the Department of Cardiology, Universitas Hospital, and uniquely identifies a patient).
5. Once the data have been acquired from the ECG machine, the data acquisition DLL will decode and calculate (reconstruct) the numeric values.
6. As soon as control has been returned to *Hearts 32*, the reconstructed ASCII file will be compressed and entered into the ECG table of the *Hearts 32* database.

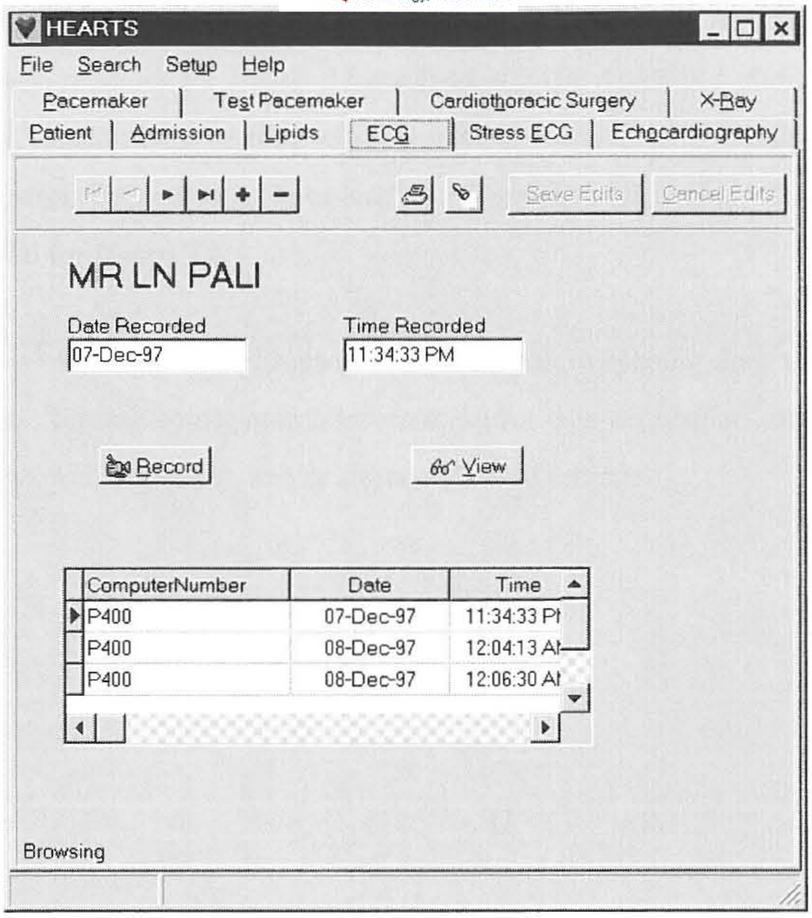

Select a Patient

Search Field: ComputerNumber

Search Text: P400

Computer Number	Surname
P39	PRINSLOO
P4	PARRY
P40	POPE
P400	PALI
P41	PRETORIUS
P42	PROKOPIOU
P43	PRETORIUS
P44	PETERSEN

Figure 4-11: Patient selection dialogue box in Hearts 32



MR LN PALI

Date Recorded: 07-Dec-97

Time Recorded: 11:34:33 PM

ComputerNumber	Date	Time
P400	07-Dec-97	11:34:33 P
P400	08-Dec-97	12:04:13 A
P400	08-Dec-97	12:06:30 A

Browsing

Figure 4-12: ECG dialogue box in Hearts 32

Details of the programming involved to enable these steps can be found in Chapter 7.

4. Summary

Data acquisition can be modelled as a "black box", consisting of a hardware and software layer. In the case of *Hearts 32*, the software layer of the data acquisition module will contain the functionality that will allow it to communicate with a HP4745A ECG machine and also to decode and calculate the ECG data set.

In order to keep *Hearts 32* flexible and responsive to a changing environment, data acquisition modules are implemented as dynamic link libraries (DLLs). DLLs have some distinct advantages over static linking of code. These include run-time loading of code, (simultaneous) code sharing, modular design and hiding of implementation details.

Implicitly loaded DLLs are connected with the main application at design time. Since data acquisition modules for *Hearts 32* can (and will) be developed after the creation of *Hearts 32*, implicit DLL loading will not suffice. Explicit DLL loading on request of the application (*Hearts 32*) allows loading of specified DLL files. This is exactly what is needed for *Hearts 32*.

The Windows® 95 Registry performs an important task in keeping track of the system configuration. Storing configuration information for data acquisition modules in the registry allows well organised, secure storage of these settings.

Chapter 5

Database systems: a brief overview

1. Introduction

According to Brookes (Brookes *et al*, 1982 : 185-186), the following drawbacks are associated with file based systems:

- Difficulties accessing data collected for a particular application, to satisfy the needs of a different application or *ad hoc* requests. This is caused by inconsistencies in data-storage formats, early aggregation of data with resulting loss of detail and poor data collection timing.
- Problems with the association of data which relate to the same entity, but are stored in different files. This is caused by inconsistent coding systems, incompatibility of storage media and most significantly, the inability of the system to access data without knowing the key to the desired record.
- Duplication of data in different files (to enable processing by multiple applications that cannot be integrated) leads to excessive costs in terms of storage and maintenance of code. An example is the storage of the same address information for the same person in more than one file.
- Changes made to the files of installed systems tend to be high, due to the inflexibility generated by conventional file structures. The tight coupling between the application programs and the file structures also contributes to these high costs.
- Management and control over data resources while safeguarding data integrity, together with fast, secure access to data, is a difficult task.

(Brookes *et al*, 1982 : 185 - 186)

These difficulties can largely be overcome by storing data in a database rather than in files. Accordingly, a database will be used to store the data of *Hearts 32*.

2. Choice of database management system

The database management systems available for the desktop PC (such as dBASE®, FoxPro and even Paradox®) have, up to recently, been nothing more than glorified file systems. Possibly the most important reason for this was the fact that the computers themselves were not powerful enough to support complex database operations. This situation has changed drastically with the advent of powerful desktop PCs which are in use today.

A first choice would have been to implement the database using Client/Server technology, and a database management system such as Oracle or InterBase. These DBMSs offer industrial strength solutions to database problems.

Table 5-1: Advantages of Client/Server database technology

1	Provides a cost-effective solution for many companies as an alternative to mainframe solutions.
2	Allow departmental access to data, allowing departments to process only the part of the business for which they are responsible.
3	Enforce data integrity rules for the entire database.
4	Provide better division of labour between the client and the server (each performs tasks for which it is best suited).
5	Provide the ability to use the advanced data integrity capabilities provided by most database servers.
6	Lower network traffic as subsets of data are returned to the client, as opposed to entire tables, as is the case with desktop databases.
7	Provides backup and recovery capabilities, while the database is on-line.

(Pacheco & Teixeira, 1996 : 838 - 839, Jensen *et al*, 1996 : 344).

According to Jensen *et al*, the client/server solution is not the correct solution for every application.

Table 5-2: Disadvantages of Client/Server database technology

1	Administering an SQL database server typically requires more time and effort than does a file server database. A database administrator is required for SQL servers. The administrator grants access privileges and performs system maintenance, among other duties.
2	Database servers require a hardware investment beyond what is required for file server system. The additional expense of adding a database server to a system can be very high.
3	The technology investment required to implement a database server is greater than that for a file server. For example, developers trained in delivering client/server systems often command higher salaries than do those building file server systems.
4	When you move between SQL servers, the different dialects can create migration problems.
5	Changing table structures, validation rules, and indexes can be a substantial undertaking in a client/server environment. Similar changes in a file server-based system are often less complex.
6	Data refreshing on client screens is not automatic in a client/server environment. In some file server-based systems, this refresh can be set up to occur automatically.
7	Table and record locking differs between the various servers. Some applications need record-level locking whereas high transaction-oriented applications do not. If record-level locking is necessary, programmatic schemes have to be implemented in some servers.

(Jensen *et al*, 1996 : 344 - 345).

The biggest problem, once again, is the price of these Relational Database Management Systems (RDBMSs). (Oracle Workgroup currently costs around R 2,000.00 per seat, academically priced. At 15 users, this represents an initial investment of nearly R 30,000.00. This figure does not take into account the upgrade to the existing file server that might be necessary. It also does not take into account the yearly licensing fee for the Oracle RDBMS software.) The Department of Cardiology, Universitas Hospital, Bloemfontein, cannot currently afford to spend money on the purchase of such a piece of software. Instead it has been decided that (at least for the initial phase) a desktop RDBMS included with Delphi™, will be used.

Table level 7 of the Paradox® driver supports validity checking, table lookup, referential integrity, extended field types and more. The lack of these features in the dBASE® III driver has necessitated the database change. Implementing the *Hearts 32* database using the functionality of the Paradox® driver, will ease migration to a true SQL database management system in future.

2.1. The Paradox® database management system

The Paradox® driver supplied with Borland® Delphi™ Professional Edition version 3 supports the following:

2.1.1. Validity Checks

In Paradox® tables, validity checks are rules imposed on a field to ensure that the data entered in the field meet certain requirements. The way in which a validity check is defined determines what can be entered in a field. Database Desktop provides five kinds of validity checks:

Table 5-3: Paradox® Validity Checks

Validity check	Meaning
Required Field	Every record in the table must have a value in this field.
Minimum Value	The values entered in this field must be equal to or greater than the specified minimum.
Maximum Value	The values entered in this field must be less than or equal to the specified maximum.
Default Value	This value will be entered into the field automatically, if no other value is entered.
Picture	A character string that acts as a template for the values that can be entered into the field.

Validity checks for a Paradox® table are saved in a file with the table's name and a .VAL file extension.

2.1.2. Table lookup

The table lookup feature makes it possible to refer to another table to look up acceptable values for a field. Valid values are then automatically copied from the lookup table into the primary table.

Table lookup is primarily a data entry tool. It is provided to help enter data that already exist in another table. To establish a more powerful tie between two tables, a referential integrity relationship should be defined. While table lookup ensures that data are copied accurately from one table to another, referential integrity ensures that the ties between like data in separate tables cannot be broken.

2.1.3. Secondary indexes

A secondary index is a field or group of fields that defines an alternate sort order for the table. Paradox® tables can have more than one secondary index. It is also possible to create composite secondary indexes by combining two or more fields.

Fields of type memo, formatted memo, binary (BLOB), OLE, graphic, logical or bytes fields cannot be used to create primary or secondary indexes.

Paradox® tables have these options for secondary indexes: Composite, Unique, Case-sensitive, Maintained, and Ascending/Descending.

Secondary indexes are used to link Paradox® tables and also to speed up search and locate operations.

2.1.4. Referential Integrity

Referential integrity means that a field or group of fields in one table (the "child" table) must refer to the key of another table (the "parent" table). Only values that exist in the "parent" table's key are valid values for the specified field(s) of the "child" table.

2.1.5. Password security

To ensure that a Paradox® table is protected from access by unauthorised users, a password can be provided. This is especially important in a multi-user environment. Not only can a password be established for the table as a whole, but specific rights to the table or individual fields can be assigned.

The master password controls access to an entire file. Auxiliary passwords provide different levels of access privileges for different users in a group.

2.1.6. Table language driver

A table's language driver determines the table's sort order and available character set.

2.1.7. Paradox® field types

Table 5-4: Valid Paradox® field types and sizes

Symbol	Size	Type	Comments
A	1 - 255	Alpha	Store string values.
N		Number	-10^{307} to 10^{308} with 15 significant digits.
$		Money	
S		Short	Integers in the range -32,767 to 32,767.
I		Long Integer	32-bit signed integers in the range -2147483648 to 2147483647 (plus or minus 2 to the 31st).
#	0 - 32*	BCD	Binary Coded Decimal.
D		Date	Valid dates from January 1, 9999 BC to December 31, 9999 AD. Database Desktop correctly handles leap years and leap centuries and checks all dates for validity.
T		Time	Time of day, stored in milliseconds since midnight, limited to 24 hours.

* Number of digits after the decimal point.

Table 5-4: Valid Paradox® field types and sizes (continued)

Symbol	Size	Type	Comments
@		Timestamp	Timestamp fields contain both time and date values.
M	1 - 240**	Memo	Use memo fields for text strings that are too long to store in an alpha field. Memo fields can be virtually any length. The size value assigned refers to the amount of the memo Database Desktop stores in the table. This can be from 1 to 240 characters. The rest of the memo is stored in a .MB file. The amount of data a memo field contains is limited only by the disk space available on the system.
F	0 - 240**	Formatted Memo	Same as Memo fields, except that text can be formatted.
G	0 - 240***	Graphic	Can store graphics files, such as scanned images (.BMP, .GIF, .TIF, .PCX and .EPS files for example)
O	0 - 240***	OLE	Use the OLE field to store different kinds of data, such as images, sound and documents.
L		Logical	Contains values that represent "True" or "False".
+		Auto-increment	Paradox® auto-increment fields contain long integer, read-only values. Database Desktop begins with the number 1 and adds one number for each record in the table. Deleting a record does not change the field values of other records.

** Memo and formatted memo fields can be virtually any length. The value you specify in the Create Table dialog box refers to the amount of the memo DataBase Desktop stores in the table (1 to 240 characters for memos and 0 to 240 characters for formatted memos). The entire memo is stored outside the table. For example, if you assign a size value of 45 to the field, DataBase Desktop stores the first 45 characters in the table. It stores the whole memo field in another file (with the extension .MB) and retrieves it as you scroll through the records of the table.

*** Optional.

Table 5-4: Valid Paradox® field types and sizes (continued)

Symbol	Size	Type	Comments
B	0 - 240***	Binary	Binary fields should be used only by advanced users who need to work with data that Database Desktop cannot interpret. Database Desktop cannot display or interpret binary fields.
Y	1 - 255	Bytes	Bytes fields should be used only by advanced users who need to work with data that Database Desktop cannot interpret. A common use of a bytes field is to store bar codes or magnetic strips.

2.2. The use of BLOB fields

A BLOB field is a field in a table that holds a reference to a binary large object (BLOB). BLOB fields can also be described as database fields that contain data of arbitrary length. Unlike binary fields (in Paradox®), BLOB fields do not store the binary data directly in the database table. Instead, the field in the physical database table contains a reference to a separate file that contains the individual BLOB value for the field. In Paradox®, BLOB fields are stored outside of the primary table file (.DB file) in a .MB file.

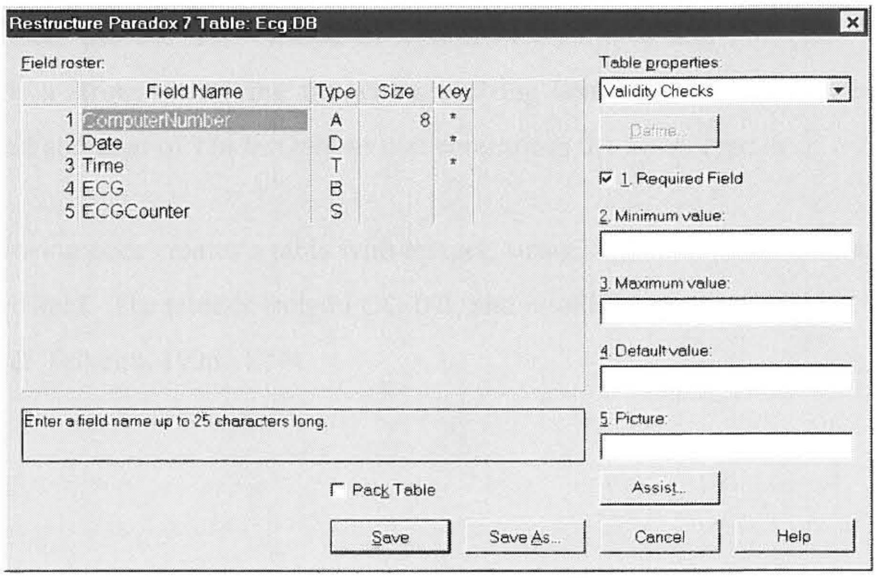
The use of a BLOB field is perfect for storing ECG data, since the size of the ECG data set is not known in advance. An average size for these data sets has been determined (Table 6-11 on page 105 in Chapter 6) for the HP4745A PageWriter II Cardiograph.

From previous experience with the storing of a large number of small files on the file server at the Department of Cardiology, Universitas Hospital, it is known that these files cause problems on the Novell® Netware® 4.1 file server. These problems surface when the contents of the hard disks is backed up using the Arcserve 6.0 backup software from Cheyenne. It appears that the backing up of one large file happens much faster than the backing up of a number of smaller files with the same total file size as that of said large file. This problem was confirmed with the support personnel at the computer centre of the University of the Orange Free State.

Another problem with the storage of these small files was identified when the existing Novell® Netware® 4.1 file server hardware was upgraded. The migration of the data contained on the hard disk of the server took nearly two days to complete, while the software used for migration predicted a migration time of about 3 hours. The problem can once again be attributed to the large number of small files stored on the file server.[1]

3. Table creation

The DataBase Desktop software that is supplied with Delphi™ is a useful tool for creating and manipulating database tables interactively (Figure 5-1). While this is fine for just creating tables, it does, not however, solve the problem of documenting the structure of the database. DataBase Desktop does not even offer the choice of printing the structure of a table!



Restructure Paradox 7 Table: Ecg.DB

Field roster:

	Field Name	Type	Size	Key
1	ComputerNumber	A	8	*
2	Date	D		*
3	Time	T		*
4	ECG	B		
5	ECGCounter	S		

Enter a field name up to 25 characters long.

Table properties: Validity Checks

☑ 1. Required Field

2. Minimum value:

3. Maximum value:

4. Default value:

5. Picture:

☐ Pack Table

Figure 5-1: Table creation with the Borland® DataBase Desktop

[1] These files include roughly 12,000 patient report files (MS Word® files), as well as digitally captured ECG data files. The plan is to store all these files in BLOB fields in future.

Jensen *et al* describes the use of the *CreateTable* method of the *TTable*[2] object in order to create tables and indexes at runtime. (Jensen *et al*, 1996 : 224 - 231). Basically all that is needed is that the field names, their data types and sizes as well as a flag to indicate whether the field is required or not, be specified. If indexes are to be created, the name of the index, the field name which is indexed as well as index options such as case sensitivity and uniqueness are specified. All that remains is to call the *CreateTable* method.

Pacheco & Teixeira (Pacheco & Teixeira, 1996 : 813 - 814) summarise the process as follows:

1. Create an instance of a *TTable*.
2. Set the *DatabaseName* property of the table to a directory or existing alias.
3. Give the table a unique name in the *TableName* property.
4. Set the *TableType* property to indicate what type of table you want to create (Paradox® or dBASE®).
5. Use *TTable.FieldDefs' Add* method to add fields to the table. Parameters include field name, field type, size of the field and a boolean parameter indicating whether or not the field is required.
6. If needed, use the *Add* method of *TTable.IndexDefs* to add indexes. Parameters include a string identifying the index, a string that matches the field name to be indexed and a set of *TIndexOptions* that determines the index-type.

"The following code creates a table with integer, string, and float fields with an index on the integer field. The table is called FOO.DB, and it will live in the C:\TEMP directory." (Pacheco & Teixeira, 1996 : 814).

2 *TTable* is used to access data in a single database table using the Borland® Database Engine (BDE). *TTable* provides direct access to every record and field in an underlying database table, whether it is from Paradox®, dBASE®, Access, FoxPro, an ODBC-compliant database, or an SQL database on a remote server, such as InterBase, Oracle, Sybase, MS-SQL Server, Informix, or DB2.

Code Snippet 5-1: Table creation via the TTable.CreateTable method

```
begin
   with TTable.Create( Self ) do begin       // Create TTable object
      DatabaseName := 'c:\temp';             // Point to directory or alias
      TableName := 'FOO';                    // Give table a name
      TableType := ttParadox;                // Make a Paradox table
      with FieldDefs do begin
         Add( 'Age', ftInteger, 0, True );   // Add an integer field
         Add( 'Name', ftString, 25, False ); // Add a string field
         Add( 'Weight', ftFloat, 0, False ); // Add a floating-point field
      end;
      { Create a primary index on the Age field... }
      IndexDefs.Add( '', 'Age', [ ixPrimary, ixUnique ] );
      CreateTable;                           // Create the table
   end;
end;
```

At a first glance this technique seems to solve the problem of programmatic table creation. For simple tables it will suffice. However, when using advanced features found in, for example, the Paradox® and InterBase drivers, this method of table creation is not sufficient.

Pacheco & Teixeira (Pacheco & Teixeira, 1996 : 888 - 900) describe the problem identified in the previous paragraph as follows: "...there are a number of capabilities provided by the Borland Database Engine (BDE) that are not surfaced by Delphi's data-access components. Because Delphi tries to maintain an interface that is database-independent, database-specific features provided by the BDE are generally the types of things for which Delphi doesn't provide." (Pacheco & Teixeira, 1996 : 888).

Fortunately Borland® supplied a set of API functions with which a program can interface directly (on a low level) with the Borland® Database Engine (BDE). It is through this mechanism that the programmatic creation of Paradox® tables, complete with validity checks and referential integrity constraints, is made possible.

It fairly quickly became apparent that the use of these low-level API functions would not be too easy. The descriptions and code examples found in the Delphi™ help files were not really helping either. A search on the Internet for the topic of low level table creation *via* the BDE API yielded only one positive result. This was in the form of a program that can analyse an existing table structure (created with a tool such as the DataBase Desktop) and generate an Object Pascal program which could, in turn, create the table. From the documentation it is apparent that the initial idea behind the

SCANNER[3] software was to create an Object Pascal unit which could be included in a software product, to programmatically create the necessary tables the first time that the program was run. This would eliminate the need for table distribution.

The output of the SCANNER software (a valid Object Pascal source code file) was adapted for use as part of *Hearts 32*. The code defines procedures for the creation of fields, indexes, validity checks, referential integrity constraints as well as table creation. Creating the tables is as easy as listing the information for each item (fields, indexes, validity checks, referential integrity constraints) in array format and calling the table creation procedure. The SCANNER software already prepares such a listing, saving a lot of work. This is illustrated as follows:

Code Snippet 5-2: Table creation using procedures from the SCANNER code

```
      ECG:
      begin

         dbDatabase.Params.Add( 'PATH=D:\HEARTS.32\DATABASE\' );
         dbDatabase.Connected := True;
         Check( DbiGetDirectory( dbDatabase.Handle, False, szDirectory ) );

         DefField( 'ComputerNumber', fldPDXCHAR,       0, 0, 8, 0 );
         DefField( 'Date',           fldPDXDATE,       0, 1, 1, 0 );
         DefField( 'Time',           fldPDXTIME,       0, 2, 1, 0 );
         DefField( 'ECG',            fldPDXBINARYBLOB, 0, 3, 0, 0 );
         DefField( 'ECGCounter',     fldINT16,         0, 4, 1, 0 );

         DefIndex( '', '', '', '', '', [ 1, 2, 3 ],
                   0, 0, 3, 16, 0, 2048, 1, True, True, False, True,
                   False, False, False, False );

         DefValCheck( 0, 1, [ 0 ], [ 0 ], [ 0 ],
                      True, False, False, False, '', '', lkupNONE );

         DefValCheck( 1, 2, [ 0, 0, 0, 128 ], [ 0, 0, 0, 128 ], [ 0, 0, 0, 128 ],
                      True, False, False, False, '', '', lkupNONE );

         DefRefInt( 0, 1, 1, [ 1 ], [ 1 ], 'refECGPatient', 'Patient.DB',
                    rintDEPENDENT, rintCASCADE, rintRESTRICT );

         DefTable( 'ECG.DB', 'PARADOX', '', 5, 1, 5, 1 );

         Check( DBICreateTable( dbDatabase.Handle, True, TableDesc ) );

end;
```

Admittedly, the heavy use of parameters in the procedure calls tends to clutter the code and makes it difficult to understand the meaning of each parameter. Some study of the procedures used is necessary before the parameters can easily be understood.

[3] All attempts to contact the author of the SCANNER software in order to pass credit failed. Only an out-dated e-mail address for the author is supplied with the software!

A code example of some of the procedures that perform the actual work is shown below:

The data types mentioned below are all defined by the BDE API.

Code Snippet 5-3: Global variables used for BDE API calls to create tables at runtime

```
var
   szDirectory   : DBIPATH;
   TableDesc     : CRTblDesc;
   FieldsDesc    : array[ 0..80 ]  of FLDDesc;
   RefIntegOp    : array[ 0..20 ]  of CROpType;
   RefInteg      : array[ 0..20 ]  of RINTDesc;
   ValCheckOp    : array[ 0..20 ]  of CROpType;
   ValCheckDesc  : array[ 0..20 ]  of VCHKDesc;
   IndexesOp     : array[ 0..20 ]  of CROpType;
   IndexesDesc   : array[ 0..20 ]  of IDXDesc;
```

Code Snippet 5-4: The DefField procedure found in the SCANNER code

```
procedure DefField ( const sName: string;
                     const iAFldType,iASubType,iAFldNum,
                           iAUnits1,iAUnits2: integer );
begin
  with FieldsDesc[ iAFldNum ] do
  begin
    iFldNum  := iAFldNum;
    StrPCopy( szName, sName );
    iFldType := iAFldType;
    iSubType := iASubType;
    iUnits1  := iAUnits1;
    iUnits2 := iAUnits2;
  end;
end;
```

Code Snippet 5-5: The DefTable procedure found in the SCANNER code

```
Procedure DefTable ( const sName, sType, sPassword : string;
                const iAFldCount, iAIDXCount, iAValChkCount, iARintCount : integer );
begin
  FillChar( TableDesc, SizeOf( CRTblDesc ), #0 );
  with TableDesc do
  begin
    StrPCopy( szTblName, sName );
    StrPCopy( szTblType, sType );
    bProtected := ( sPassword <> '' );
    if bProtected then
    begin
      StrPCopy( szPassword, sPassword );
      Session.AddPassword( sPassword );
    end;
    bPack := true;
    iFldCount := iAFldCount;
    pFldDesc := @FieldsDesc;
    iRintCount := iARintCount;
    pecrRintOp := @RefIntegOp;
    pRINTDesc := @RefInteg;
    iValChkCount := iAValChkCount;
    pecrValChkOp := @ValCheckOp;
    pvchkDesc := @ValCheckDesc;
    iIDXCount := iAIDXCount;
```

Code Snippet 5-5: The DefTable procedure found in the SCANNER code (continued)

```
    pecrIDXOp := @IndexesOp;
    pIDXDesc := @IndexesDesc;
  end;
end;
```

The bulk of the code shown in Code Snippets 5-2 through 5-5 represents manipulation of the supplied information (specifically changing strings from Pascal-style to ASCIIZ strings) and copying of the information into data structures required by the BDE API. The code lines marked in bold in Code Snippet 5-2 contain the actual BDE API calls used to create the database table.

4. Loading of the test database

Creating a test database for an application such as *Hearts 32* is a large and complex undertaking. Instead of creating imaginary data for each table, a conversion program was written to convert some of the existing data found in the *Hearts* database files (dBASE® III Plus format) to the new Paradox® database files.

Traditional tools would require code along the following lines:

Code Snippet 5-6: Pseudo code for table conversion

```
Open source database
Open target database
While not end of source database
   Create a blank record in the target database
   Copy the values of all fields from the source record to the target record
   Read the next source record
While end
```

Delphi™ provides a very useful component in the Visual Component Library (VCL), namely the *BatchMove* component. Cantù summarises the action of the *BatchMove* component as follows: "A third useful component for database manipulation is *BatchMove*, which allows a program to copy, append, or delete groups of records or an entire table from two different databases." (Cantù, 1997 : 835).

By using *BatchMove*, code such as shown above is not needed. Basically, a Delphi™ form must be created containing two *TTable* components and a *BatchMove* component. The first table is connected with the original table file. The second table is connected with the target table file. The *Source* and *Destination* properties of the *BatchMove* component is set to reflect the names of these two *TTable* components. An example of this can be seen in Figure 5-2.

BatchMove can create the destination table, but for this research the destination table was previously created. The main reason for this is the fact that some structural changes were made to the original dBASE® III Plus database file structures. The *Mappings* property of *BatchMove* is used to specify the correspondence between fields in the source and destination tables when field names do not correspond.

In a very basic scenario, the conversion process can be completed without even compiling the program! Simply right click with the mouse on the *BatchMove* component on the form and select *Execute*.

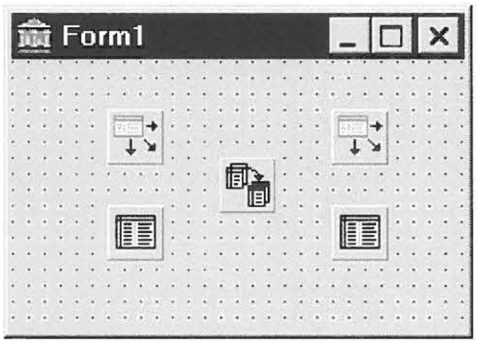


Figure 5-2: Design of a data conversion program using the BatchMove component

The objects and their corresponding properties (for the example found in Figure 5-2) were as follows:

Code Snippet 5-7: Properties of a form for data conversion with BatchMove

```
object Form1: TForm1
  Left = 200
  Top = 121
  Width = 234
  Height = 167
  Caption = 'Form1'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 120
  TextHeight = 16
  object DataSource1: TDataSource
    DataSet = Table1
    Left = 48
    Top = 24
  end
  object DataSource2: TDataSource
    DataSet = Table2
    Left = 160
    Top = 24
  end
  object Table1: TTable
    DatabaseName = 'D:\HEARTS.32\DATABASE\DBASE'
    TableName = 'dokter.dbf'
    Left = 48
    Top = 72
  end
  object Table2: TTable
    DatabaseName = 'Hearts32'
    TableName = 'doctor.db'
    Left = 160
    Top = 72
  end
object BatchMove1: TBatchMove
    AbortOnKeyViol = False
    ChangedTableName = 'ChangTab'
    Destination = Table2
    KeyViolTableName = 'KeyViol'
    ProblemTableName = 'ProbTabl'
    Source = Table1
    Left = 104
    Top = 48
  end
end
```

BatchMove works very easy and efficiently as described above. Normally only one table is translated to another. The situation tends to become more complex when the database consists of a number of tables, such as in the case of *Hearts 32*. What is then needed is a program that will convert the tables one after the other. This automates the process which would otherwise be quite labour intensive (if a programmer had to interactively change the properties of the *TTable* and *BatchMove* components for each table to be converted!).

It is worth mentioning that the complexity of the conversion is further increased by the referential integrity rules defined between the tables in the *Hearts 32* database. As a trivial example, consider the following referential integrity rule:

For each record in the ECG table, a master record containing the computer number for the patient must exist in the Patient table.

If the data conversion process attempts to convert data from the dBASE® III Plus ECG table before the data for the Paradox® Patient table have been converted, the conversion process will abort with a key violation error, since the primary keys needed to satisfy the referential integrity rule in the abovementioned example do not exist in the master (Patient) table.

It is important to point out that the loading of existing ECG data that were captured and stored in files on the departmental file server at the Department of Cardiology, Universitas Hospital, could not be managed by using the *BatchMove* component. The main reason for this was that the data had to be decoded, calculated, reformatted and compressed before they could be stored in the *Hearts 32* database.

5. Summary

Flat file storage systems have some serious drawbacks that can be overcome by the use of a database management system. Client/Server database management systems can provide good solutions, such as cost-effectiveness, enforcing of data integrity rules, division of labour between server and client, lowering of network traffic and backup and recovery utilities. Client/Server technology is not suited to all situations, and generally require a bigger investment than a file server database in terms of effort, skill and finances.

Table level 7 of the Paradox® driver supports enhanced features which makes Paradox® a good choice for a desktop database. The Paradox® support for BLOB fields are of particular interest for the digital storage of ECG data in the *Hearts 32* database, since the exact length of each ECG data set (in bytes) is not known at design-time (of the database).

The DataBase Desktop allows for interactive table creation (definition and population of tables). No provision for documenting these tables is made by the DataBase

Desktop. To overcome this problem, tables can be created using the *TTable* component. While *TTable* allows programmatic table creation, it does not support advanced features found in the Paradox® and InterBase drivers.

Low level access to advanced Paradox® features is facilitated *via* the BDE API. This allows the programmatic creation of a Paradox® table with validation rules, primary and secondary indexes, referential integrity rules and password protection.

The *BatchMove* VCL component is extremely easy to use and very useful for duplicating data from one table to another, even across tables of different types (such as dBASE® and Paradox®).

Chapter 6

Database Storage Format

1. Introduction

In order to make the captured digital ECG data really useful, the data have to be stored on a computer system. The following considerations are important:

1. The size of each file must be kept as small as possible (since we anticipate a large number of files to be stored for the particular application).
2. Retrieval of the stored data must be performed within a reasonable time (less than 5 seconds).
3. The stored data should not be encoded (in other words, the use of special decoding modules should not be required).

It should be stressed at this point, that this chapter is not a study in compression technology. An optimal method for database storage of digitally acquired ECG data in a hardware independent manner will be identified.[1]

2. Description of the data used for analysis

2.1. Introduction

An additional module was created for the existing *Hearts* database to capture digital ECG data sets whilst awaiting the completion of this research.[2] The rationale behind this step was that these digital data sets could later be converted and stored in an appropriate format (as determined by the results of this research).

[1] Hardware independence in this case refers to the format of the data as determined by the ECG machine. It does not refer to storage in the computer system.

[2] A special version of the communications program was prepared in Clipper 5.2. The CA Clipper Tools library version 3.0 (by Computer Associates, the manufacturer of Clipper) was used to implement the RS-232 communications routines.

2.2. Population

One thousand and twelve (1012) ECG data files (HP4745A format) were collected between 20 December 1996 and 8 May 1997. These data files contain digital copies of recorded ECG tracings. In order to better understand the composition of these files, an analysis was done. From the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol* (as discussed in Appendix A) it is known that the data file consists of two major sections:

1. Header Data (further discussed under 5.13 Header Data Description on page 194 in Appendix A). Header Data includes information on patient, CART settings and meta information on the Lead Data, which follows in the section "Lead Data".
2. Lead Data (also referred to as Waveform data in this thesis), are further discussed under 5.14 Lead Data Description on page 198 in Appendix A. Lead Data include lead identifier information as well as the actual lead data which represent the bulk of the data file.

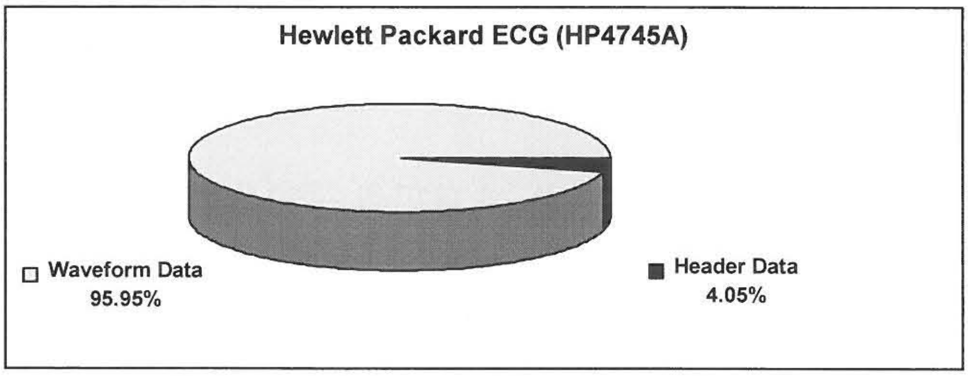


Figure 6-1: Components of digitally recorded ECG traces.

A basic descriptive statistical analysis of the data files yielded the following information:

Table 6-1: Descriptive Statistics (HP Digital Storage Format)

Parameter	Total file size	Header	Waveform$_1$	Patient	Cart Settings	Wave-form$_2$
Min	8766.00	397	8366	32	91	274
Max	15003.00	411	14603	46	91	274
Mean	9878.42	400	9478.38	35.04	91.00	274.00
SD	639.06	0.79	639.07	0.79	0.00	0.00
CV%	6.47	0.20	6.74	2.26	0.00	0.00

In Table 6-1 above, the column descriptions have the following meaning: *Parameter* identifies the statistical parameter, *Total file size* shows the total size of the ECG data file in bytes, *Header* shows the total size of the header information contained in each file, *Waveform₁* shows the total size of the waveform information in bytes. *Patient* shows the portion of the header information which is dedicated to patient information. *Cart settings* shows the portion of the header information dedicated to the settings of the electrocardiograph. *Waveform₂* indicates the number of bytes needed to describe the actual waveform data (which follows in the Lead Data block directly after the Header Data block). The value for the *Header* column is thus the sum of the *Patient*, *Cart Settings* and *Waveform₂* columns.

The following deductions can be made from the descriptive statistics:

- Not all digital ECG files are of the same size (this is indicated by the difference between the minimum and maximum values for *Total file size*).

- The relatively small CV% (6.47%) for the total file size indicates that there is little overall variation from the mean *Total file size* of 9878.42 bytes.

- The extremely small CV% (0.20%) for *Header* information indicates that the *Header* portion of each data set has nearly the same size (400 bytes) and that the *Header* portion of the data set does not contribute significantly to the variation in *Total file size*.

- The portion of the header information responsible for the variation in size (of the header information) is the patient information section. (This can be attributed to the fact that patient identification numbers are represented as ASCII strings of variable length.)

- The CV% of 6.74% for waveform information (*Waveform₁*) shows that the biggest variation in *Total file size* is caused by the size of the waveform information (*Waveform₁*).

- The sum of the storage space needed for the 1012 data files theoretically equals 9.53 MB. (This figure was calculated by summation of all file sizes and does not account for hardware implementation details such as the block size of the fixed disk on which the data are stored, which will allocate more storage space than the size of the file, given that the block size is larger than the average file size. This is typically the case with large hard disk drives. Fortunately, modern operating systems are starting to overcome this problem by implementing block sub-allocation.)

One should keep in mind that for this specific application, not all data fields in the patient section of the digital ECG are present, due to the way in which the electrocardiographs were set up. If all of the patient fields (as defined in Table A-14 on page 195 of Appendix A) were to be entered, the size of the patient section would be 183 bytes (99 bytes for all data, and 4 x 21 (84) bytes for identification purposes).

The reason why the portions of the header data needed for cart settings and waveform information (*Waveform*$_2$) stay constant is that only one ECG configuration is used as standard. If different recording formats were used, these values would also have varied.

The digital ECG files do not only contain recorded ECG data, but also meta-data. These meta-data are used for the identification and description of fields. (The sequence *GS Code RS* is used to identify data fields in the header part of the record. The header part also contains meta-data which detail the waveform data following the header data. Please refer to Table A-13 on page 194 of Appendix A for a complete description of the header block format.)

3. HP Digital Storage Format

The first storage option considered was that of utilising the existing digital storage format as developed by Hewlett Packard. Digital ECG files extracted from HP4745A PageWriter II Cardiograph machines were used for analysis and testing. (A discussion of the extraction procedure can be found in Chapters 4, 8, 9 and Appendix

A.) Please refer to the description of the data used for analysis on page 89 for more information on the characteristics of these files.

With an average *Total file size* of 9878.42 bytes, one cannot help but be tempted to implement HPs native file format. Although these files are extremely compact, the biggest drawback associated with them is the relatively high degree of complexity needed to decode and calculate the actual values from these files, compared to a list (file) with all the relevant data points readily available. The fact that hardware independence is lost, makes such an implementation less attractive.

Another interesting possibility is that of transmitting the file stored on computer back to the ECG machine, thus re-creating the ECG on the ECG machine itself. The biggest drawback of this option is that it is too restrictive in terms of hardware used.

These digital files will be used as a baseline value (at least when comparing file sizes), against which comparisons with other storage options will be made.

Table 6-2: HP Digital Storage Format - Advantages & Disadvantages

Advantages	Disadvantages
• Very small files. • Can be transmitted back to ECG machine.	• Proprietary coding format.

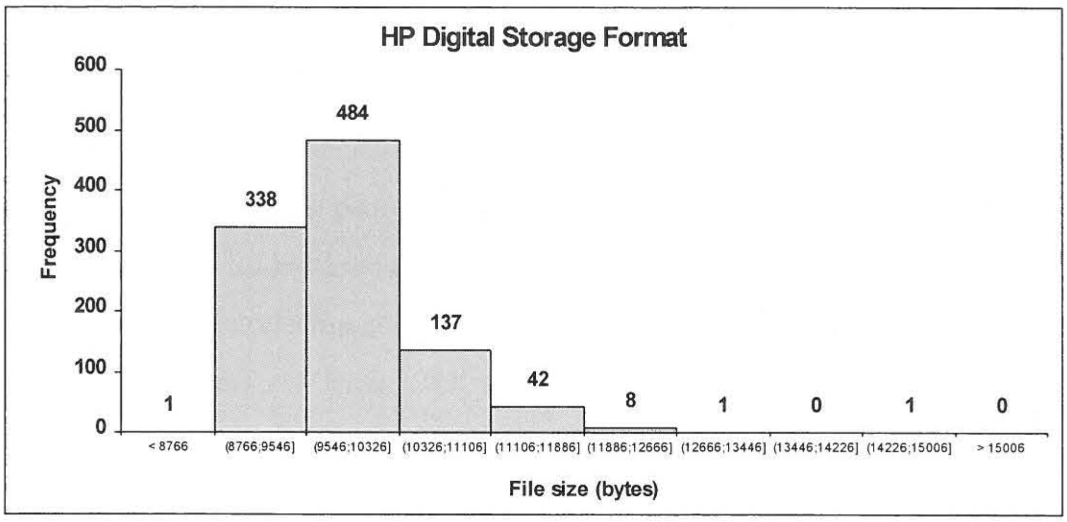


Figure 6-2: Frequency Distribution of file size for the HP Digital Storage Format

4. Storage of decoded data in INTEL binary format

With this storage method the aim is to decode and calculate encoded data. The result of this step (decoded values) is then stored as INTEL formatted binary integers. By this is meant that each integer will be represented by a two-byte word. The binary representation of a number should in most cases occupy less storage space than the ASCII representation for the same number. As an example, consider the number 40960 (A000$_h$). In its ASCII format, this number requires 5 bytes of storage:

4	0	9	6	0

As an INTEL word, it only requires 2 bytes:

00	A0

No extraneous header information was stored. Only header information describing the waveform along with the waveform itself was stored. Data such as patient number, age and sex were omitted from the INTEL binary file (also for subsequent storage formats).

Table 6-3: Storage results for files stored in INTEL binary format.

Size	Header	Waveform
31791	63	31728

Since the electrocardiogram data files collected at the Department of Cardiology were all recorded with the same parameter settings on the electrocardiograph machines, the length of recorded leads stayed the same (2.5 seconds of recorded data per lead). This caused the different electrocardiogram data files to each have the same number of decoded data points per lead. The number of bytes needed to describe the lead information also stayed static across all data files.

Using a binary representation where each value has a fixed length (one word in this case) leads to data files which all have exactly the same length (as can be seen from Table 6-3). The effect of this storage method is that the entropy[3] of the data is in essence ignored. The result is that files are now nearly 300% larger than with the HP Digital Storage Format.

Although this method appears to be better than the HP Digital Storage Format (at least from a decoding point of view) it is not ideal. The storage method should make provision for floating point numbers, as well as be more flexible when it comes to number representation.

No frequency distribution was represented for this storage format, since all files have the same size.

Table 6-4: INTEL Binary Storage Format - Advantages & Disadvantages

Advantages	Disadvantages
• Data already decoded (no proprietary decoding to be performed). • Data are easy to write in this format. • Data can easily be read by a program.	• Massive (300%) increase in file size, compared to the HP Digital Storage Format files. • Entropy ignored, thus all files in this application will have the same size. • Data set cannot easily be migrated to another computing environment (other than the PC), due to the fact that this method actually stores values in their internal representation. • This method only allows storage of short integer (word) values. If the decoded values were real numbers, this method would not suffice.

[3] Nelson (1991 : 15) defines entropy as a measure of the information contained in a message. In this sense, the term message refers to any stream of characters. "The entropy of a symbol is defined as the negative logarithm of its probability. To determine the information content of a message in bits, we express the entropy using the base 2 logarithm:

Number of bits = $-\log_2$(probability)

The entropy of an entire message is simply the sum of the entropy of all individual symbols." (Nelson, 1991 : 16).

5. Storage of decoded data in ASCII format

Storing data in an ASCII format should protect the data set from different implementations of internal representation used for integers. (For example, on a PC, an integer is stored as a word (two bytes) using 16-bit compilers. Modern 32-bit compilers use 32 bits to represent an integer. Another issue is that of swapping of the most significant byte and the least significant byte, as found with the implementation on the PC.)

The file sizes of ASCII files are expected to generally be larger than their binary counterparts, the reason for this being that one byte is used to represent each digit of each value. Where the value 40960 only needs 2 bytes for its binary representation, five bytes are needed for the ASCII representation.

Another problem is that, since data units are not of a pre-defined, fixed length, each unit will have to be separated from the other by using a token. This token character is typically a space. Said token character also adds to the total file size, without really contributing to the data itself.

In contrast with the data stored in INTEL binary format, data files with ASCII data are expected to differ in size, since values are encoded using different lengths. The reason for this is simply that larger numbers are represented using more digits, and thus need more storage space than smaller (shorter) numbers.

Table 6-5: Descriptive statistics (ASCII Storage Format)

Parameter	File Size	Header	Waveform
Min	47313.00	381.00	46932.00
Max	65647.00	381.00	65266.00
Mean	54636.33	381.00	54255.28
SD	2934.02	0.00	2933.97
CV%	5.37	0.00	5.41

The following deductions can be made from the descriptive statistics:

- The size of the header stays constant at 381 bytes for all ASCII files.

- The header size (381 bytes) has increased by just more than six times that of the INTEL binary storage method (63 bytes) as described in Table 6-3.
- The average file size is 1.72 times the size of a file stored in INTEL binary format.
- It is interesting to note that the CV% (5.37%) for the ASCII files is nearly the same as the CV% (6.47%) for the digital storage method developed by Hewlett Packard. Since the bulk of the data is contained in the waveform portion of each file, the variation is caused by variation in the waveform data.

Table 6-6: ASCII Storage Format - Advantages & Disadvantages

Advantages	Disadvantages
• Data already decoded (no proprietary decoding to be performed).	• Files are 1.72 times bigger than INTEL binary files (and nearly 550% bigger than a HP ECG file).
• File sizes are no longer static. Larger values lead to larger files.	• Space wasted since characters to separate values are needed.
• Data sets can easily be migrated to other computers which implement the ASCII coding system.	• More complex to read data. Object Pascal does not allow movement of file pointer for text files. Special routine needed to read values.
• If data originating from other ECG machines are stored in floating point format, these data can easily be stored without any change.	

Considering all points mentioned above, it would appear that the ASCII Storage Format should be the format of choice if one would like to keep the storage format straightforward and simple and also protect the data from differences in different hardware and software. Something needs to be done, however, to decrease the average size of the ASCII data set. The HP Digital Storage Format requires on average 9878.42 bytes per file. The ASCII Storage Format requires on average 54636.33 bytes per file, an increase of 550%!

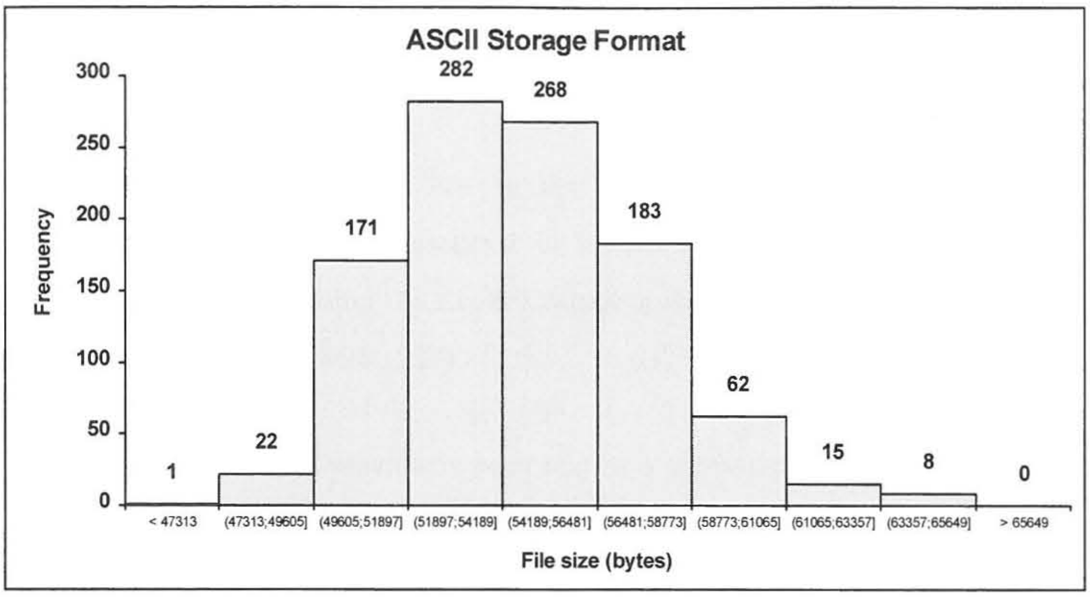


Figure 6-3: Frequency Distribution of file size for the ASCII Storage Format

One way to reduce the size of the ASCII data file is to apply a lossless data compression technique to the data, such as LZ77, LZ78, LZSS or LSW.

The idea is that the module responsible for data acquisition and decoding will produce a data set in a standard ASCII format (as prescribed by the host application, *Hearts 32* in this case). The host application will then store the standard ASCII file in a database. It is up to the host application to perform the compression of the ASCII files before entering them into the database as Binary Large Objects (BLOBs).

An Object Pascal implementation of ZLIB 1.0.4 is included with the Delphi™ Professional Edition version 3. ZLIB 1.0.4 is a general purpose data compression library. The data format used by the ZLIB library is described by Request for comments (RFCs) 1950 to 1952 (ftp://ds.internic.net/rfc/rfc1950.txt, 1951.txt and rfc1952.txt). ZLIB is copyrighted by Jean-loup Gailly & Mark Adler, 1995 - 1996. Permission to use the library is granted, on condition that the original copyright be honoured.

According to Deutsch (Deutsch, 1996b : 4), the DEFLATE compression method used by ZLIB is a lossless data compression method combining the LZ77 compression technique with Huffman coding, with efficiency comparable to the best general-purpose compression methods currently available.

The LZ77 compression algorithm is the genesis of modern dictionary-based compression methods. It is described in the paper "A Universal Algorithm for Sequential Data Compression" by Ziv & Lempel in IEEE Transactions on Information Theory, 1977. (Nelson, 1991 : 233).

"LZ77 compression uses previously seen text as a dictionary. It replaces phrases in the input text with pointers into the dictionary to achieve compression. The amount of compression depends on how long the dictionary phrases are, how large the window into previously seen text is, and the entropy of the source text with respect to the LZ77 model." (Nelson, 1991 : 233).

Dictionary-based compression algorithms such as LZ77 represent the most popular lossless compression methods. LZ77 does, however, have some problems, most notably the performance bottleneck caused by string comparisons against the look-ahead buffer for every position in the text window. In order to improve compression performance, the size of the window (and thus the size of the dictionary) can be increased. This leads to a worsening of the mentioned performance bottleneck. Another performance problem is found in the way that the sliding window is managed. Since phrases may span across windows, normal string comparison functions such as *strncmp()*[4] can no longer be used. Modulo indexes rather than normal indexes into the window should be used. One major efficiency problem associated with LZ77 is that of no matching phrases in the dictionary. Three tokens are used to identify phrases. When a dictionary entry is found, the length of the phrase plus the tokens is less than the original phrase. When no dictionary entry is found, the three tokens are still output, which leads to an increase in the length of the new phrase! (Nelson, 1991 : 238 - 240).

[4] According to Barkakati (Barkakati, 1989 : 289 - 290), the *strncmp()* function is used "to compare a specified number of characters of two strings to one another. The comparison is case sensitive."

Huffman coding creates variable-length codes that consist of an integral number of bits. “Symbols with higher probabilities get shorter codes. Huffman codes have the unique prefix attribute, which means that they can be correctly decoded despite being variable length.” (Nelson, 1991 : 34).

Important properties of the DEFLATE compression method include:

- Independence of CPU type, operating system, file system and character set, thus allowing interchange between different machine types.
- No patent rights, therefore the algorithm can freely be used in programs.
- DEFLATE defines a data format that can produce or consume data for an arbitrarily long, sequentially presented input data stream, using only an *a priori* bounded amount of intermediate storage, and hence can be used in data communications or similar structures such as UNIX filters. (Deutsch, 1996a : 2).
- “Is compatible with the file format produced by the current widely used *gzip* utility, in that conforming decompressors will be able to read data produced by the existing *gzip* compressor.” (Deutsch, 1996b : 3).

The compression methods found in the ZLIB library were applied to the ASCII data files. The results are presented in Table 6-7 below:

Table 6-7: Descriptive statistics (Compressed ASCII Storage Format)

Parameter	ASCII File Size	Compressed ASCII File Size	HP Digital Storage Format
Min	47313.00	8553.00	8766.00
Max	65647.00	20963.00	15003.00
Mean	54636.33	11716.39	9878.42
SD	2934.02	1549.92	639.06
CV%	5.37	13.23	6.47

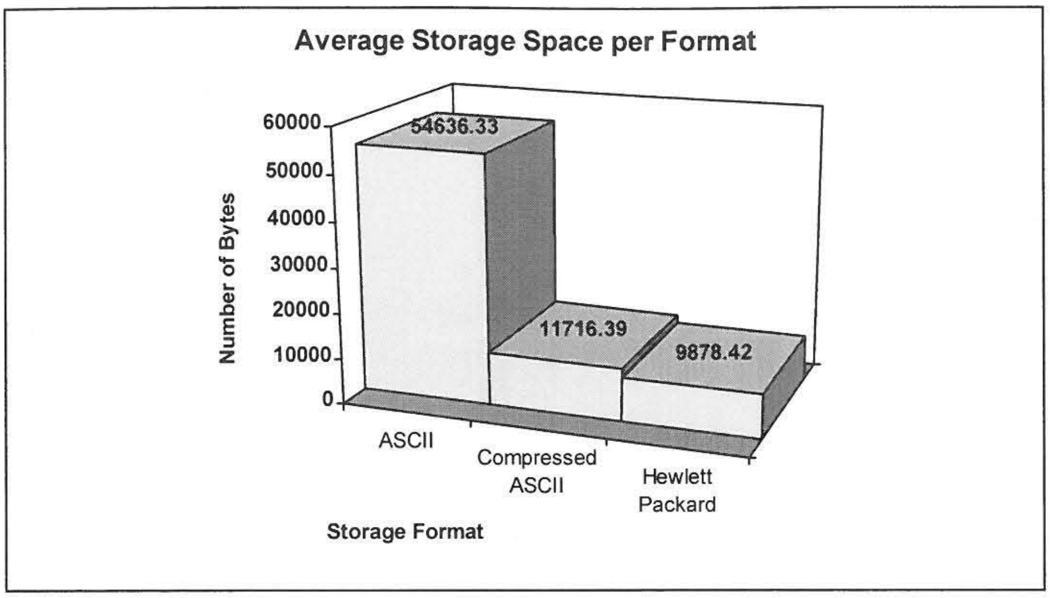


Figure 6-4: Average storage space per format

The following can be seen from the descriptive statistics in Table 6-7:

- Compression of the ASCII files yielded on average a 78.56% reduction of the original ASCII file size.

- The average increase in file size compared to the original HP Digital Storage Format is 18.61%.

Table 6-8: Compressed ASCII Storage Format - Advantages & Disadvantages

Advantages	Disadvantages
• File size dramatically smaller than with plain ASCII storage format. • Underlying format of the ASCII file is not influenced by the compression method chosen.	• Format more complex than simple ASCII files, since decompression has to be performed in order to access the ASCII data files.

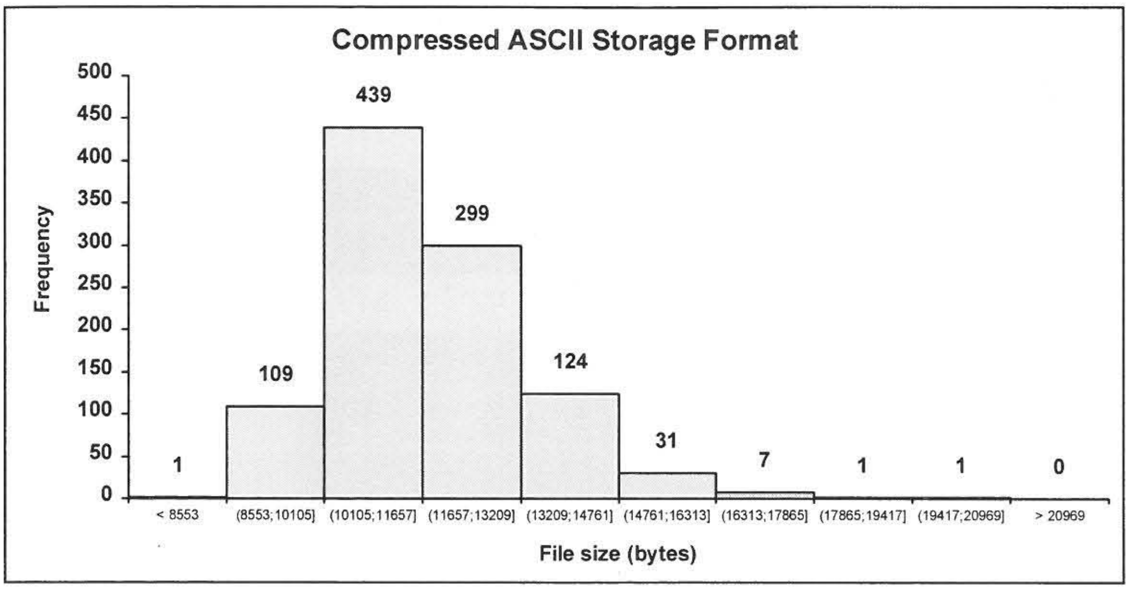


Figure 6-5: Frequency Distribution of file size for the Compressed ASCII Storage Format

6. Selective storage of decoded data in ASCII format

The data acquired from a HP4745A actually contain 21 leads (excluding the position bit leads).

Table 6-9: Leads present in a HP4745A digital ECG file

Number	ID	Lead description
1	1	I
2	2	II
3	3	III
4	4	aVR
5	5	aVL
6	6	aVF
7	7	V1
8	8	V2
9	9	V3
10	10	V4

Table 6-9: Leads present in a HP4745A digital ECG file (continued)

Number	ID	Lead description
11	11	V5
12	12	V6
13	101	ACAL 1
14	102	ACAL 2
15	103	ACAL 3
16	24	$V1_r$
17	25	$V2_r$
18	26	$V3_r$
19	104	RCAL 1
20	105	RCAL 2
21	106[5]	RCAL 3

However, not all of these leads are needed for recreating the ECG; of the 21 leads the data for only 14 leads are needed. Leads that could be discarded are marked in grey in Table 6-9. Remember that the height of the calibration pulse is equal to a voltage of one millivolt (this is the standard format in use at the Department of Cardiology, Universitas Hospital, Bloemfontein). The calibration pulse is thus used for measuring the height of the waves present on the ECG.

A large saving in the total file size could be expected if two of the three rhythm leads are omitted (items 17 & 18 in Table 6-9 on page 102). Rhythm lead data are collected for a 10 second period. A sample is recorded every 0.004 seconds, resulting in 2500 samples collected for the 10 second period. Theoretically, the storage space required to store 5000 values could be saved by omitting the data for two leads. An examination of the actual ECG data files showed that the rhythm leads only contain 2488 samples. This means that the actual saving is the storage space required to store

[5] Note that the lead IDs 101, 102, 103, 104, 105 and 106 do not match IDs from Hewlett Packard. There are no IDs prescribed for ACAL 1, ACAL 2, RCAL 1 and RCAL 2 in the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol*. The ID for ACAL 3 is 39, and 42 for RCAL 3 (according to the documentation). For the sake of simplicity the IDs for the ACAL and RCAL leads are kept uniform.

4976 values. The actual saving in bytes (length of the values) cannot easily be calculated, since the length of each value is not fixed.

Furthermore, the storage space required to save another 530 values can be omitted from the file by omitting five of the six calibration pulse leads (items 14, 15 and 19 - 21 in Table 6-9 on page 102). Each of these calibration pulse leads represents 106 samples.

Except for discarding the extraneous lead information, this new ASCII file is in all respects identical to the original ASCII files as described in section 5 on page 96. The savings achieved are significant, as can be seen from the following results:

Table 6-10: Descriptive statistics (ASCII Storage Format, Selective)

Parameter	File Size	Header	Waveform
Min	29932.00	255.00	29677.00
Max	40902.00	255.00	40647.00
Mean	34552.65	255.00	34297.65
SD	1780.63	0.00	1780.63
CV%	5.15	0.00	5.19

The following deductions can be made from the descriptive statistics in Table 6-10:

- The size of the header stays constant at 255 bytes for all ASCII files which selectively store data.
- There is an average saving of 20083.68 bytes (54636.33 - 34552.65).

The saving illustrated above is not significant enough to justify storage of the data set in ASCII form.

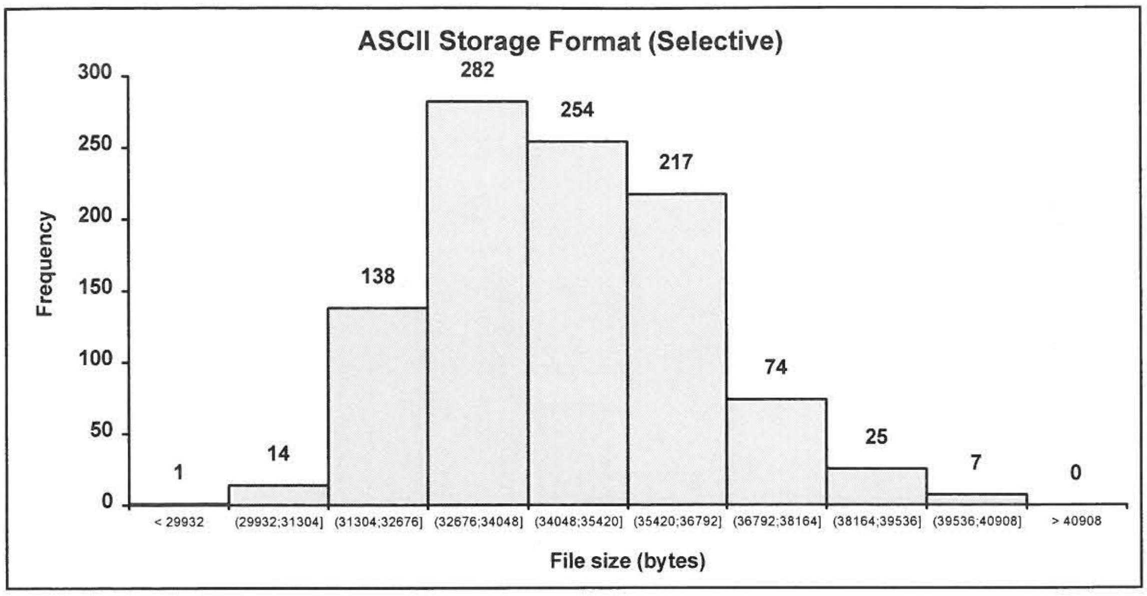


Figure 6-6: Frequency Distribution of file size for the ASCII Storage Format (Selective)

Once again the data were subjected to the same compression process described in section 5 page 96. The results are presented below:

Table 6-11: Descriptive statistics (Compressed ASCII Storage Format, Selective)

Parameter	ASCII File Size (Selective)	Compressed ASCII File Size (Selective)	HP Digital Storage Format
Min	29932.00	5252.00	8766.00
Max	40902.00	13784.00	15003.00
Mean	34552.65	7435.40	9878.42
SD	1780.63	1000.81	639.06
CV%	5.15	13.46	6.47

The following deductions can be made from the descriptive statistics in Table 6-11:

- Compression of the ASCII files yielded on average a 78.48% reduction of the original ASCII file size.
- The average decrease in file size compared to the original HP Digital Storage Format is 24.73%.

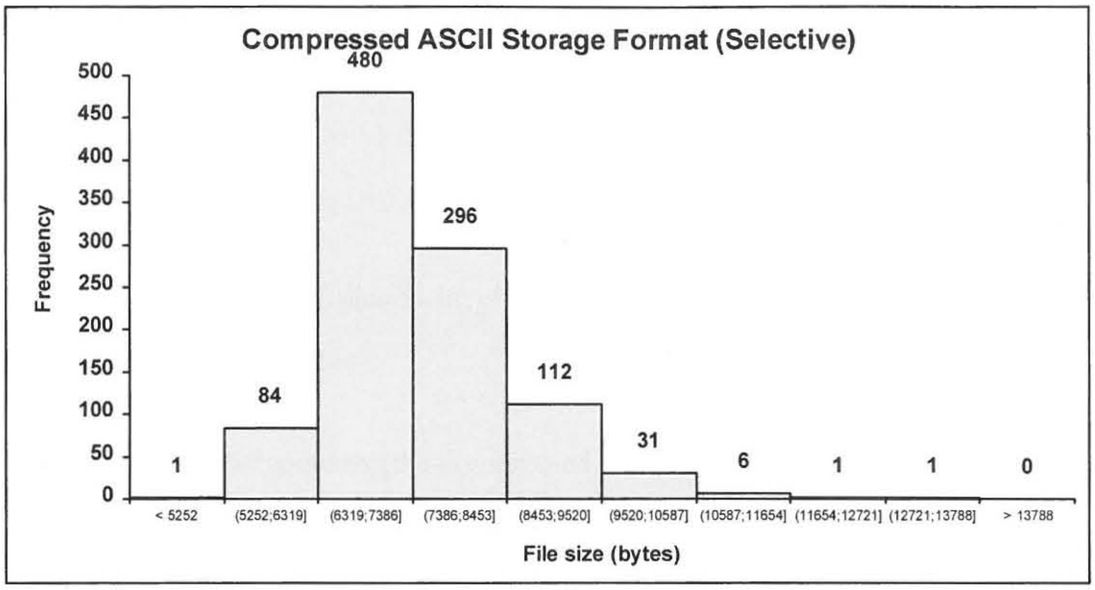


Figure 6-7: Frequency Distribution of file size for the Compressed ASCII Storage Format (Selective)

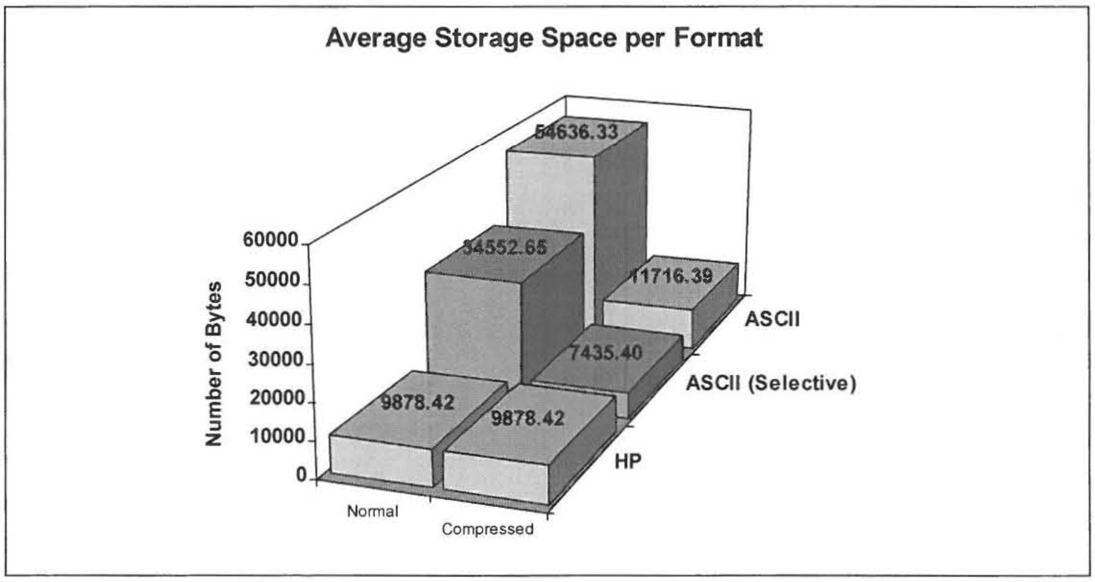


Figure 6-8: Average storage space per format, with the effect of compression

It can clearly be seen from Figure 6-8 above that the selective storage of data in a compressed ASCII format will result in the most compact utilisation of storage space, without any loss of detail in the ECG itself.

7. Retrieval time for compressed data

Even though the selective ASCII data could be stored in a limited amount of space by using the ZLIB compression software, care had to be taken regarding the amount of time needed for retrieval.

There were three distinct phases involved in the access to ECG data stored in the *Hearts 32* database:

1. Extraction of the contents (the compressed ECG data set) from the BLOB field in the *Hearts 32* database, with the result saved to a temporary disk file.
2. Decompression of the contents of the temporary disk file into another temporary disk file, containing the reconstructed ASCII data set.
3. Loading of the reconstructed ASCII data set by the *Hearts 32* ECG Browser software for graphical display and interaction.

The user's perception is, of course, that there is only one phase; that being the retrieval of the ECG data set with the result graphically displayed.

A program was written to analyse the amount of time needed for access to the stored ECG data sets. The population contained 627[6] ECG data sets in the *Hearts 32* database. The program measured the amount of time needed for each one of the phases as outlined above. All results are in milliseconds (ms), and are summarised in Table 6-12 below:

6 Although the rest of the analysis was performed using 1012 ECG data sets, only 627 ECG data sets were physically present in the *Hearts 32* test database. The reason for this was that many of the existing patient records did not migrate from the existing *Hearts* database into the new *Hearts 32* database, due to conflicts with the new referential integrity rules defined in the *Hearts 32* database.

Table 6-12: Descriptive statistics (Retrieval time for Compressed Data)

Parameter	Time to Extract (ms)	Time to Decompress (ms)	Time to Load (ms)	Total Time (ms)
Min	15.00	41.00	4189.00	4285.00
Max	513.00	1036.00	8336.00	8459.00
Mean	59.31	74.06	4658.35	4791.71
SD	32.85	58.60	422.31	429.45
CV%	55.39	79.13	9.07	8.96

From the means in Table 6-12 it can clearly be seen that the average time to extract the ECG data set from the database, as well as the decompression time, represent only a small portion (2.783%) of the total time needed to display the ECG. The lion share of the time is used by the I/O routines in the *Hearts 32* ECG Browser.

An initial thought was that a correlation between the file size and the time needed for extraction and decompression might exist. Since other factors[7] also influence the extraction and decompression time, it was decided not to perform further statistical analysis. If such a statistical analysis was performed without the effect of these factors, the result would be skewed.

The main purpose of this exercise is to supply a ballpark figure for the amount of time needed for retrieval of ECG data.[8]

A complete discussion on the actual decoding (decompression) of a digital ECG file (HP4745A format) can be found in Chapter 8, and also in Appendix A.

7 These factors are difficult to quantify, and include items such as the number of processes running in Windows® 95 at a given point in time, the CPU intensity of these processes, the size of the disk cache in memory, the size of the virtual memory swap file and also the priority of the tast being executed.

8 The results were generated on a 486 DX4 100 MHz IBM compatible PC with 32 MB RAM and 2 x 1.3 GB hard disk drives.

8. Summary

Since the amount of ECG data to be stored is expected to be large, care must be taken to choose a storage format that allows for relatively small data files whilst allowing standardised access to these data, thus conserving storage space. In this chapter these concerns were identified, gathered data were analysed and a storage format specified. ECG data will be stored in the *Hearts 32* database in a compressed ASCII format, using only selected leads. The ZLIB compression library will be used to compress the ASCII ECG data files.

Chapter 7

Hearts 32 ECG Browser

1. Introduction

A graphic display module is needed to present the digitally captured ECG data stored in *Hearts 32*. Besides display and printing capabilities, some additional functionality, such as simultaneous display of multiple ECGs, measurement, zoom and superimposing, is needed.

2. Description of the browser

The browser also supports zoom functions, callipers[1] for measuring of interval duration and voltation, as well as superimposing of selected leads in the same ECG and also printing of the ECG.

2.1. Selecting ECGs in Hearts 32

It is assumed that the *ECG* tab of Hearts 32 has been selected and that the *ECG* page is active. Before ECGs can be selected, the correct patient record needs to be identified. This can be done by clicking on the *Search* speedbutton.[2] A dialogue box such as Figure 4-11 (Chapter 4, page 68) will be displayed. (This dialogue box can also be activated by using the *Search* command located in the main menu at the top of the window.) As soon as the patient record has been retrieved, a list of available ECGs will be displayed (as in Figure 7-1).

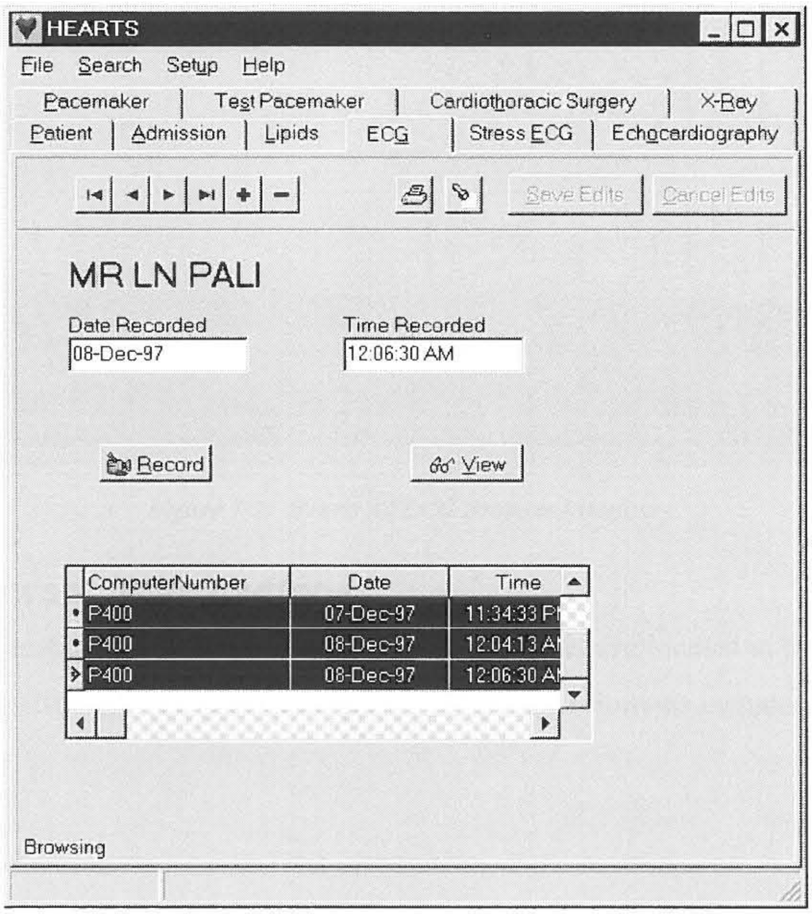
Selecting an ECG for viewing is easy: simply click on the desired record with the mouse. Multiple ECGs can be selected by holding down the *Ctrl* key and clicking on the desired records. Note that the Shift-Click method of Windows® (selecting a

[1] A calliper is a measuring instrument.

[2] The *Search* speedbutton is located to the top of the screen, right of the print speedbutton and left of the *Save Edits* button, and contains a bitmap of a flashlight.

range) is not implemented in the browser. This appears to be the default behaviour for Delphi™.

Figure 7-1 illustrates the selection of the three ECG records available for the selected patient.



MR LN PALI

Date Recorded: 08-Dec-97

Time Recorded: 12:06:30 AM

ComputerNumber	Date	Time
P400	07-Dec-97	11:34:33 P
P400	08-Dec-97	12:04:13 A
P400	08-Dec-97	12:06:30 A

Figure 7-1: Hearts 32 ECG page

2.2. Viewing ECGs in Hearts 32

To view the selected ECGs, click on the *View* button.[3] The *Hearts 32* ECG Browser will load. Initially, no ECG will be displayed. To start the display, click on the *Draw* speedbutton.[4] The selected ECGs will be displayed within a few seconds, as illustrated in Figure 7-2.

[3] The *View* button is located underneath the *Time Recorded* edit box, and contains a bitmap of a pair of glasses next to the word *View*.

[4] The *Draw* speedbutton is located directly underneath the *File* item of the main menu, at the top left hand of the window.

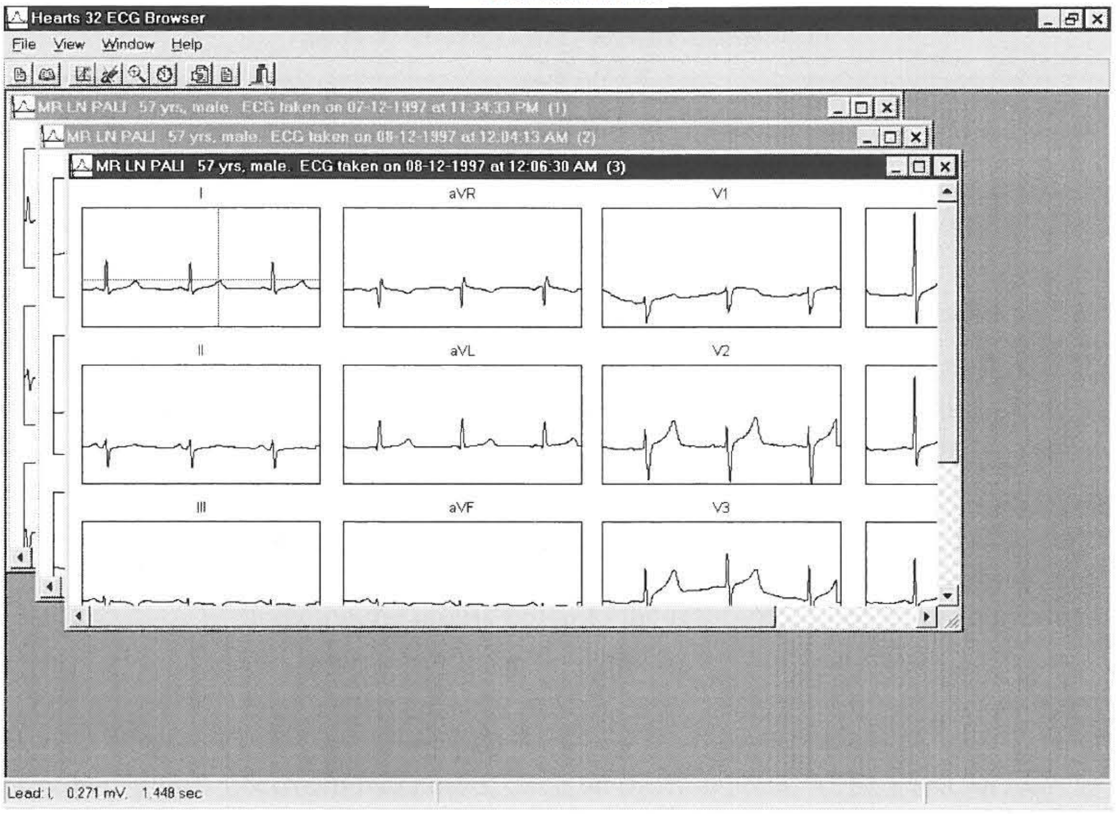

Hearts 32 ECG Browser
File View Window Help
MR LN PALI 57 yrs, male. ECG taken on 07-12-1997 at 11:34:33 PM (1)
MR LN PALI 57 yrs, male. ECG taken on 08-12-1997 at 12:04:13 AM (2)
MR LN PALI 57 yrs, male. ECG taken on 08-12-1997 at 12:06:30 AM (3)
I
aVR
V1
II
aVL
V2
III
aVF
V3
Lead: I, 0.271 mV, 1.448 sec

Figure 7-2: Hearts 32 ECG Browser Interface

2.3. Browser Speedbuttons

The nine speedbuttons used to interact with the Browser are located at the top of the window, directly underneath the main menu. These speedbuttons include:

Table 7-1: Hearts 32 ECG Browser Speedbuttons

No	Name	Description
1	Draw	Draws ECGs selected in ECG page of *Hearts 32*.
2	Full size	Enlarges the currently active MDI child window to its full size (978 pixels wide by 610 pixels high.
3	Toggle Grid	Enables/Disables drawing of a grid in the background of the current active MDI child window.
4	Refresh ECG	Refreshes the contents of the currently active MDI child window. This is sometimes needed when the crosshair leaves marks on the graph.
5	Toggle Zoom	Enables/Disables zooming of the currently active MDI child window. Zooming-in is performed by clicking, holding and dragging the mouse down to the right. Zooming-out is performed by clicking, holding and dragging the mouse up left.
6	Toggle Calliper	Enables/Disables the use of a calliper in the currently active MDI child window. To measure the interval duration and voltage between two points, click on the first point. (The word *Measuring* is displayed in the status bar of the main window, bottom right.) Click on the second point. The result of the measurement is displayed in the status bar of the main window, bottom centre.
7	Print ECG	Prints the ECG to the printer designated as the default printer in Windows® 95.
8	Super-impose	Draws more than one lead of the currently active MDI child window on the same graph (in a new window).
9	Exit	Terminates the Browser and return control to *Hearts 32*.

2.4. Window management & viewing of multiple ECGs

Window management commands are used in the Browser to allow manipulation of the placement of MDI child windows within the application frame. There are currently no speedbuttons available in the Browser for these window management functions. These functions are accessed from the *Window* item in the main menu (located at the top of the Browser window). They include:

Table 7-2: Hearts 32 ECG Browser Window Management Commands

No	Name	Description
1	Cascade	Stack the MDI child windows on top of each other, as in Figure 7-2.
2	Tile Horizontal	Use this for simultaneous viewing of long horizontal items, such as the Rhythm Lead (shown in Figure 7-4).
3	Tile Vertical	Enables comparison of the same lead groups across different ECGs, as displayed in Figure 7-3.
4	Arrange Icons	Move all minimised MDI child windows to the bottom left side of the screen.
5	Minimise All	Minimise all MDI child windows. A minimised MDI child window can be restored by clicking on the restore button of the window (located at the top right side of the window, two small intersecting boxes).

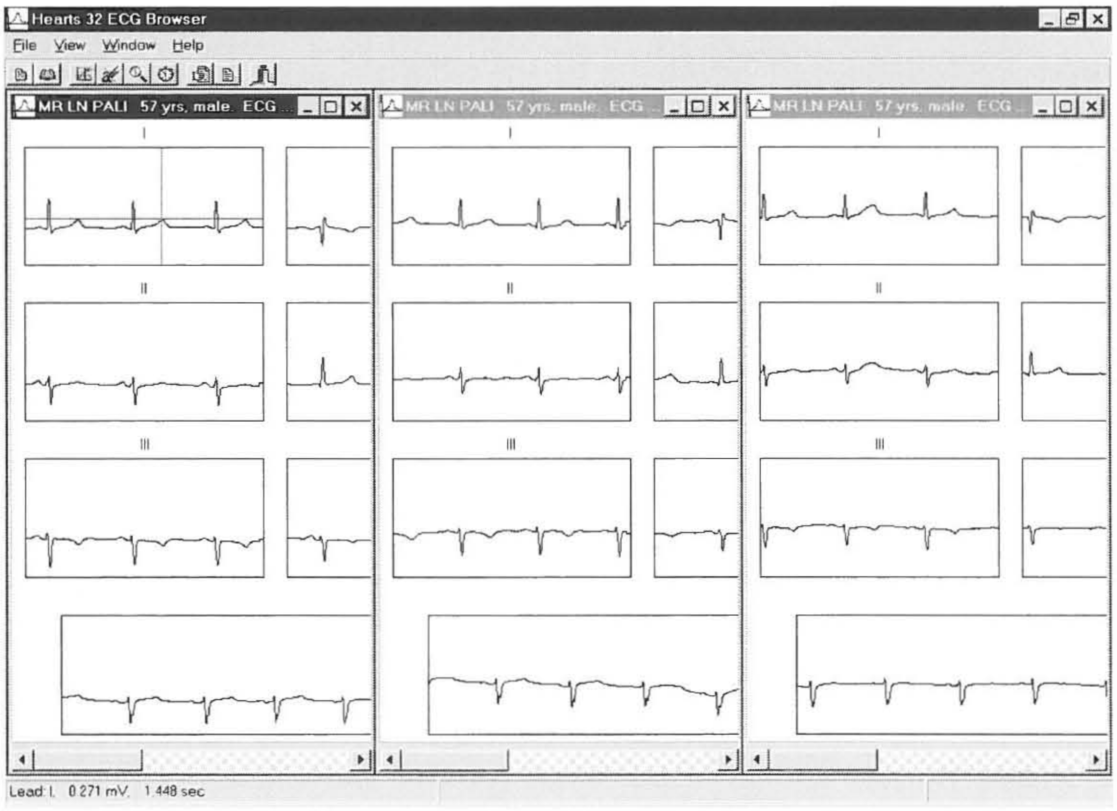


Figure 7-3: Vertical Tiling of MDI Child Windows in the Hearts 32 ECG Browser

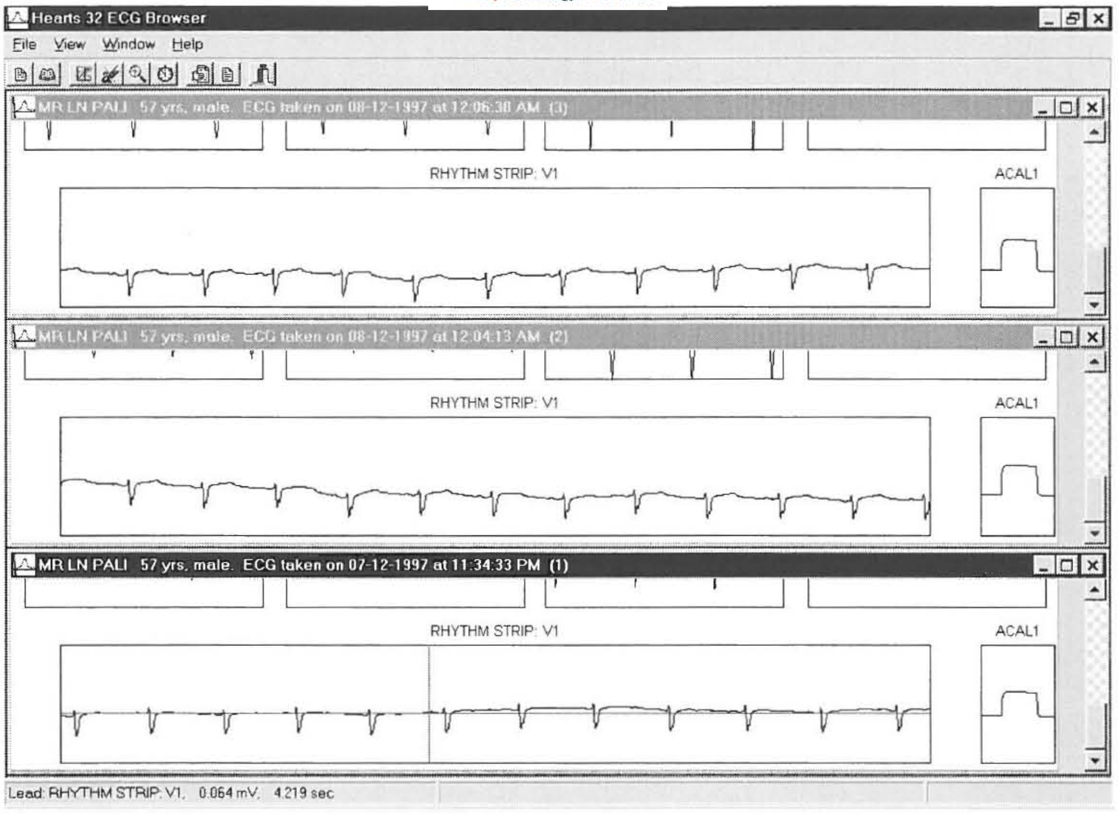

Hearts 32 ECG Browser
File View Window Help
MR LN PALI 57 yrs, male. ECG taken on 08-12-1997 at 12:06:38 AM (3)
RHYTHM STRIP: V1
ACAL1
MR LN PALI 57 yrs, male. ECG taken on 08-12-1997 at 12:04:13 AM (2)
MR LN PALI 57 yrs, male. ECG taken on 07-12-1997 at 11:34:33 PM (1)
Lead: RHYTHM STRIP: V1, 0.064 mV, 4.219 sec

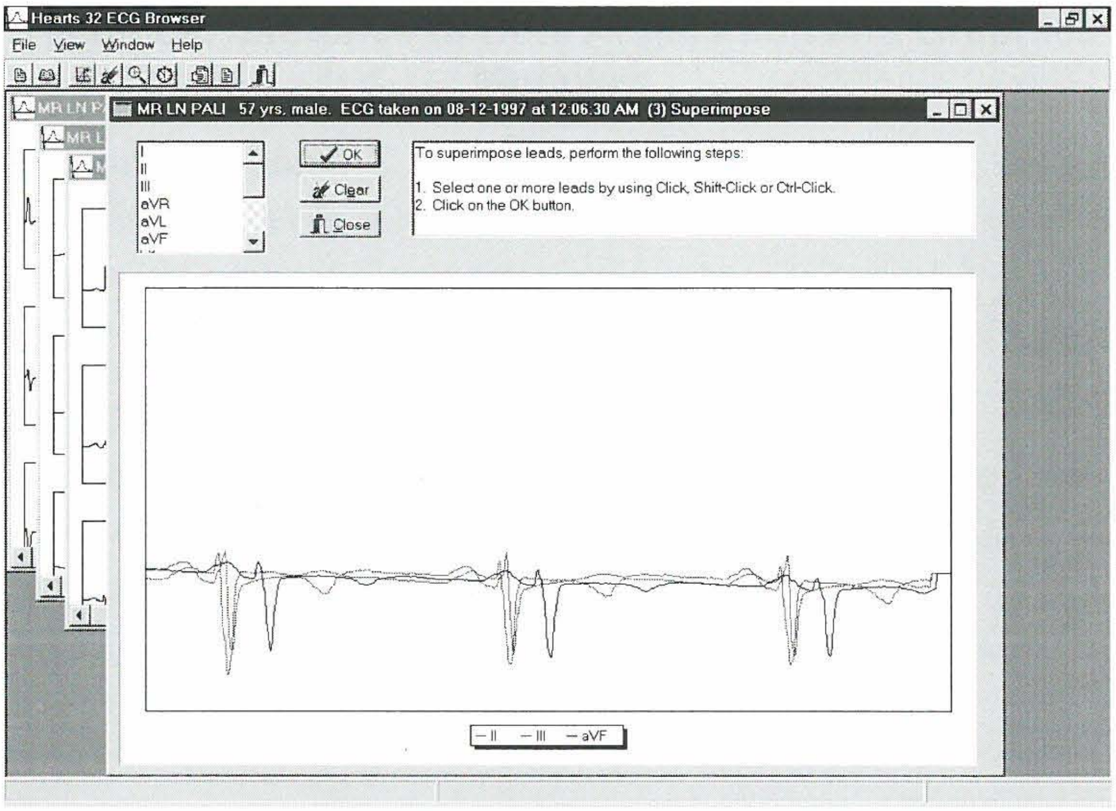
Figure 7-4: Horizontal Tiling of MDI Child Windows in the Hearts 32 ECG Browser

2.5. Superimposing of selected leads of an ECG

To superimpose leads from the same ECG, click on the *Superimpose* speedbutton. A dialogue box, such as in Figure 7-5, will be displayed. Follow the steps outlined on the top right side of the window by selecting the desired leads on the left side of the screen. The *Ctrl*-Click and *Shift*-Click methods of Windows® 95, used for multiple selection (separate items and list selection), apply. When the leads have been selected, click on the *OK* button. The *Clear* button is used to clear the graph area. The *Close* button will close this window and return to the *Hearts 32* ECG Browser.

The result of the superimposing of leads for one ECG cannot be printed. Furthermore, the crosshair, calliper and grid tools are not supported. The purpose of the superimpose tool is to provide the cardiologist with a quick view of the selected leads.

While leads can be freely selected for superimposing, it does not make sense to select leads at random. The following groupings should be used: II + II + aVF (inferior aspect), I + aVL + V5 + V6 (lateral aspect) and V1 + V2 + V3 + V4 (anteroseptal aspect). aVR represents a mirror image of II and is not used.



MR LN PALI 57 yrs. male. ECG taken on 08-12-1997 at 12:06:30 AM (3) Superimpose

To superimpose leads, perform the following steps:

1. Select one or more leads by using Click, Shift-Click or Ctrl-Click.
2. Click on the OK button.

Figure 7-5: Superimposing of selected leads of an ECG in the Hearts 32 ECG Browser

3. Summary

The ECG browser software developed for *Hearts 32* allows the user to view, interact and print ECGs stored in the *Hearts 32* database. The Multiple Document Interface (MDI) architecture of the browser ensures that simultaneous views of different ECGs can easily be displayed.

Chapter 8

Technical reference of Data Acquisition and Graphics Modules

1. Introduction

This chapter discusses the technical detail of the data acquisition module as well as the graphical display module developed as a result of this research.

2. Development environment

The *Hearts 32* database application (as well as other programs described in this thesis) was developed on a 486 DX4 100 MHz computer with 32 MB RAM and two 1.3 GB hard disk drives. A 2 MB PCI VGA card running at a resolution of 1024 x 768 pixels with a colour depth of 256 colours was connected to a GoldStar StudioWorks 78i 17" super VGA monitor.

The operating system used on the development machine (including workstations) was Windows® 95 build 4.00.950. The following software development tools and libraries were used during the research:

- Borland® Delphi™ Professional Edition version 3.0
- Async Professional™ 2.11 for Delphi™ 3.0
- InstallShield Express Delphi™ Edition version 1.11
- Borland® C++ version 3.01
- C/Math Toolchest version 1.0
- The SemWare® Editor pre-release version 1.00A
- Microsoft Office Professional version 4.3

The Novell® Netware® 4.1 file server at the Cardiology Department was connected *via* a 10BaseT Ethernet segment with 20 workstations. The IBM file server was equipped with a 3COM PCI 10/100 Mbit network interface card, 64 MB RAM, a 4 GB SCSI hard disk and a 4 GB SCSI DAT drive. The Arcserve 6.0 backup software from Cheyenne was used for preparing backups.

The basic configuration details of the workstations were as follows:

- 486 DX4 100 MHz
- 16 MB RAM
- 1 GB hard disk
- 1 MB VGA card
- SMC Ultra 10 Mbit network interface card
- Mouse

Connected to one of these workstations was a HP4745A PageWriter II Cardiograph. The configuration details of the RS-232 communications port can be found on page 167 of Appendix A.

3. Data Acquisition Module

The Data Acquisition Module can be divided into three major sections, namely Data Communications (transmission), Decompression (decoding) and Calculation. This is illustrated in Figure 8-1 below. During data communications, data acquired from the HP4745A are buffered in main memory, where they await further processing performed by the decoding and calculation steps. The end result is the decoded ECG data set.

NOTE: Since the information contained in the HP Diagnostic Cardiology Digital Transmission Protocol was made available to the researcher by the Hewlett-Packard Company under a confidential disclosure agreement, a detailed discussion of the program code needed to decode the digital ECG data set can be found in Appendix A.

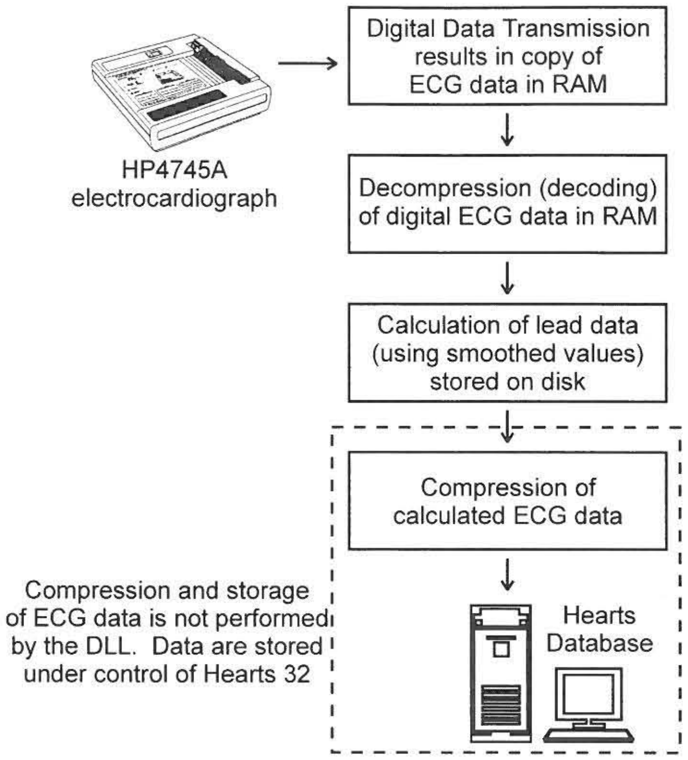

HP4745A electrocardiograph
Digital Data Transmission results in copy of ECG data in RAM
Decompression (decoding) of digital ECG data in RAM
Calculation of lead data (using smoothed values) stored on disk
Compression of calculated ECG data
Hearts Database
Compression and storage of ECG data is not performed by the DLL. Data are stored under control of Hearts 32

Figure 8-1: Graphic overview of the data acquisition process

3.1. Global Variables

Table 8-1: Global variables used during data acquisition

Variable	Type	Short description
Form1	TForm1	Object instantiated from the TForm1 class. This represents the window which is the actual user interface.
BlockIn	BUFFER	Temporary buffer used for reading blocks of data from the HP4745A.
BlockOut	BUFFER	Temporary buffer used for creating responses which are sent to the HP4745A.
CountIn	Integer	Numeric value of the number of bytes in the following data block.
Ack	Integer	Counter used to determine the value of the next sequence indicator for the Acknowledgement block. The value of the counter continuously increases. The sequence indicator is determined by dividing the value of the counter modulo 2.

Table 8-1: Global variables used during data acquisition (continued)

Variable	Type	Short description
CountBlockReceived	Boolean	If a count block has been received, the acknowledgement block does not contain any sequence indicators (0 or 1). Other blocks are followed by an acknowledgement plus an additional sequence byte (0 or 1). This mechanism attempts to prevent the loss of a count block.
Status	FSM	The Status variable indicates the status of the Finite State Machine used to control the download.
ReadIndex	Integer	Specifies the position of the input buffer where the next character is stored from the COM port. Also used to determine whether all the characters which are expected for a given block type have been read.
Quit	Boolean	The main data communications loop is controlled by this variable. As soon as Quit has been set to true, the loop terminates.
MemArea	PChar	Pointer to character buffer where ECG data will temporarily be stored. This buffer is 32 KB large.
ByteCountMA	Integer	Keeps track of the total number of bytes stored in main memory. Indicates length of data stream.
Data	Byte	Byte value containing the data pointed to by the pointer MemArea + Offset.
ShiftCount	Byte	Keeps a tally of the number of bits extracted per byte. As soon as a byte has been scanned, the next byte has to be set up for bit scanning.
Index	Integer	Used to index the digital ECG data stream when dereferencing sequential byte values.
OffSet	Integer	Offset into the digital ECG data stream (from the start of the Lead data).
DynamicArray	^DynArr	Pointer to dynamically allocated two dimensional array. Values for each of the three leads in each lead group will be stored here, one lead group at a time.

3.2. Data Communications

Data Communications group together the activities responsible for the actual downloading of the digital data from the HP4745A ECG machine. The following procedures and functions work together to this end:

3.2.1. Function ReadDecodeCalcStoreECG

This procedure is the only exported module contained in the DLL. Its only purpose is to serve as an interface between the application program (such as *Hearts 32*) and the procedure/function in the DLL that is responsible for data communications. If a DLL is developed for another ECG machine, it should contain an exported procedure called *ReadDecodeCalcStoreECG*.

A description of the parameters passed to the *ReadDecodeCalcStoreECG* function can be found in Table B-6 on page 223 of Appendix B. The possible return values for the *ReadDecodeCalcStoreECG* function are also documented in Appendix B.

When implementing a Delphi™ form as a DLL, care must be taken not to have any of the forms automatically created. This means that all forms in the DLL must be removed from the application's form auto create list. When a form needs to be displayed, the form's *ShowModal* method must be used. Displaying forms in a DLL using the *Show* method can cause problems, since the *Show* method is used to display modeless forms.

In the *ReadDecodeCalcStoreECG* function of the HP4745A DLL, the parameter values passed to *ReadDecodeCalcStoreECG* are assigned to global variables. This eases access to these values by the different procedures and functions contained in the DLL, since multiple parameters do not have to be passed to each and every function/procedure.

Code Snippet 8-1: Outline of the ReadDecodeCalcStoreECG function

```
var
   Form1 : TForm1;

begin
   ReturnValue := -1;
   Application.Handle := hAppHandle;
   Form1 := TForm1.Create( Application );
   try
      with Form1 do
      begin
         ShowModal;
      end;
   finally
      Form1.Free
   end;
   Result := ReturnValue;
end;
```

In Code Snippet 8-1 above, the assignment of the parent application's handle to the instance variable of the DLL is made. Manual form creation, invoked *via* the *ShowModal* method, and destruction of the form are also illustrated. The value of the global variable *ReturnValue* is assigned to the *ReadDecodeCalcStoreStoreECG* function.[1]

3.2.2. Procedure GetFile

GetFile is responsible for the acquisition of the ECG data from the HP4745A. This procedure contains the main program loop and invokes other functions as needed. In order to speed up processing, a buffer is set up in main memory for storage of digital ECG data. From the analysis of 1012 digital ECG files it was found that the average size of these files is 9878 bytes. To be safe, however, a buffer of 32 KB is reserved.

Since the *HP Diagnostic Cardiology Digital Transmission Protocol* was not redeveloped in software, some way was needed to control the flow of execution. To this end a primitive form of a finite state machine was used. The download process can only be in one given state at any time. The following states were identified:

FSM_INITIAL, FSM_ATTENTION, FSM_LINE_BID, FSM_SYS_READY, FSM_ID_TEST, FSM_RUN_TEST, FSM_ECGIN, FSM_HEADER, FSM_LEAD, FSM_MESSAGE, FSM_QUIT.

[1] Assigning a value to the name of a function is the method used by Object Pascal for returning a value from a function. A new method is to assign the return value to the *Result* variable.

GetFile opens the COM port, and enters the main loop. During the main program loop, *GetFile* attempts to read a block of data from the HP4745A. If a block has been read, it is interpreted in order to determine what action should be taken. This action is of course guided by the current state of the download process, that being one of the states indicated above.

As soon as the digital ECG information has been transferred, the program loop is terminated and the COM port closed. The buffer containing the digital ECG data is NOT cleared, since it will be used during the decoding step to produce the decoded data set.

A time-out feature was built into the main loop to ensure that the data acquisition module can escape from situations where no data are received, or a communications lock up occurs. This was managed by using the Windows® API function *GetTickCount*[2], recording the start and end times during different cycles of the loop. A period of five seconds of inactivity terminates the loop and signals an error condition, resulting in a return value of -1.

Code Snippet 8-2: Pseudo code for GetFile procedure

```
Setup buffer in memory
Open communications port
Record a start and end time
Repeat
   If a low level data block has been read
      Record new start time
      Handle each type of data block
   Else
      Record new end time
      If (End time - Start time) > 5 seconds
         Abort procedure
      End if
   End if
Until finished
Close communications port
```

3.2.3. Function CRC

The *CRC* function is used to calculate a Cyclic Redundancy Checksum for a block of data. Since the digital ECG data are compressed, an error of even 1 bit can be disastrous. For this reason, CRC checking is implemented. The CRC calculation

[2] The Win32® API function *GetTickCount* retrieves the number of milliseconds that have elapsed since Windows® was started.

routine implemented in the data acquisition module differs from that described in the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol.* It does, however, produce CRCs which are identical to the CRCs produced by the method described in the *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol.* The latter method calculates CRCs on a bit-by-bit basis. The algorithm implemented here performs CRC calculation on a byte basis, which speeds up the calculation. The CRC calculation routine implemented for this research is strongly based on the work of Campbell. (Campbell, 1987 : 539 - 542).

A 16-bit coefficient table was built using a program. The output of the program (i.e. the coefficient table) was then imported into the Object Pascal unit.

Note that the initial value of the CRC has to be set to $FFFF_h$ in order to stay compatible with the HP4745A CRC values.

An additional discussion on the subject of CRCs can be found in section 4.7.7 *Cyclic Redundancy Checking Algorithm* on page 181 of Appendix A.

3.2.4. Function ReadLLPBlock

The *ReadLLPBlock* function is responsible for reading characters from the ECG machine and constructing a low level protocol block that the *GetFile* procedure can use. *ReadLLPBlock* is aware of the length of each low level protocol block (in bytes), based on the type of the block. It will gather the bytes needed per block, before passing the block on. The use of the *ProcessMessages* method of the *Application* object (*Application.ProcessMessages*) ensures that Windows® is able to service other events when a routine is busy in a tight loop, such as found in the *ReadLLPBlock* function.[3]

[3] In Windows 95® the effect of *Application.ProcessMessages* will be local to the running application. In 16-bit versions of Windows® (such as Windows 3.1®), all the programs being multi-tasked suffered if one application contained a processor-intensive loop which did not occasionally yield control to Windows 3.1® itself to allow multi-tasking.

3.2.5. Procedure WriteLLPBlock

The *WriteLLPBlock* procedure accepts a string parameter which has to be sent (written) to the HP4745A. *WriteLLPBlock* builds the count block, sends it to the HP4745A and handles the response. The next step is to construct the low level protocol data block. This includes calculating the CRC for the low level protocol data block. Note that the sequence of the two bytes making up the CRC value (a word) has to be swapped for the HP4745A. On the HP4745A the Most Significant Byte (MSB) is written first, followed by the Least Significant Byte (LSB).

3.2.6. Function CheckSum

The *CheckSum* function calculates a checksum value for the count block. Details of this function is privileged and can be found in section 4.7.1 *The Count Block* on page 176 of Appendix A.

3.2.7. Function GetCount

The count block is basically just a text string, containing digits from 0 to 9, and letters from A to F (valid tokens for a hexadecimal number). No numeric value is automatically associated with the string. A hex to decimal conversion has to be performed before the value of the count block is obtained.

The C/C++ runtime libraries contain the *scanf()* family of functions which will, amongst others, read a text string containing a valid hexadecimal number, and return it as a numeric value. (Barkakati, 1989 : 459 - 462). The Object Pascal runtime libraries do not supply such a function by default.

In order to avoid having to raise numbers to certain powers, the calculations were performed and the values stored in an array.[4] (This approach worked here, since we knew beforehand exactly how many digits can be present in the input string.) The array is populated with 1 (16^0), 16 (16^1), 256 (16^2) and 4096 (16^3).

[4] The Object Pascal Run Time Library contains a procedure *IntPower*, that will raise x to the power of y. In order to avoid the overhead of repetitively calling *IntPower*, the result of raising x to y can be calculated beforehand. This can easily be done, since the number of y's are small and known beforehand.

A small loop is used to convert the hexadecimal string to a numeric value. The numeric value of each (hexadecimal) digit is determined by subtracting the ASCII code of the digit from the ASCII code for 0. If the digit is in the range A - F, the ASCII code of the digit is subtracted from the ASCII code for A. 10 is then added, since A_h represents 10_d. Once the numeric value of the digit has been established, the digit is multiplied by the corresponding value in the array of numbers (1, 16, 256, 4096).

The sum of these values represents the converted numeric value, in decimal.

3.2.8. Function CheckCount

CheckCount determines the validity of the count block by:

- Checking the value of the count block (valid values lie between 0 and 255, inclusive).

- Calculating a check digit for the checksum, extracting the check digit from the checksum and comparing the two check digits.

If the count block value is invalid, or the check digits do not match, *CheckCount* returns false.

3.2.9. Procedure CommitDataToMemory

CommitDataToMemory copies the bytes just received from the HP4745A ECG machine from the input buffer into the memory which has dynamically been allocated for this purpose. At the end of a successful transmission procedure, this buffer will contain a complete copy of a digital ECG, ready for further processing.

3.2.10. Procedure ComPort1TriggerAvail

The *ComPort1TriggerAvail* function is an event handler invoked by the *TComPort* VCL component whenever data have arrived at the COM port and are ready to be dispatched. This function reads the characters from the COM port buffer and places these characters into the buffer declared by the application. The *ComPort1TriggerAvail* function ensures that the *buffer full* and *buffer resume* properties of the *TComPort* VCL component are set up correctly. This can be done by either setting the values in the Object Inspector, or assigning the desired values to the properties of the *TComPort* object at runtime. In the case of the *ComPort1TriggerAvail* function, the properties were set in the Object Inspector.

3.3. Decompression (Decoding)

The digital ECG data have to be decoded and manipulated before it can be used at all by other applications. The decoding and calculation that have to be performed will now, in part, be described. Please refer to section 5.16 *Decompression Detail* on page 209 of Appendix A for more information.

3.3.1. Procedure Decode

Technical details of the *Decode* procedure is documented in section 5.16.1 on page 209 of Appendix A. Two problems encountered during the coding of the *Decode* procedure deserve further discussion here.

Consider the case where access to the two individual bytes forming a word is needed.[5] This can be facilitated through the use of a variant record. Object Pascal typically requires a variable (tag field) used to specify which variation is used.

Swan describes a free union as a case variant record without a tag field. The case part of the record declares an unidentified type, a nameless entity that occupies no space. (Swan, 1991 : 130). In this way the different bytes that constitute a word can be accessed, as well as the actual word value. This technique works well where the individual bytes of a word value have to be manipulated to form the word value.

The boolean free union used in the *Decode* function has been declared as follows:

Code Snippet 8-3: The Boolean Free Union

```
Integers = record
     CASE Boolean of
        True : ( sInteger : SmallInt);
        False: ( uInteger : Bytes );
  end;
```

The record type *Bytes* has been declared as follows:

Code Snippet 8-4: A Record with two Byte Fields

```
  Bytes = record
     lsb : Byte;
     msb : Byte;
  end;
```

The second problem that deserves attention is that of dynamic array allocation, since such dynamic allocation lies at the heart of the *Decode* procedure. According to Swan (Swan, 1991 : 208) a common complaint about the Pascal language is the inability to dynamically resize an array during runtime. Index ranges must be constants, and therefore array sizes are fixed when the program is compiled. This limitation is also present in Object Pascal, the Pascal implementation used in Borland® Delphi™ 3.0.

5 Swan (Swan, 1991 : 875, 898 - 899) describes the use of the *Hi* function to return the MSB of a word, and the use of the *Lo* function to return the LSB of a word. The result of *Hi* and *Lo* is always a byte in the range 0 to 255.

What is basically needed in the *Decode* procedure is a two dimensional array with *x* number of rows (where *x* can only be determined at runtime) and 3 columns. (Remember that there are 3 leads which are simultaneously used (in some cases) for calculating of actual lead data.)

Swan describes a very interesting technique (Swan, 1991 : 208 - 210) that can be used to dynamically allocate a one dimensional array. The steps are surprisingly easy:

1. Declare the data type. Also declare an array of this data type, but be sure to declare the indices as [0..0]. (Although this appears to be a typographical error, it is not.) Such a declaration results in an array with only one index entry, [0]. With range checking off (default state of the compiler) it is possible to reference indices outside of the declared boundaries. It is up to the programmer to ensure that the variable referenced at the specified location exists.
2. Declare a pointer of the array type.
3. Use the *GetMem* procedure to allocate a specific number of bytes on the heap and assign the address of the first byte to the pointer declared in the previous step. (The *SizeOf* function returns the number of bytes occupied by the data type.)
4. Deallocate the space on the heap by using the *FreeMem* procedure with the same arguments as the *GetMem* procedure.

This idea is expanded somewhat in the *Decode* procedure. In order to create a two dimensional array, the data type declared in step 1 is a one dimensional array with three rows. Here follows a brief example of what is used in the *Decode* procedure:

1. Declare the new data types. Since a standard data type is used in *Decode*, no new type declaration is needed. We will, however, declare types for the arrays so that these types can later be used for pointer variable declarations. Note the declaration of the array, which has indices that are both set to 0.

Code Snippet 8-5: Type Declarations for Dynamic Arrays used in Decoding

```
type
     DynRow = array[ 0..2 ] of smallint;
     DynArr = array[ 0..0 ] of DynRow;
```

2. Declare a pointer of the array type.

Code Snippet 8-6: Pointer to Dynamic Array

```
var
     DynamicArray : ^DynArr;
```

3. Dynamically allocate memory using the *GetMem* procedure.

Code Snippet 8-7: Dynamic Memory Allocation

```
GetMem( DynamicArray, SizeOf( DynRow ) * SamplesInChannel );
```

(Note that in the *Decode* procedure such a dynamic memory allocation is only performed once per lead group. Each of the three leads in the same group has the same number of samples.) The array can now be used simply by dereferencing the pointer *DynamicArray*. An example of this could look as follows:

Code Snippet 8-8: Using the Dynamically Allocated Array

```
DynamicArray^[ Row, Column ] := Value;
```

4. Deallocating the array using the *FreeMem* procedure.

Code Snippet 8-9: Releasing the Dynamically Allocated Array

```
FreeMem( DynamicArray, SizeOf( DynRow ) * SamplesInChannel );
```

3.3.2. Procedure BinDump

BinDump is used primarily for debugging purposes. The binary equivalent of each byte is constructed by ANDing the value of the byte with the value 80_h (1000000_b) and writing either 1 or 0 to a string, depending on the result. The value is shifted left 8 times in order to determine the values of the 8 bits contained in the byte.

3.3.3. Function ScanBits

The compressed data found in the digital ECG data stream represent a variable length stream of bits. This poses some difficulty, as the smallest unit easily dealt with is normally one byte. *ScanBits* determines the value of the bit being dealt with and returns the value as a byte so that it can easily be used by the rest of the program. Examination is done by ANDing the value of each byte with the value 80_h (1000000_b). (This has the effect of only returning the value of the most significant bit.) After the comparison, the value is shifted left once, effectively moving the next bit into bit 8 and inserting a 0 at the right end of the value. The function furthermore keeps count of the number of bits examined and advances to the next byte in memory for every 8 bits used. This relieves higher level modules of keeping track of the bit stream. *ScanBits* feeds *MakeValue* with bits for decompression (decoding).

3.3.4. Function MakeValue

MakeValue lies at the heart of the reconstruction of the compressed data. The technical detail of the *MakeValue* function is documented in section 5.16.2 on page 213 of Appendix A.

One of the obstacles which had to be overcome in the *MakeValue* function was the correct reconstruction of compressed data into two's complement form, since numeric variables are stored in two's complement form in the memory of the PC.

The *MakeValue* function takes as parameters two byte values (containing the scanned bits from the compressed data stream) as well as the bracket code preceding the bits. *MakeValue* returns a signed 16-bit integer value, which is the decoded value.

In order to better understand the techniques implemented in the *MakeValue* function, a brief discussion of the internal representation of numeric values is in order.

On the PC, a short integer (also known as a small integer or word) is made up of two bytes. Each byte consists of 8 bits. This allows 16 bits of information to be stored.

According to Morse (Morse, 1982 : 14), the 80x86 family of microprocessors stores the least significant byte (LSB) first, and the most significant byte (MSB) second. This is illustrated as follows:

Short Integer

LSB	MSB

As an example, consider the value 7F30$_h$. This is stored as follows:

Short Integer

LSB	MSB
30	7F

As mentioned, each byte consists of 8 bits. Bit numbering starts at 0 (the least significant bit) from the right. This can graphically be represented as follows:

msb							lsb
b7	b6	b5	b4	b3	b2	b1	b0

Positive numbers and zero can easily be described using binary notation. Negative numbers introduce more complexity: an additional mechanism is needed to indicate the sign of the number. Using the leftmost (most significant bit for the sign is called *sign-magnitude* representation (Morse, 1982 : 5). Special arithmetic rules are needed! Consider the following example:

Subtract +1 from 0 (expecting a result of -1).

$$\begin{array}{rrrr} & 0000 & 0000_b & 0 \\ - & 0000 & 0001_b & +1 \\ \hline & 1111 & 1111_b & -127 \end{array}$$

(Morse, 1982 : 5)

We find that with the sign-magnitude representation the answer is actually -127!

A signed-number system is needed that can perform the same binary arithmetic on signed as well as unsigned numbers. In such a number-system 1111 1111$_b$ should represent -1 and not -127. Subtracting +1 from -1 should yield -2.

$$\begin{array}{rlll} & 1111 & 1111_b & -1 \\ - & 0000 & 0001_b & +1 \\ \hline & 1111 & 1110_b & -2 \end{array}$$

(Morse, 1982 : 5)

This representation is called two's complement representation. Tanenbaum (Tanenbaum, 1984 : 445 - 446) suggests that a negative number can be expressed in its two's complement form by following a two step process. All ones are replaced by zeroes, and zeroes are replaced by ones. One is then added to the result. If any carry bit is generated during the addition operation, the additional bit (left most carry bit) is ignored.

Properties of the two's complement form

- Binary additions and subtractions will yield the correct two's complement result.
- The most significant bit of a positive number is 0.
- The most significant bit of a negative number is 1. (The msb serves as a sign bit.)
- The sign of a two's complement number can be changed by negating the value of each bit, and adding 1.
- When an 8-bit two's complement number is extended to 16 bits (or more) the bits on the left side of the 8-bit number will have the same value as the original sign bit. This is called sign extending.

(Morse, 1982 : 5 - 6).

3.3.5. Function ScanMem

ScanMem receives as input a character pointer and the number of characters to process. It proceeds by copying the required number of characters from memory into a string and converting this string into an integer value, which is returned.

3.4. Calculation

As soon as all three columns (leads) for each lead group have been decoded, the calculations needed to produce the actual lead data are performed. A second two dimensional dynamic array is declared for this purpose. The reason for this step is to facilitate the calculation. While some leads are already correct and can directly be written to file, others need to be calculated. In some cases only one of the three leads is known and the other two leads need to be calculated.

The formulæ for calculating the actual lead values are described in Table A-22: Smoothing Formulæ on page 202 of Appendix A. At the end of the calculation of the actual lead values, the results are written to a disk file.

The structure of such a data file is important and deserves discussion. One of the aims of this research is to provide a graphics tool that can re-create an ECG on screen or on paper. Such a tool should provide the capability to draw the same leads of different ECGs on a screen/page for recognition of trends by a cardiologist. To allow random access to lead data in the file, the decoded data file was structured containing header and lead portions (as illustrated by Table B-2: Structure of a Hearts 32 ECG data file on page 219 of Appendix B).

Although this organisation seems to work well, there were some implementation difficulties associated with the creation of such a data file. When declaring a file of type *Text* in Object Pascal it is not possible to use the *Seek* procedure. The *Seek* procedure is used to position the internal file pointer within a file. In this research, the *Seek* procedure was used mainly for two reasons:

1. When the data file is created, the lead map needs to be written to the new data file before any lead data can be written. Since the data contained in the lead map are only calculated as the processing of the digital ECG data stream progresses, it is impossible to write the correct values for the lead map with one pass to the new data file. During calculation of leads, the lead map information is gathered and stored in an array. At the end of the process, the internal file pointer is reset back to the beginning of the file and the lead map is overwritten, this time with the correct values. Note that the lead data stay intact.

2. The graph unit has to gain direct access to user-specified leads. By reading the initial lead map this is a trivial exercise. (Such direct access speeds up processing, since it is not necessary to read all values even if they are not needed.) In this case the *Seek* procedure is used to move the internal file pointer to the starting value of the desired lead.

Swan (Swan, 1991 : 957) suggests that a *file of Char* must be declared to use *Seek* in text files. The *Read* and *Write* procedures can only be used with files of type text. To write to any other file (such as *Char*), the *BlockRead* and *BlockWrite* procedures must be used. Please refer to Chapter 6 for a complete discussion on why an ASCII data format was chosen.

The data elements stored in a decoded data set (*Hearts 32* format) are documented in Table B-3 (page 220) of Appendix B, with examples in Table B-4 (page 221) and Table B-5 (page 222).

The offset information in the lead map array can only be updated after the data have actually been written to disk. The reason for this is (as has already been mentioned) that the individual data values are not of the same length. The following method is used to update the offset information:

Code Snippet 8-10: Final Update of Lead Map Information

```
ASCIILeadMap[ 1 ].Offset := MapSize + 3;
for i := 2 to Leads do
begin
   ASCIILeadMap[ i ].Offset := ASCIILeadMap[ i - 1 ].LeadLength +
                               ASCIILeadMap[ i - 1 ].Offset;
end;
```

The offset for the first lead can very easily be calculated: it directly follows the lead map and is calculated by summation of the size of the number of leads field and the size of the lead map. We do not need to add one to the sum, since all offsets are zero-based. For subsequent leads, the offset is calculated as the sum of the lead length of the previous lead and the offset of the previous lead.

4. Graphic Display Module

It has been found that the font size setting in Windows® 95 (small/large fonts) has a profound effect on the way that the forms in a Delphi™ application are presented. It seems that the font size setting tends to influence the size of the form when auto scaling is enabled. *Hearts 32* (and the *Hearts 32* ECG Browser) was developed for high resolution use; 1024 x 768 pixels, using large fonts.

The *Hearts32* ECG Browser has been developed as DLL, in order to allow enhancements to be made to the browser, but at the same time protecting the *Hearts 32* database against such changes.[6] The second reason for this design was the fact that the Multiple Document Interface architecture (section 4.1 on page 137) is best suited to the browser (rather than the Single Document Interface (SDI) used for the *Hearts 32* application).

Access to a *Hearts 32* ECG data set is realised by *Hearts 32* itself; the browser does not directly interact with the *Hearts 32* database. This is illustrated in Figure 8-2 below:

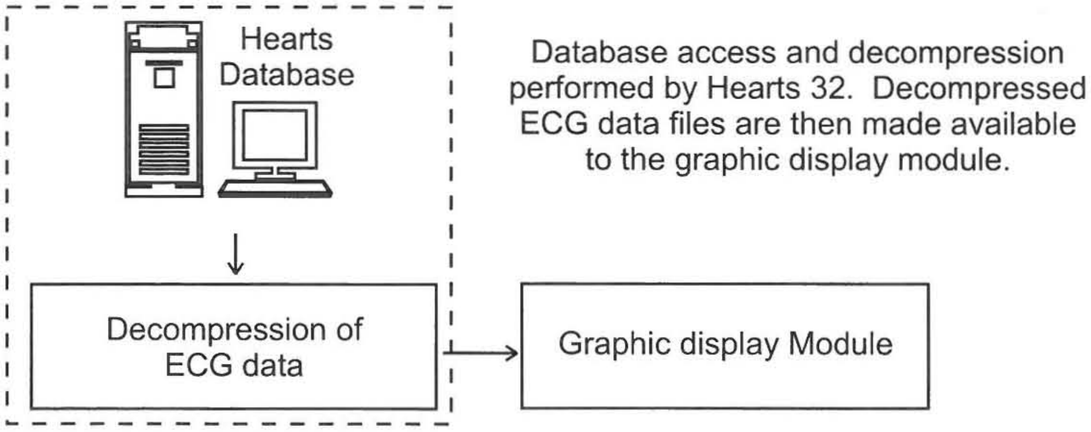

Hearts Database
Database access and decompression performed by Hearts 32. Decompressed ECG data files are then made available to the graphic display module.
Decompression of ECG data
Graphic display Module

Figure 8-2: Graphic overview of the Hearts 32 ECG Browser

6 This means that it will not be necessary to re-compile the complete *Hearts 32* application, since updates are localised to the browser software only.

The *Hearts 32* ECG Browser comprises of four Object Pascal unit files:

Table 8-2: Object Pascal units found in the Hearts 32 ECG Browser

Unit	Short overview
EKG10P	Responsible for frame management.
ChildWin	Responsible for actual display of an ECG, together with tools such as crosshairs, callipers, gridlines and zooming.
DrawLD10	Responsible for superimposing of different leads of the same ECG.
About	Responsible for the Help|About dialogue box.

Although the ECG graphic display screen appears to be one graph, the graphic display actually consists of 14 different *TChart* components, which, in turn, each contain one *TFastLineSeries* component. The individual *TChart* components are placed next to each other, forming what appears to be one graph.

This configuration meant that items such as crosshairs, callipers, gridlines and zooming had to be copied for 14 different charts.

4.1. The Multiple Document Interface

The browser must be capable of displaying *n* ECGs simultaneously, in order to allow the cardiologist to compare different ECGs for the same patient. The MDI approach delivers a solution that is already familiar to most users of word processing packages. MDI allows more than one document to be opened simultaneously.

Cantù (Cantù, 1997 : 736) describes MDI applications as follows: "MDI applications are made up of a number of forms that appear inside a single main form."

The different windows involved in an MDI application can be summarised as follows:

Table 8-3: MDI Window Types

Window type	Description
Frame window	The application's main window. It has a caption, menu bar, and system menu. Minimize, Maximize and Close buttons appear in its upper-right corner. The blank space inside the frame window is known as its *client area* and is actually the client window.
Client window	The manager for MDI applications. The client window handles all MDI-specific commands and manages the child windows that reside on its surface - including the drawing of MDI child windows. The client window is created automatically by VCL when a frame window is created.
Child window(s)	MDI child windows are the actual documents - text files, spreadsheets, bitmaps and other document types. Child windows, like frame windows, have a caption, system menu, Minimize, Maximize, and Close buttons, and possibly a menu. It's possible to place a help button on a child window. A child window's menu is combined with the frame window's menu. Child windows never move outside the client area.

(Pacheco & Teixeira, 1996 : 312 - 313).

The functions and procedures found in the EKG10P unit work together to form the frame window of the *Hearts 32* ECG Browser. These include:

4.1.1.Function DrawLeads

The *DrawLeads* function is the only exported function in the DRAWECG.DLL file. It serves as the hook between *Hearts 32* and the *Hearts 32* ECG Browser. *DrawLeads* is responsible for the assignment of the parameters passed from *Hearts 32* to global variables, and also for the instantiation of the main form of the browser. This can be illustrated as follows:

Code Snippet 8-11: Dynamic Creation of the Frame Window

```
NewParmList := ParmList;
Application.Initialize;
Application.CreateForm( TMainForm, MainForm );
Application.Run;
```

The parameter list passed to the *Hearts 32* ECG Browser is somewhat complicated and deserves a short discussion. Put in English, the patient's name, sex and date of birth, together with a list of the ECG data file names (and the date and time of recording of each ECG) need to be sent to the browser (from *Hearts 32*).

Since there is no way of determining the number of data file names before they are passed as parameters, a different parameter passing technique is needed. Cantù (Cantù, 1997 : 170) notes that "Unlike C, a Pascal function or procedure always has a fixed number of parameters. However, there is a way to pass a varying number of parameters to a routine using an open array. The basic definition of an open array parameter is that of a typed open array. This means you indicate the type of the parameter but do not know how many elements of that type the array is going to have."

An open array parameter would have been perfect if an array with file names were to be passed. Since other information is also needed, this method has been abandoned in favour of passing one complex (record) variable. The structure of this variable is defined in a few steps as follows:

Code Snippet 8-12: Record used for Parameter Passing between Hearts 32 and the Hearts 32 ECG Browser

```
type
   TAr20 = Array[ 0 .. 20 ] of Char;
   TAr80 = Array[ 0 .. 80 ] of Char;

   TEDFArray = Record
      FileName : TAr80;
      Date     : TAr20;
      Time     : TAr20;
   end;

   EDF = Array [ 0 .. 0 ] of TEDFArray;

   TParmList = Record
      PatientName : TAr80;
      DateOfBirth : TAr20;
      Sex         : Char;
      FileCount   : Integer;
      ECGDataFiles: ^EDF;
   end;
```

Types *TAr80* and *TAr20* were created as zero-based character arrays, so that parameters of this type can be compatible with ASCIIZ strings. The use of the Object Pascal type *String* causes problems when writing DLLs.[7] The same approach for

[7] Important note about DLL memory management: ShareMem must be the first unit in your library's USES clause AND your project's (select View-Project Source) USES clause if your DLL exports any procedures or functions that pass strings as parameters or function results. This applies to all strings passed to and from your DLL;

dynamic array allocation as described in section 3.3.1 on page 127 has been used to dynamically allocate the space needed for the ECG files.

4.1.2. Procedure TMainForm.FormCreate

Additional processing during the form creation includes enabling of hints during the program execution, and also the updating of menu items (enabling/disabling items, managing tick marks etc.).

4.1.3. Procedure TMainForm.ShowHint

ShowHint passes hint strings to the status bar at the bottom of the window. Note that the *Hint* property of a form can contain both short and long hints. The different hints are separated from each other by a pipe (|) character.

4.1.4. Procedure TMainForm.CreateMDIChild

CreateMDIChild creates a new MDI child window in which a new document (ECG in this case) will be managed.

4.1.5. Procedure TMainForm.FileCloseItemClick

The currently active MDI child window is closed using the *Close* method of the MDI child window. A check is performed to determine whether MDI child windows are present before *Close* is called. This is done by using the *ActiveMDIChild* procedure. If no MDI child windows are available, *ActiveMDIChild* returns nil, otherwise the active MDI window is returned.

4.1.6. Procedure TMainForm.FileExitItemClick

The frame window is closed by calling the *Close* method of the *Application* object.

even those that are nested in records and classes. ShareMem is the interface unit to the DELPHIMM.DLL shared memory manager, which must be deployed along with your DLL. To avoid using DELPHIMM.DLL, pass string information using PChar or ShortString parameters. (Pacheco & Teixeira, 1996 : 676).

4.1.7.Procedure TMainForm.WindowCascadeItemClick

MDI child windows are cascaded by calling the *Cascade* method of the frame form (window).

4.1.8.Procedure TMainForm.WindowTileHItemClick

MDI child windows are horizontally tiled by setting the *TileMode* property of the frame form (window) to *tbHorizontal*, and calling the *Tile* method of the frame form.

4.1.9.Procedure TMainForm.WindowArrangeItemClick

Minimised MDI child windows are arranged by calling the *ArrangeIcons* method of the frame form.

4.1.10.Procedure TMainForm.WindowMinimizeItemClick

All MDI child windows are minimised by looping through the list of MDI child windows, and setting the *WindowState* property of each MDI child window to *wsMinimized*. The number of MDI child windows can be determined by querying the *MDIChildCount* property of the frame form.

4.1.11.Procedure TMainForm.UpdateMenuItems

Menu items as well as speed buttons are enabled or disabled, depending on the availability of a MDI child window. It does not make sense, for example, to allow the user to close a file if no file was opened to begin with! This can be illustrated as follows:

Code Snippet 8-13: Enabling/disabling menu items and speed buttons

```
FileCloseItem.Enabled := MDIChildCount > 0;
```

4.1.12.Procedure TMainForm.FormDestroy

Care must be taken that, before the frame form is destroyed, the link to the *ShowHint* procedure is disabled. If this is not done, the frame form will be destroyed, resulting in a dangling pointer which will cause a program crash.

4.1.13.Procedure TMainForm.sbZoomButtonClick

To enable the zoom feature, the *AllowZoom* property of each *TChart* object needs to be set to *True*. *sbZoomButtonClick* makes use of the Run-Time Type Information (RTTI) operator *as* to ensure a safe type cast. Cantù elaborates on the use of the *as* cast as follows: "The difference between the traditional cast and the use of the as cast is that the second one raises an exception if the type of the object is not compatible with the type you are trying to cast to." (Cantù, 1997 : 233).

4.1.14.Procedure TMainForm.sbCalliperButtonClick

When the calliper tool has been enabled, the *OnClick* event handler of the *TChart* object points to the *RSClick* procedure. Disabling the calliper tool sets the *OnClick* event handler to *nil*.

4.1.15.Procedure TMainForm.sbRefreshButtonClick

The ECG display is refreshed by calling the *Repaint* method for each *TFastLineSeries* object.

4.1.16.Procedure TMainForm.sbCloseClick

Clicking on the Close speedbutton closes the frame form by calling the *FileExitItemClick* procedure.

4.1.17.Procedure TMainForm.sbGridButtonClick

The status of the *Grid* menu item is updated, depending on the status of the GridButton. This is accomplished by setting the value of the Checked property of the *Grid* menu item. The *Draw* method of each *TChart* object is then called.

4.1.18.Procedure TMainForm.sbDrawSameLeadsClick

Before leads can be superimposed, a new MDI child form needs to be created. After creation of this form, control is passed to the form.

4.1.19.Procedure TMainForm.sbFullSizeButtonClick

Setting the MDI child window to its maximum size can be achieved by setting the values of the top, left, height and width properties of the MDI child form.

4.1.20.Procedure TMainForm.Refresh1Click

Selecting the *Refresh* menu item will call the *sbRefreshButtonClick* procedure to complete the refresh task.

4.1.21.Procedure TMainForm.Grid1Click

Selecting the *Draw Grid* menu item will call the *sbGridButtonClick* procedure to complete the drawing of the grid.

4.1.22.Procedure TMainForm.Zoom1Click

Selecting the *Zoom* menu item will call the *sbZoomButtonClick* procedure to complete selection/deselection of the zoom tool.

4.1.23.Procedure TMainForm.WindowTileVItemClick

MDI child windows are vertically tiled by setting the *TileMode* property of the frame form (window) to *tbVertical*, and calling the *Tile* method of the frame form.

4.1.24.Procedure TMainForm.sbPrintClick

The *sbPrintClick* procedure starts by forcing the page orientation to *poLandscape*. The shape of the cursor is set to an hourglass (*crHourGlass*). *Printer.BeginDoc* prepares the print job. The *PrintPartial* method of the *TChart* object is used to allow individual placing of *TChart* objects on the printed page, at program-supplied co-ordinates. *Printer.EndDoc* finalises printing. At the end of the print job, the printer orientation as well as the cursor are restored to their old values.

4.1.25.Procedure TMainForm.StatusBarDblClick

Double clicking on the status bar will hide or display the toolbar, depending on the current setting of the toolbar visibility.

4.1.26.Procedure TMainForm.Open1Click

The *Open1Click* procedure loads the ECG data sets as listed in the parameter list (Code Snippet 8-12 on page 139), each into each its own MDI child window. During loading of the data set, the name of each data set is displayed in the status bar at the bottom of the screen. The caption of each MDI child window is set to the patient name, age in years and sex, as well as the date and time on which the ECG was recorded. The patient age is determined by subtracting the birth date from the ECG recording date, and dividing the answer by 365.25.[8] Only the integer portion of the calculation is displayed.

4.1.27.Procedure TMainForm.sbOpenButtonClick

Selecting the *Open* menu item will call the *sbOpenButtonClick* procedure to complete the loading of the ECG data sets.

4.1.28.Procedure TMainForm.Print1Click

Selecting the *Print* menu item will call the *sbPrintClick* procedure to complete the printing of the active MDI child window.

4.1.29.Procedure TMainForm.Superimpose1Click

Selecting the *Superimpose* menu item will call the *sbDrawSameLeadsClick* procedure to facilitate superimposing of leads for the same ECG.

4.1.30.Procedure TMainForm.Help1Click

The Help|About dialogue box is displayed as soon as the *AboutBox* has been instantiated.

4.2. Managing the contents of a child window

The procedures and functions contained in the *ChildWin* Object Pascal unit concern themselves with the management of the contents of the MDI child window. This

[8] The HP4745A Cardiograph does not support dates past 31 December 1999. This means that the age calculation will have to be revised, if the HP4745A Cardiograph is used past 31/12/1999.

includes items such as loading ECG data sets and displaying the ECG graphs, as well as drawing crosshairs.

As mentioned previously, a total of 14 *TChart* objects are present in the unit. Due to this fact, some functions had to duplicated. Since the contents of these functions stay the same, save for a change in an array subscript, only one of these functions will be listed and discussed.

4.2.1. Procedure TMDIChild.FormClose

FormClose closes the MDI child window by setting the *Action* parameter of the *OnClose* event to *caFree*. The default closing behaviour for MDI child windows is to minimise, rather than close. This is controlled by a parameter of the *OnClose* event called *Action*. By default, *Action* has the value *caMinimize*. By setting *Action* to *caFree*, the application is forced to destroy the window and free its resources on closing.

4.2.2. Procedure TMDIChild.RSClick

RSClick implements the calliper functionality by recording the time and voltation on an initial mouse click, waiting for a second mouse click, recording the time and voltation and then calculating the difference between these values.

4.2.3. Procedure TMDIChild.Exit1Click

Exit1Click closes the MDI child window by calling its *Close* method.

4.2.4. Procedure TMDIChild.AfterDrawValues

AfterDrawValues is responsible for drawing the gridlines after the *TChart* object has drawn itself.

4.2.5. Procedure TMDIChild.ScanFile

The *ScanFile* procedure is used to read items such as lists of numbers from an ASCII data file. Each value is separated from the others by a space. *ScanFile* reads the contents of the data file a character at a time, terminating as soon as a delimiting character (such as a space, comma or tab) has been read. Although character operations sound painfully slow, the file I/O buffering techniques provided by the operating system, as well as the increased access speeds of modern hard disks speed up processing considerably.

4.2.6. Procedure TMDIChild.RSMouseMove

RSMouseMove is responsible for crosshair management. The *DrawCross* procedure (which is local to *RSMouseMove*) performs the actual drawing of the crosshair. The crosshair is managed by determining whether a crosshair was already drawn or not. If this is true, the previously drawn crosshair is effectively erased by drawing it in its previous position. This is achieved by setting the pen mode to *pmXor*. Drawing the crosshair in a new position with a pen mode set to *pmXor* will display the crosshair in a different colour.

The *GetCursorValues* method of the *TChart* object enables the determination of the actual values under the mouse cursor (X,Y coordinates) at any given point. If the result of the *GetCursorValues* was to be used directly, it would make no sense. The numeric values read from the *Hearts 32* ECG data set do not represent values in millivolts or time directly. It is known that the total time per lead is 2.5 seconds. We have roughly 630 data points per lead. 2.5 / 630 = 0.004 seconds per data point. This means that for every X value, the associated elapsed time can be expressed as X multiplied by 0.004.

An analysis of 740 calibration pulses showed that the average numeric representation of 1 mV is 199.56 (with a CV% of 0.56%). In order to ease the calculation, this figure has been rounded to 200. To determine the value of each Y in millivolts, the value of Y has to be multiplied by 0.005. An example of the calibration pulse can be found in Figure 3-20 on page 48 of Chapter 3 (Item F). Please refer to Table 3-9 on page 46 on Chapter 3 as well.

The result of these calculations is displayed in the status bar at the bottom of the window, as found in Figure 7-3 and Figure 7-4 on page 115 of Chapter 7.

4.2.7.Procedure TMDIChild.FastLineSeries10AfterDrawValues

The *FastLineSeriesNAfterDrawValues*[9] functions are needed to facilitate correct handling of the crosshairs found in each *TChart* object. These functions set a flag to allow a new set of readings to be taken, as needed.

4.2.8.Procedure TMDIChild.FormCreate

The *FormCreate* procedure starts by reading the ECG data set from the ASCII file supplied as a string parameter. The first step is to determine the size of the lead map, dynamically allocate enough memory to store such a lead map, and to read the lead map from the ECG data set. Subsequently, lead data are read by looking up the offset of each lead in the lead map, moving to the correct offset in the data file and reading the data. Data points are added directly to the correct *TFastLineSeries* object as they are read.

The determination of the maximum and minimum values follows next. This ensures that all graphs are scaled equally. The ASCII data file is now closed and memory dynamically allocated for the lead map, is freed.

4.2.9.Function TMDIChild.FindSubscript

FindSubscript returns the subscript of the item in the *MouseStuff* array by searching the entries in the *MouseStuff* array for the title of the chart.

4.2.10.Function TMDIChild.FindLeadOffset

FindLeadOffset allows access to the lead offset in the header data of the *Hearts 32* ECG data set by searching the lead map for the supplied lead identification, and returning the subscript where the ID is found.

4.3. Superimposing of selected ECG leads

The *DrawLD10* Object Pascal unit declares a new form that is used to select and display a selection of leads on the same graph. The selection process is facilitated

[9] The range for *N* is between 1 and 14, inclusive.

through the use of a *TListBox* component. The *TChart* object contains 12 *TFastLineSeries* objects.[10]

The initial value of the *Active* properties of these *TFastLineSeries* objects are set to *False*, effectively hiding the series. Each selected lead is copied from its corresponding *TSeries* and *TChart* objects contained in the *EKG10P* Object Pascal unit. This saves time and effort to re-load the information from the ECG data set. The series *Title* and *Active* properties are updated, and the series are displayed.

In order to ensure that the selected leads are drawn according to the same scale, the *LeftAxis.AutomaticMaximum* and *LeftAxis.AutomaticMinimum* properties of the *TChart* object are set to *False*. The *LeftAxis.Maximum* and *LeftAxis.Minimum* properties are set to the calculated maximum and minimum values.

The colours used are of type *TColor*, and include *clRed, clGreen, clBlue, clBlack, clTeal, clOlive, clFuchsia, clYellow, clNavy, clMaroon, clLime* and *clGray*.

10 There are 12 leads in the ECG data set eligible for superimposing.

Chapter 9

Conclusion

1. Introduction

The hypothesis of this study as outlined in Chapter 1 can be summarised as follows:

To facilitate the digital availability of electrocardiogram tracings recorded with a HP4745A PageWriter II Cardiograph.

The milestones needed in order to reach this goal were listed in the Research Method in Chapter 2. Obtaining the technical documentation from the Hewlett-Packard Company took a few months.[1] The *Hewlett Packard Diagnostic Cardiology Digital Transmission Protocol* covered the information needed for digital transmission. Nothing was handed to the reader on a plate.

Parts of the document necessitated further research using other information technology textbooks. The CRC calculation routine, as well as the modulo-16 calculation of checksums were not clearly documented in the protocol text. Another grey area (which incidentally could not be solved) is that of the position bits[2] found in the digital ECG data. Even without these position bits and their effect on the data, the resulting ECG data appeared to be fine.

The transmission protocol discussed the smoothing and compression methods used to create the digital data set, but no hint was given as to how to decompress these data. These procedures had to be devised from scratch. Since the compression and decompression methods perform their tasks on the bit level, some reading had to be performed in order to get acquainted with the internal representation of numbers in the 80x86 chips.

[1] Negotiations for acquisition of the document started around 20 March 1996. The document was received on the 18th of June 1996.

[2] Discussed in section 5.14.2 Position Bits on page 198 in Appendix A.

2. Highlights

Some of the most important highlights experienced during the life of the research project include:

2.1. *First communications (HP4745A to HP4745A)*

The very first attempt at communications was performed by connecting two HP4745As to each other, and to transmit an ECG from one machine to the other.[3] In the beginning this was rather difficult, since the menu structure and controls of the HP4745As were still virgin territory. These tests were carried out using the original cables from Hewlett-Packard.

2.2. *Construction of the RS-232 communications cable*

The original cables[4] supplied with the HP4745A PageWriter II Cardiograph are intended to either connect the HP4745A to a modem or to another HP4745A. On enquiry, Hewlett-Packard in Cape Town, South Africa, confirmed that a cable to connect the HP4745A to a PC would cost ≈R 1,900.00.

The cable diagrams supplied in the transmission protocol document did not illuminate the structure of the planned cable either. After careful examination of the two cables mentioned in the previous paragraph, a wiring diagram for a single cable was constructed.

Great was the excitement when a fully functional cable, connecting the HP4745A to a PC, was completed for around R 25.00.[5] Even some soldering skills were acquired, and the burnt fingers could not dampen the excitement.

[3] The first successful transmission was performed on the 20th of June 1996.

[4] Details of these cables can be found in the section "Cable Configuration" in Appendix A.

[5] This milestone was reached on the 22nd of July 1996.

2.3. First digital capture of a conversation between two HP4745As

At this point in time the text of the transmission protocol did not make much sense.[6] It was deemed best to try and determine the exact contents of the digital conversation between the two HP4745As. To facilitate this step, a RS-232 communications monitor was needed. The Computer Centre of the University of the Orange Free State had such a device, and access to it could be arranged. The biggest problem was the fact that the results of the probe could not be stored in a way which would facilitate later examination and experimentation.

For this reason, an inexpensive PC-based RS-232 communications monitor was needed. A search on the Internet yielded RS232 Version 1.01.[7] RS232 V1.01 basically eavesdrops on the conversation between two communicating devices. All that is needed is the RS232 V1.01 software, and a simple RS-232 cable as specified in the documentation.

First attempts at recording the digital conversation failed, since the structure of the RS232 V1.01 cable interfered with the initial communications tests of the HP4745A. As soon as this was identified and the problem corrected (by simply cutting through two cables!), the recording was completed.[8]

2.4. Initial attempts at analysis of the digital ECG data

Initially the hexadecimal dump of the RS232 V1.01 program was manually interpreted and examined in an attempt to better understand the text of the communications protocol. It soon became apparent that this would be a mammoth undertaking, something at which a computer would excel.

The first attempts at programmatic analysis of the recorded digital conversation were extremely cumbersome, since the data were not in native format. For a start each byte was represented by its hexadecimal ASCII code! Another problem was that all the

[6] The transmission protocol had, in fact, to be studied several times in detail before the different parts fell into place.

[7] By Michael Ring, Ring Development, 10750 108th Ave. N., Maple Grove, Minnesota, 55469, USA.

[8] The first recording of such a digital conversation was successfully performed on the 26th of July 1996.

control characters used by the protocol to facilitate the transfer of information were still present in the file.

These extraneous characters did, however, play an important part in better understanding the text of the transmission protocol.

2.5. *Manual decompression and calculation of ECG data*

The transmission protocol described the transmission process in a fair amount of detail. The decompression of the data was not mentioned at all. In the light of this it was decided to start off with test runs at data decompression. There would be no point in transmitting data that could not be decompressed and interpreted.

For lack of a complete understanding of the decompression process, an initial manual decompression was performed.[9]

These results were entered into an MS-Excel 5.0 spreadsheet and the resulting graph held great promise. A close resemblance between the test ECG and the graph could easily be seen.

2.6. *Initial graphing with MS-Excel of the decoded data*

The sheer volume of the data that had to be decoded, made it impossible to attempt such an operation by hand. The knowledge acquired during the manual decode step was sufficient to write a Borland® C++ 3.1 program that could read a binary ECG file (HP4745A format) and return a file with a set of numbers representing the data.

These data were entered into MS-Excel 5.0 for graphing. Some calculations still had to be performed, and MS-Excel was used for this.[10] By manually comparing the MS-Excel graphs with the test ECG, an error was discovered in the decoding process.

[9] Started towards the end of July 1996, and ended on the 29th of August 1996. A meticulous task which would have driven anyone blind and insane, provided that they were mad enough to stick to it!

[10] The first correct set of graphs were produced on the 18th of September 1996.

2.7. The first RS-232 communications program (Borland® C++ 3.1)

Between the creation of the decoding program mentioned in the previous section and the writing of the communications program, some important decisions had to be made about the way in which the RS-232 communications were to be implemented. Enough textbooks and experience were at hand for coding the RS-232 communications functions from scratch.

This did not seem like a viable option, since the point of the exercise was not to write yet another RS-232 communications library. A suitable RS-232 communications library (completely written in ANSI C) was located.[11] The immediate problem was solved.

Work on the RS-232 communications program lasted nearly three weeks.[12] Initial communications attempts failed. After days of debugging, the reason for the failure was attributed to the fact that the buffer of the communications port overflowed due to the inability of the RS-232 communications library to operate at such high baud rates (19,200 baud). The conclusion was made that the fact that the communications library was written entirely in C slowed down the servicing of interrupts (due to high overheads associated with parameter passing using the stack with function calls). Should the library be rewritten in Assembler, higher throughput was to be expected.

Nevertheless, a satisfactory answer was obtained and the baud rate was simply lowered to 9600 baud, resulting in success.

2.8. Capturing data for statistical analysis

In order to determine the optimal storage method, some data had to be captured. This activity took place in parallel with the rest of the research. To realise this, a special version of the communications program was prepared in Clipper 5.2. The CA Clipper Tools library version 3.0 (by Computer Associates, the manufacturer of Clipper) was used to implement the RS-232 communications routines.

[11] Another product called RS232, by Chris A. Karcher, 9537 Evanston Ave. N., Seattle, Washington, 98103-3131, USA. ©1992.

[12] Started on the 8th of November 1996, with the first successful run on the 1st of December 1996.

The result of this was that the existing *Hearts* application could digitally acquire ECG Data. These data could not, however, be stored in the database. Instead, the ECG data were stored in files on the file server, awaiting database storage in the new *Hearts 32* database.

One thousand and twelve ECG data files were digitally captured between 20 December 1996 and 8 May 1997.

2.9. Development cross-roads

Although the C language seemed like a good choice for such low level work, the character based interface did not appeal. The new trend was Windows® and Graphical User Interfaces (GUI), and it did not make sense to develop a new product using an outdated interface.

At that point in time, Borland had brought their new Rapid Application Development (RAD) tool, Delphi™, to the market. The choice was largely between Microsoft Visual C (from previous experience not an attractive idea) and Delphi™.

Although the researcher was not versed in the Pascal language[13], a decision was made to move to Delphi™. This meant not only that another computer language had to be learnt, but also that a paradigm shift had to be made. Rather than traditional procedural program development, object oriented, event driven programming was to be performed.

2.10. The first RS-232 communications program (Delphi™ 1.0)

Once again, the biggest obstacle to overcome was the RS-232 communications component. It was quickly discovered that the low-level knowledge of serial communications programming in DOS would not suffice in a multi-tasking environment

[13] The development language used in Borland® Delphi™. Microsoft Visual C uses the C programming language.

such as Windows®. The Internet provided a solution in the form of the *MSComm* VCL component.[14]

As a first program in Delphi™ 1.0, the communications program came into being, downloading data in the Windows® environment at speeds of 19,200 baud.[15] This was quite surprising, when the overheads of multi-tasking and messaging in Windows® are considered. It was found that higher transmission rates do not dramatically reduce the time needed for complete data transfer.

The most probable reason for this phenomenon is the fact that half-duplex communications are used. Every packet of data that is sent to the PC must be confirmed. If a large stream of data was continuously sent, an increased transmission rate would have had a greater influence on the transmission time. In the realm of the research problem, the largest packet is limited to 256 bytes by the transmission protocol.

2.11. Windows® 95 and the 32-bit environment (Delphi™ 2.0)

Windows® 95 had by then positioned itself in the market and Borland released Delphi™ 2.0, a 32-bit version of their best selling compiler. Since no shattering progress had been made developing with the 16-bit version of Delphi™ (for this research), and also because of the claimed benefits of the new compiler, it was decided to move to the 32-bit environment.

A brief spell with a beta version of Borland's C++ Builder led to the decision to stick with Delphi™. The most important reasons for this included that the product was not ready for release yet (no documentation was supplied), and also that it was extremely slow in compiling, when compared with Delphi™. C++ Builder also needed roughly 10 MB of disk space for scratch files when writing a "Hello World" program.

While the bulk of the code for the communications program could be ported to Delphi™ 2.0, the same old problem arose once more: The 16-bit version of the RS-232

[14] MSComm V1.10 by Jeff Atwood. The MSComm VCL component duplicates the functionality of the MSCOMM Visual Basic VBX component. All attempts to contact Jeff for acquiring of personal details in order to pass credit for his work failed.

[15] Work started on the 7th of December 1996, with the first successful run on the 29th of December 1996.

communications component (*MSComm*) no longer functioned with Delphi™ 2.0. One of the most important reasons for this was the fact (discovered after many hours of toil) that Microsoft had, in their infinite wisdom, renamed and removed some of the API functions found in the Windows® kernel. The search for a new RS-232 communications component alas started anew.

This time the Internet did not deliver the goods. Countless searches resulted in failure or unsatisfactory solutions. The answer came in the form of AsyncPro™, a professional communications library for DOS®, Windows®, Windows® 95 and Windows® NT. A trial edition of the software was found on the CD bundled together with the book "Delphi™ 2 Developer's Guide" by Pacheco & Teixeira. The license for the library cost R 1,000.00.

The new communications component naturally did not use the same methods and properties as the 16-bit component. This meant a re-write of the communications portion of the program.[16]

2.12. Following up on decoding and calculation

In the meanwhile, work on integrated decoding and calculation proceeded, using Borland® C++, since the researcher could work faster developing initial code in C. Some technical problems were soon experienced, as the limitations of DOS® (most notably the 640 KB barrier and the segmented memory architecture) were reached. Large data sets such as those found in the ECG files could simply not be handled with ease in DOS®. As an interim solution, a statistical/mathematical package[17] was used to temporary overcome some of these problems.

The answer to these architectural problems was, of course, to use Delphi™ 2.0 in the 32-bit environment, where each program theoretically has a 2GB memory space that it can use. No more memory model nightmares!

[16] Work on this re-write started on the 25th of January 1997, and was successfully completed on the 1st of February 1997.

[17] C/Math Toolchest by Mix Software Inc., 1132 Commerce Drive, Richardson, Texas, 75081, USA. ©1991.

2.13. TeeChart and Delphi™ 3.0

As the research progressed, and more milestones were reached, the choice of a graphing tool became inevitable. Drawing the graphs with MS Excel 5.0 no longer sufficed. An integrated solution was needed.

An investigation into writing the routines needed from scratch, soon showed that this idea was not viable. There is no point in re-inventing the wheel! A good "wheel" was found in the form of TeeChart, a professional graphics library for Delphi™.[18]

The issue of money arose once again. The purchase price of TeeChart would mean an additional expenditure of R 1,000.00. Roughly one month later, Borland released version 3 of Delphi™, which promised even more improvements on version 2. Interestingly enough, Borland chose to bundle the TeeChart software with Delphi™ version 3. The choice was clear: purchase the upgrade to Delphi™ at a cost of R1,250.00, with TeeChart bundled in the package.

TeeChart made it possible to reach yet another milestone, namely graphic re-creation of digitally stored ECG data.[19] TeeChart also permitted interesting options such as zoom functions and the creation of callipers to be implemented with relative ease.

2.14. Choice of optimal storage method

Statistical analysis of the digitally acquired ECG data files yielded good information on the composition of these files. Different options were considered, and a storage format was proposed in Chapter 6.[20] The researcher gained experience with the use of statistical functions in MS-Excel 5.0 during this part of the research work.

[18] TeeChart V3.0, runtime version, ©1995 - 1997, David Berneda, TeeMach SL, Barcelona, Catalonia, SPAIN.

[19] Research on the graphic display tool started on the 9th of June 1997, and continued through to the 13th of October 1997.

[20] Research on this topic started on the 13th of May 1997, and continued on and off until roughly the 27th of September 1997.

2.15. Interfacing with the Windows® Registry

With the *Hearts 32* application on the one hand, and the data acquisition DLLs on the other hand, some "glue" was needed to ensure a connection. The connection was realised by utilising the Windows® Registry. Chapter 4 discusses the design issues behind this decision. Appendix B contains more detailed programmatic information on this topic. Creating an interface with the Windows® Registry represented another step towards the final solution.[21]

2.16. Developing a Dynamic Link Library

Chapter 4 discusses the rationale for implementing the data acquisition module as a dynamic link library. Initial work on testing the descriptions for DLL writing, as found in the literature, succeeded instantly. However, when trying to put this knowledge into practice with a real life application, something was missing. Appendix B throws more light on this matter. Suffice it to say that the problems with DLL implementation have been sorted out and the conceptual model has been put into practice - yet another milestone reached.[22]

2.17. The literature research on the electrical activity of the heart

The researcher deemed it necessary to perform a literature research on the basic anatomy and physiology of the heart, as well as the electrical activity of the heart in order to place the ECG in context. This part of the research was extremely interesting and satisfying.[23]

2.18. The use of MS Word

During the course of the writing of this thesis, a lot was learnt about the functionality of MS Word.[24] The writing of a document as complex as a research report would be

[21] Research work on the Registry started on the 1st of September 1997 and continued until the 3rd of December 1997.

[22] Research on DLL creation started on the 18th of August 1997 and continued on and off until the 7th of December 1997.

[23] The writing of this chapter, together with the graphics work, lasted nearly 6 weeks.

[24] Many of these skills were learnt by reading the help text and trying until the desired effect was achieved.

much more difficult without features such as styles, templates, captions, auto numbering, cross referencing and spell checking found in MS Word.

3. Areas for future research

Future research areas include the following:

- Computer-aided interpretation of ECG data, comparing newly acquired ECGs with previous data. This can typically be performed using an expert system.
- Lead to lead, and ECG to ECG QT_c comparison, as an indication of subtle pathological changes in the patient's ECG.
- Annotation of the ECG stored in the *Hearts 32* database, so that the cardiologist can mark and store areas of interest on the ECG directly in the *Hearts 32* database.
- A web-enabled version of the *Hearts 32* database. This should make the data more accessible to distant users.

4. Future of the system

4.1. Hardware

The HP4745A PageWriter II Cardiograph is rapidly ageing. It was initially introduced during 1987. One cannot help but wonder how long the Hewlett-Packard Company will continue to provide service and support for these machines. The Department of Cardiology, Universitas Hospital, Bloemfontein has access to the professional services of a team of highly skilled bio-engineers, who can service the HP4745A ECG machines. This of course, depends on the availability of spare components, in case of component malfunction.

It is reasonable to foresee that these instruments will be in use for at least another two years, during which time the use of the software system resulting from this research can make a big impact on the work performed by the Department of Cardiology.

It is important to note that the HP4745A does not support the year 2000, the reason for this being that dates are only 6 digits long.

4.2. Software

A vast array of software development tools is currently available on the market. The new generation of software development tools aims at increasing programmer productivity by reducing or eliminating as many of the mundane tasks as possible. These Rapid Application Development tools enable the fast creation of a prototype that can easily be turned into a fully functional system.

The time span between major releases of existing software development tools also seems to shorten continuously.

These factors had a big influence on the choice of software development tool used for development of the *Hearts 32* application. The main considerations for the choice of software development tool (application development environment) included:

- Does the manufacturer of the product have a history of developing quality products?
- Does the manufacturer have a sound financial position? (Will the manufacturer be around in years to come in order to support the product?)
- Does the manufacturer have a sound commitment to the product?
- Is this a brand new product, or something new based on tried and trusted technology?
- How is the product received by the market?

It is believed that the choice of Borland® Delphi™ was solid and responsible. Borland has been in the compiler-business in excess of 10 years, producing top quality professional compilers for Pascal, C, Basic, Prolog and Lisp. Borland was also responsible for database products such as Paradox® and Reflex, and through acquisitions, dBASE®.

Admittedly, Borland has experienced some financial difficulties before the release of Delphi™, but Delphi™ (together with C++ Builder) has become Borland's flagship product, solving these financial problems.

Delphi™ uses the Object Pascal language. Borland has, as mentioned, been writing Pascal compilers for roughly 10 years. Delphi™ was also written in Borland® Pascal. The resulting programs are true executable 32-bit Windows® programs. No pseudo-code or run-time modules for program interpretation are needed.

Within a short time after the release of Delphi™, a large part of advertisements for programmer positions, required skills in Delphi™. The bewildering number of newsgroups and Internet sites on Delphi™ also proves that the industry has embraced Delphi™ completely.

One point which does raise a few questions is the following: Since Object Pascal is deemed a high level language, it should be portable. For the largest part, this is true. However, the experience with the different RS-232 communications modules which did not function correctly with different versions of the Delphi™ compiler (even different 32-bit versions) was frustrating, to say the least. Suffice it to say that the blame cannot completely be laid on Delphi™ or Object Pascal. As already mentioned, Microsoft modified the API functions in the Windows® kernel. TurboPower, the manufacturers of AsyncPro™, also reworked the code of their product in order to make it compatible with Delphi™ 3.0.

Another point which deserves attention is that of the ZLIB 1.0.4 general purpose data compression library which was used for the storage of ECG data in the database (Chapter 6). The source code for this compression library was written in ANSI C, to allow compilation on different platforms. For the PC, the code was compiled using the Borland® C++ 5.0 compiler. An Object Pascal interface to these object files was then developed by the authors of ZLIB.[25]

It is possible that the interface to the ZLIB compression library will not automatically function with future releases of Delphi™. (There always seem to be some or other "minor" change between different releases of the compilers which normally has an influence on topics such as mixed language programming.)

[25] Jean-loup Gaily & Mark Adler. ZLIB ©1995-1996.

ZLIB was chosen for the following reasons:

- It is a general purpose compression library.
- It is available, for free.
- The source code is available and the standards on which it is based are documented well in request for comments (RFCs).
- It is available on different platforms.

In the light of these facts, it is sincerely believed that, should the existing ZLIB library not function easily with future releases of Delphi™, there will be some support to overcome the problem.

The Paradox® RDBMS supports the year 2000. This means that the *Hearts 32* application is in no immediate danger of the year 2000 problem.

The experience gained with RAD tools and visual development environments does raise one important point: since a large portion of the program is hidden in the values of the properties of the objects used, it is no longer possible to read the source code listing alone to understand the working of a particular piece of code. The programmer must always refer to the visual components as well. This can be impractical at times, and must be kept in mind when preparing printed copies of source code.

5. Extension of the system

The data acquisition module developed for the HP4745A PageWriter II Cardiograph can be used as long as the HP4745A is in use (provided that the *Hearts 32* application is used).

The *Hearts 32* application was developed in such a way that new data acquisition modules for other ECG machines can be created as dynamic link libraries. All that is needed is that these data acquisition modules deliver their data sets in the format outlined in Appendix B.

This safeguards *Hearts 32* against changes in ECG equipment, and also protects the investment of the Department of Cardiology, Universitas Hospital, Bloemfontein, in the *Hearts 32* application.

The objectives as defined in Chapter 1 can be summarised as follows:

Objective	Result
Data acquisition and decoding module for HP4745A PageWriter II Cardiograph.	Success.
A method for database storage and retrieval of digitally captured ECG Data.	Success.
Develop specifications to allow other developers to create data acquisition modules for *Hearts 32*.	Success.
Develop a graphic browser for recreating of stored ECGs.	
Simultaneous views of different ECGs	Success.
Superimposing of selected leads	Success.
Printing of ECGs	Success.

From the summary above it can be seen that the aims of the study has successfully been reached.

6. Summary

Although many problems were encountered during the course of this study, and many questions had to be answered through reading and own interpretation, the hypothesis as outlined in Chapter 1 could be proved.

The hypothesis was proved in a test environment. This means that a limited version of the new *Hearts 32* database was developed to act as the host application and also to illustrate the interface between the data acquisition and graphics modules. Conversion of the contents of the current *Hearts* database has been programmed, and this work will ease the migration of the *Hearts* database to the *Hearts 32* database considerably.

Appendix A

HP Diagnostic Cardiology Digital Transmission Protocol

Rev. 3.0, September 25, 1985.

WARNING: The information contained in this appendix contains confidential information of a proprietary nature which has been made available to the researcher by Hewlett Packard under a confidential disclosure agreement. No part of this appendix may be made public or used without proper prior written approval from the Hewlett Packard Company.

Pages 166 - 217 left blank intentionally.

Appendix B

Hearts 32 ECG Storage Format

1. Introduction

While it is recognised that different electrocardiographs store data in their individual proprietary format, one of the goals of this research was to find a common storage format that would enable the storage of ECG data in the *Hearts 32* database.[1]

Should it be required that any electrocardiograph other than a HP4745A PageWriter II Cardiograph be connected to the *Hearts 32* database application, a Data Acquisition Module must be developed for said electrocardiograph. The Data Acquisition Module will be in the form of a Dynamic Link Library (DLL) (as discussed in Chapter 4, section 3.1). In order to be compatible with the *Hearts 32* database application, the data files produced by such a DLL must conform to the format described in this appendix.

The storage format described in this appendix is intended to allow other developers to create ECG data sets that will be compatible with the *Hearts 32* database.[2]

2. Lead Identifiers

The 14 leads present in a *Hearts 32* ECG data file are summarised in Table B-1 below:

Table B-1: Leads present in a binary ECG file

Number	ID	Lead description
1	1	I
2	2	II
3	3	III
4	4	aVR

[1] A device (hardware) independent storage format for digitally acquired ECG data.

[2] The storage format resulted from the research described in Chapters 4, 6 and Appendix A.

Table B-1: Leads present in a binary ECG file (continued)

Number	ID	Lead description
5	5	aVL
6	6	aVF
7	7	V1
8	8	V2
9	9	V3
10	10	V4
11	11	V5
12	12	V6
13	99	(Calibration Pulse)
14	24	Rhythm Strip ($V1_r$)

3. File Structure

In order to allow random access to lead data, it was decided that the ECG data file will consist of two parts, as described in Table B-2 below:

Table B-2: Structure of a Hearts 32 ECG data file

Item	Description
Header	Contains lead data characteristics. Items include number of leads in the file, Lead ID, number of samples per lead and also the offset of the first value of the lead in the data file.
Lead data	Contains the actual lead data.

Table B-3 describes each part as identified in Table B-2 in more detail. Item 1 (Lead count) will occur only once in the decoded file, at the very beginning. Item 2 will occur once in the file. Note that item 2 consists of a set of items 2.1 to 2.6 inclusive, where the number of repetitions is equal to the number of leads as indicated by item 1 (Lead count). A total of 14 leads is present, as showed in Table B-1. Item 2 is followed by item 3, which occurs once in the file. As with item 2, item 3 consists of a set of items 3.1 to 3.2 inclusive, where the number of repetitions is equal to the number of leads as indicated by item 1 (Lead count).

Table B-3: Data elements in a Hearts 32 ECG data file

Number	Item	Data Type	Length (bytes)
1	Lead count	Integer	3
2	Lead map information		
2.1		Space	1
2.2	Lead ID	Integer	3
2.3		Space	1
2.4	Samples in lead	Integer	5
2.5		Space	1
2.6	Offset in file	Long integer	7
3	Lead Data		
3.1		Space	1
3.2	Lead Data	Integer value for the HP4745A. (Could also be real value for other ECG machines. Does not matter.[3])	N/A

In order to identify the different values in the file, they need to be separated from each other, normally with a space (ASCII 32, indicated with the letter "b" in this appendix). This is also the case with the decoded data sets, as indicated in Table B-3. Proper placement of these spaces (please refer to items 2.1 and 3.1 in Table B-3) ensures that no unnecessary spaces are written at the end of the file. Should the first space have been written directly following the lead count (item 1), as well as items 2.6 and 3.2, an unnecessary space would be present at the end of the file.

The width of items 1 to 2.6 inclusive has been fixed, to ease manipulation of the header information. Since the header is relatively small (representing only 0.738% of the average total file size[4]), the additional spaces do not incur too much of a storage overhead.

3 Stored data are read as ASCII strings from the data file. Tests showed that using the *StrToInt* and *StrToFloat* functions in Object Pascal yielded the same results for integer values.

4 Mean Header Size / Mean Total File Size * 100 = (255 / 34552.65) * 100 = 0.738%. Please refer to Table 6-10: Descriptive statistics (ASCII Storage Format, Selective) in Chapter 6.

An example of such data can be seen in Table B-4 below. The example shows the lead count (item 1), header data (lead map items 2.1 - 2.6) for leads I, II and V1$_r$. Table B-5 shows the lead data (lead data items 3.1 & 3.2) with the first few data points for lead I.

Table B-4: Example of header (lead map) data in a Hearts 32 ECG data file

Lead ID	Item Number	Item Description	Data Type	Actual Data	Length (bytes)
Header	data				
	1	Lead count	Integer	b14	3
Lead-map	data				
I	2.1	Padding	Character	b	1
	2.2	Lead ID	Integer	bb1	3
	2.3	Padding	Character	b	1
	2.4	Samples in lead	Integer	bb633	5
	2.5	Padding	Character	b	1
	2.6	Offset in file	Long integer	bbbb255	7
II	2.1	Padding	Character	b	1
	2.2	Lead ID	Integer	bb2	3
	2.3	Padding	Character	b	1
	2.4	Samples in lead	Integer	bb633	5
	2.5	Padding	Character	b	1
	2.6	Offset in file	Long integer	bbb2313	7
V1$_r$	2.1	Padding	Character	b	1
	2.2	Lead ID	Integer	b24	3
	2.3	Padding	Character	b	1
	2.4	Samples in lead	Integer	b2488	5
	2.5	Padding	Character	b	1
	2.6	Offset in file	Long integer	bb27712	7

Table B-5: Example of lead data in a Hearts 32 ECG data file

Lead ID	Item Number	Item Description	Data Type	Actual Data	Length (bytes)
Lead	data				
I	3.1	Padding	Character	b	1
	3.2	Lead data	Integer/float[5]	-15	N/A
	3.1	Padding	Character	b	1
	3.2	Lead data	Integer/float	-4	N/A
	3.1	Padding	Character	b	1
	3.2	Lead data	Integer/float	0	N/A
	3.1	Padding	Character	b	1
	3.2	Lead data	Integer/float	3	N/A
	3.1	Padding	Character	b	1
	3.2	Lead data	Integer/float	4	N/A

Since the different data values are not of the same length, some character is needed to separate the values from one another. A space is written between adjacent values. Note the technique of writing a space before writing the actual value. This circumvents the problem of an unnecessary trailing space being written at the end of the file. (This, of course, only works due to the structure of the lead map itself. If there was no lead map, an unnecessary leading space would be present in the file.)

Items 2.1 to 2.6 are repeated for every lead (up to the number of leads specified in Item 1). Items 3.1 and 3.2 are repeated for every value in each lead, up to the number or samples per lead (Item 2.4).

4. DLL Specifics and Rules

It is suggested that a mnemonic name for the data acquisition be given, such as HP4745A.DLL in the case of the HP4745A PageWriter II Cardiograph, for example.

The exported function contained in the DLL file must be declared as follows:[6]

[5] Data acquired from the HP4745A are in integer format. Using the proposed storage format, it does not matter whether data are represented as integers or as real numbers. Because of this, no pre-allocated length is associated with each data item.

Code Snippet B-1: Object Pascal function declaration

```
function ReadDecodeCalcStoreECG( hAppHandle: THandle; sDescription: PChar; nComPort:
Integer; nBaudRate : Integer; sBinaryFile : PChar; sASCIIFile : PChar; var sECGTime:
TAr20; var sECGDate: TAr20 ) : Integer;  StdCall;
```

The ReadDecodeCalcStoreECG function accepts eight parameters, and returns an integer result. Successful downloading of the ECG data set from the cardiograph will result in a return value of 0. If any problems were encountered, the return value of the function is set to -1. A brief description of the parameters and their data types follows:

Table B-6: Function call parameters

No	Name	Object Pascal Data Type	Generic Data type	Comments
1	hAppHandle	THandle	32 bit signed integer	Parent Application Handle[7].
2	sDescription	PChar	ASCIIZ string	Description of DLL.
3	nComPort	Integer	32 bit signed integer	RS-232 COM Port to which ECG machine is connected.
4	nBaudRate	Integer	32 bit signed integer	Baud rate at which ECG machine communicates.
5	sBinaryFile	PChar	ASCIIZ string	Name of disk file to receive a copy of the binary ECG data.
6	sASCIIFile	PChar	ASCIIZ string	Name of temporary disk file to receive decoded ASCII ECG data.
7	sECGTime	var TAr20	Zero-based array of 20 characters	Time ECG was taken.
8	sECGDate	var TAr20	Zero-based array of 20 characters	Date ECG was taken.

The main application handle is used to associate the data acquisition with the main application (*Hearts 32*). This ensures that the windows belonging to the DLL are closed together with the main application.

[6] In other languages such as C, the corresponding data type must be used instead. The name of the function must, however, be kept intact. Care must be taken with C++ compilers to avoid mangling of the function name.

[7] In the HP4745A.DLL file, the application's instance was assigned the value of the parent application handle as follows: Application.Handle := hAppHandle.

The values of the binary and ASCII file names are determined and supplied by *Hearts 32*. The contents is made up of the path name of the Windows® 95 directory, followed by TEMP. Appended to this is the eight digit computer number that uniquely identifies each patient in *Hearts 32*. The file extension is .E*nn* for binary files, and .A*nn* for ASCII files, where *nn* is the number of the ECG in the *Hearts 32* database.[8] An example of such a file name is as follows: C:\WIN95\TEMP\123456789.E01.

Items 7 and 8 in Table B-6 need additional attention. Note that these variable parameters have assignments made by the data acquisition module. The format of the time parameter is hh:mm:ss. The format of the data parameter is yy-mm-dd, but yyyy-mm-dd is also acceptable.

Code Snippet B-2: C/C++ function declaration

```
int ReadDecodeCalcStoreECG( HINSTANCE hAppHandle, char * sDescription, int nComPort,
int nBaudRate, char * sBinaryFile, char * sASCIIFile, char * sECGTime, char * sECGDate
);
```

Depending on the coding style used, the function ReadDecodeCalcStoreECG could be defined as follows:

Code Snippet B-3: Alternative C/C++ function declaration

```
int ReadDecodeCalcStoreECG( HINSTANCE, char*, int, int, char*, char*, char*, char* );
```

[8] The range of this number is 1 to 99.

Appendix C

Commonly used abbreviations and terms

Table C-1: Commonly used abbreviations and terms

Abbreviation/Term	Description
API	Application Program Interface.
BDE	Borland Database Engine.
Cart	A trolley with an electrocardiograph machine.
CV%	Coefficient of variation.
DLL	Dynamic Link Library.
ECG	Electrocardiogram.
ECG machine	Electrocardiograph.
ICU	Intensive Care Unit.
LSB	Least Significant Byte.
lsb	least significant bit.
MSB	Most Signifcant Byte.
msb	most significant bit.
mV	Millivolt.
OLE	Object Linking and Embedding.
RS-232	Recommended Standard Number 232, Revision C from the Engineering Department of the Electronic Industries association.
RTL	Run Time Library.
VCL	Visual Component Library
QTc	Corrected QT time.

Bibliography

Books

Abel, P. 1991. *IBM PC Assembly Language and Programming.* Second Edition. Englewood Cliffs, N.J. Prentice-Hall International.

Barkakati, N. 1989. *The Waite Group's Turbo C© Bible.* First Edition. Indianapolis, Indiana. Howard W. Sams & Company, A Division of Macmillan Inc.

Brookes, C.H.P., Grouse, P.J., Jeffery, D.R., Lawrence, M.J. 1982. *Information Systems Design.* Sydney, Australia. Prentice-Hall of Australia Pty Ltd.

Campbell, J. 1987. *C Programmer's Guide to Serial Communications.* First Edition. Carmel, Indiana. SAMS, A Division of Macmillan Computer Publishing.

Campbell, J. 1984. *The RS-232 Solution: How To Use Your Serial Port.* Second Edition. Alameda, California. SYBEX.

Cantù, M. 1997. *Mastering™ Delphi™ 3.* Second Edition. Alameda, California. SYBEX.

Cardenas, A.F. 1985. *Data Base Management Systems.* Second Edition. Newton, Massachusetts. Allyn and Bacon, Inc.

Date, C.J. 1990. An Introduction to *Database Systems.* Fifth Edition. Reading, Massachusetts. Addison-Wesley Publishing Company, Inc.

Dubin, D. 1989. *Rapid Interpretation of EKG's... a programmed course.* Fourth Edition. Tampa, Florida. COVER Publishing Company.

Guyton, A.C. 1966. *Textbook of Medical Physiology.* Third Edition, illustrated. Philadelphia and London. W. B. Saunders Company.

Jensen, C., Anderson, L., Fung, J., Lynnworth, A., Ostroff, M., Rudy, M., Vivrette, R. 1996. *Delphi in depth.* Berkeley, California. Osborne McGraw-Hill.

McFadden, F.R., Hoffer, J.A. 1991. *Database Management.* Third Edition. Redwood City, California. The Benjamin/Cummings Publishing Company, Inc.

Meyer, B.J., Meij, H.S., Labuschagne, C.J.J., Theron, J.J., Grey, S.V., Stewart, R.I., Pitout, M.J., Van Papendorp, D.H., Brown, J.M.M., Smit, Z.M., Seegers, J.C., Meyer, A.C., Haag, M. 1988. *Die Fisiologiese Basis van Geneeskunde.* Vierde hersiene uitgawe. Pretoria. HAUM Uitgewery.

Monk, T.S. 1992. *Windows™ Programmer's Guide to Serial Communications.* First Edition. Carmel, Indiana. Sams Publishing.

Morse, S.P. 1982. *The 8086 8088 Primer.* Second Edition. Hasbrouck Heights, NJ. Hayden Book Company Inc.

Nelson, M. 1991. *The data compression book.* Redwood City, California. M & T Publishing, Inc.

Pacheco, X., Teixeira, S. 1996. *Delphi 2 Developer's Guide.* Second Edition. Indianapolis, Indiana. Sams Publishing.

Swan, T. 1991. *Mastering Turbo Pascal® 6.* Fourth Edition. Carmel, Indiana. Hayden Books.

Tanenbaum, A. S. 1984. *Structured Computer Organization.* Second Edition. Englewood Cliffs, N.J. Prentice-Hall, Inc.

Thurrott, P., Brent, G., Bagdazian, R., Tendon, S. 1997. *Delphi 3 SuperBible.* Corte Madera, California. Waite Group Press™, A Division of Sams Publishing.

Tompkins, W.J., Webster, J.G., Eds. 1981. *Design of Microcomputer-Based Medical Instrumentation.* Englewood Cliffs, NJ : Prentice-Hall.

Publications

Abenstein, J.P., Tompkins, W.J. 1982. *A New Data-Reduction Algorithm for Real-Time Electrocardiogram Analysis.* <u>IEEE Transactions on Biomedical Engineering</u>, vol. BME-29, pp.43-48.

Axenborg, J.E. 1989. *BIOLAB - a computerized on-line system for physiological measurements in experimental animals.* Computer Methods and Programs in Biomedicine, vol. 28, pp. 75-85.

Berson, A.S., Wojick, J.M., Pipberger, H.V. 1977. *Precision Requirements for Electrocardiographic Measurements Computed Automatically.* IEEE Transactions on Biomedical Engineering, vol. BME-24, no. 4, pp. 382-385.

Farrell, A.P., Bruce, F. 1987. *Data Acquisition and Analysis of Pulsatile Signals Using a Personal Computer: An Application in Cardiovascular Physiology.* Computers in Biology and Medicine, vol. 17, no. 3, pp. 151-159.

Herbst, C.P., Diedericks, J., Uys, N.J., Brummer, J., Lötter, M.G. 1991. *Use of a Personal Computer for Fast Acquisition of Cardiovascular Data Over an Extended Period.* Computers in Biology and Medicine, vol. 21, no. 6, pp. 407 - 415.

Jalaleddine, S.M.S., Hutchens, C.G., Strattan, R.D., Coberly, W.A. 1990. *Electrocardiogram Data Compression Techniques - A Unified Approach.* IEEE Transactions on Biomedical Engineering, vol. 37, no. 4, pp. 329-340.

Jossinet, J., Leftheriotis, G., Vernier, F., Saumet, J.L. 1990. *A Computerized Bioelectrical Cardiac Monitor.* Computers in Biology and Medicine, vol. 20, no. 4, pp. 253-260.

Kennedy, H.L., Ratcliff, J.W. 1987. *Ambulatory electrocardiography and computer technology - practical advantages.* Americal Heart Journal, vol. 113, no. 1, pp. 186-193.

Mustard, R.A., Cosolo, A., Fisher, J., Pike, T., Schouten, B.D., Swanson, H.T. 1990. *PC-Based System for Collection and Analysis of Physiological Data.* Computers in Biology and Medicine, vol. 20, no. 2, pp. 65-74.

Perez, A. 1983. *Byte-wise CRC Calculations.* IEEE Micro, June, pp. 40-49.

Piper, I., Guha, A., Tator, C.H., Gentles, W. 1987. *A Microcomputer System for On-Line Collection of Blood Flow and Related Physiological Data.* Computers in Biology and Medicine, vol. 17, no. 4, pp. 279-291.

Tai, S.C. 1991. *SLOPE - a real-time electrocardiogram data compressor.* Medical & Biological Engineering & Computing, vol. 29, pp. 175-179.

Tai, S.C. 1992. *ECG data compression by corner detection.* Medical & Biological Engineering & Computing, vol. 30, pp. 584-590.

Tai, S.C. 1993. *AZTDIS - a two-phase real-time electrocardiogram data compressor.* Journal of Biomedical Engineering, vol. 15, pp. 510-515.

Van Vliet, B.N., West, N.H., Road, J.D. 1987. *Measurement of Signal Period on a Personal Microcomputer and its Application to the Analysis of Cardiac Interval and Blood Pressure.* Computers in Biology and Medicine, vol. 17, no. 3, pp. 143-150.

Proceedings

Brodie, D.A., Mann, B. 1982. *A low-cost data acquisition and display system for physiological measures.* Proceedings of the Physiological Society. Leeds meeting. p. 1.

Protocols

Hewlett-Packard Company. 1985. *HP DIAGNOSTIC CARDIOLOGY DIGITAL COMMUNICATIONS OVERVIEW for the 4750A Option A50 and A60, and 4760A Cardiographs, and the 5600C electrocardiogram Management System.* McMinnville, Oregon. Rev. 3.0.

Hardware Manuals

Hewlett-Packard Company. 1983. *Hewlett Packard HP4700A Cardiograph Formatting Guide.* Andover, Massachusetts. Part Number 04700-91997.

Hewlett-Packard Company. 1988. *HP 4745A PageWriter II Cardiograph Operating Guide.* USA. Part Number 04745-91908.

Hewlett-Packard Company. 1989. *Model 4745A/4755A Cardiograph Service Manual.* USA. Part Number 04755-91909.

Software Manuals

Borland®. 1997. *Object Pascal Language Guide*. Borland International, Inc. Scotts Valley, California.

Product Brochures

Cardio Control BV. [s.a.]. Cardio Perfect. PC Based electrocardiogram. Rijswijk, The Netherlands.

Hewlett-Packard Company. 1995. HP M1730B TraceMaster electrocardiogram System, United States of America.

Marquette electronics. 1992. MAC VU. Milwaukee, Wisconsin.

Request for Comments

Deutsch, L. P., Gaily, J-L. 1996a. *ZLIB Compressed Data Format Specification version 3.3.* Network Working Group, Request for Comments: 1950. ftp://ds.internic.net/rfc/rfc1950.txt.

Deutsch, L. P. 1996b. *DEFLATE Compressed Data Format Specification version 1.3.* Network Working Group, Request for Comments: 1951. ftp://ds.internic.net/rfc/rfc1951.txt.

Deutsch, L. P. 1996c. *GZIP File Format Specification version 4.3.* Network Working Group, Request for Comments: 1952. ftp://ds.internic.net/rfc/rfc1952.txt.