

**ORIGINAL**

~~FILED~~ IN CLERK'S OFFICE

~~U.S.P.C. 1-1-1971~~

~~JUN 28 2004~~

UNITED STATES DISTRICT COURT  
NORTHERN DISTRICT OF GEORGIA  
ATLANTA DIVISION

LUTHER D. THOMAS, Clerk  
By: *[Signature]* Deputy Clerk

PICIS, INC. and  
PICIS, S.A.,

Plaintiffs,

V.

**SURGICAL INFORMATION  
SYSTEMS, LLC and  
CAPSULE TECHNOLOGIE,**

**Defendants.**

**104 CV 1870**  
Civil Action No.:

## JURY TRIAL DEMANDED

# CAP

## COMPLAINT

NOW COMES PICIS, INC. and PICIS, S.A., plaintiffs in the above-captioned matter, and make and file this Complaint and request for injunctive relief and monetary damages against Defendants SURGICAL INFORMATION SYSTEMS, LLC and CAPSULE TECHNOLOGIE, and in support thereof respectfully show as follows:

## NATURE AND BASIS OF ACTION

1. This is an action for patent infringement arising under the patent laws of the United States, including 35 U.S.C. §§ 271, 281, 283, 284 and 285.

**FORMS RECEIVED**

**Consent To US Mag.**

## Pretrial Instructions

## Titel VII NIE

515.

### THE PARTIES

2. Picis, Inc. ("Picis") is a Delaware corporation having a principal place of business at 100 Quannapowitt Parkway, Wakefield, MA, 01880.

3. Picis, S.A. is a societe anonyme organized under the laws of France having a principal place of business at 38 rue de Berri, Paris, France, 75008.

4. Surgical Information Systems, LLC ("SIS") is a Georgia limited liability company having a principal place of business at 3650 Mansell Road, Suite 300, Alpharetta, GA 30022.

5. Capsule Technologie ("Capsule") is a French corporation having a principal place of business at 79, rue du Faubourg Poissonnière, Paris, France 75009.

### JURISDICTION AND VENUE

6. This Court has jurisdiction over the subject matter of this action under 28 U.S.C. § 1338(a) because this action arises under the patent laws of the United States, Title 35, United States Code.

7. This Court has personal jurisdiction over SIS because SIS has its principal place of business in the Atlanta Division of this District, and because SIS is engaged in activities in the Atlanta Division of this District.

8. This Court has personal jurisdiction over Capsule because Capsule is engaged in activities in concert with SIS, in the Atlanta Division of this District.

9. Venue properly lies in this District under 28 U.S.C. §§ 1391(b)-(c) and 1400(b) because SIS and Capsule are subject to personal jurisdiction in this District, and because SIS and Capsule are engaging in infringing activities and other activities related to this action in the Atlanta Division of this District.

### BACKGROUND

10. Plaintiff realleges and incorporates by reference paragraphs 1 through 9 above.

11. Picis provides integrated operating room, anesthesia and critical care software solutions that automate clinical documentation and business practices throughout the operating room, recovery room and intensive care units of hospitals (“Picis’s Medical Software”).

12. One aspect of Picis’s Medical Software is a software-based sub-application (“Picis’s Software Engine”) that captures data from different medical devices, including bedside medical devices (such as physiological monitors, ventilators and others), at a patient’s bedside automatically for further automated patient charting, and automated storage in a database for the purpose of assisting nurses and physicians in their review and analysis of the patient’s condition.

13. Picis's Software Engine operates in conjunction with a library of medical device drivers. A given medical device driver enables Picis's Software Engine to communicate with a corresponding medical device and capture data from that medical device.

14. The commercial implementation of Picis's Medical Software began in the early 1980s.

15. Features of Picis's Software Engine, including features of the medical device drivers, are described and claimed in U.S. Patent No. 5,161,222, entitled "Software Engine Having an Adaptive Driver for Interpreting Variables Produced at a Plurality of Sensors" ("the '222 patent"), which issued on November 3, 1993 and which issued from an application filed on August 20, 1990. A copy of the '222 patent is attached hereto as Exhibit A.

16. Picis, S.A is the owner by assignment of the '222 patent.

17. Picis is the exclusive licensee of the '222 patent and has the express right to enforce the '222 patent.

18. Human Microprocessing, Inc., a predecessor of Picis, S.A., introduced the first commercial, personal computer-based application for automating the anesthesia, recovery and critical care environments.

19. In May 1995, Steven Forrester ("Forrester") became an employee of Picis, S.A. Forrester was tasked to further develop software, including medical device drivers that, among other uses, enable Picis' Software Engine to connect to medical devices.

20. In April 1997, Forrester left the employ of Picis, S.A. to form a new company to provide software installation and integration services in Europe.

21. In 1997, Forrester founded, along with others, a French corporation called Capsule Technologie ("Capsule").

22. Picis, S.A. thereafter contracted with Capsule to continue developing software and medical device drivers that, among other things, were operable with Picis's Software Engine.

23. The contractual relationship between Picis, S.A. and Capsule ceased in June 1999.

24. Picis continues to market and sell Picis's Medical Software to hospitals in the United States and worldwide.

25. Capsule has a world wide web web site at [www.capsuletech.com](http://www.capsuletech.com) ("Capsule's web site"), selected pages of which are attached at Exhibit B.

26. Capsule markets a software application called "DataCaptor<sup>TM</sup>."

27. According to Capsule's web site, DataCaptor™ is a data acquisition and distribution software package enabling collection of data from a plurality of different types of medical devices for, among other things, use in a clinical application.

28. According to Capsule's web site, DataCaptor™ collects all variable data from bedside medical devices through any type of communication hardware or through a direct connection to a bedside computer, DataCaptor™ retrieves and delivers real-time data from more than 250 different medical devices, and DataCaptor™ facilitates software creation by merging data flows and creating a *common protocol for many different data sources*, thus optimizing software development and application management.

29. SIS has a world wide web web site at [www.orsoftware.com](http://www.orsoftware.com) ("SIS's web site"), selected pages of which are attached at Exhibit C.

30. According to SIS's web site, SIS offers hospital surgery software that captures clinical, medical and financial data on every patient event from surgery scheduling through transcription of the surgical report.

31. According to a press release dated February 22, 2004 and posted on SIS's web site, SIS has agreed to integrate Capsule's DataCaptor™ in SIS's

perioperative information system to collect and manage bedside medical device data.

COUNT I  
PATENT INFRINGEMENT

32. Picis realleges and incorporates by reference paragraphs 1 through 31 above.

33. Picis is the exclusive licensee of the '222 patent by virtue of a license granted to Picis by Picis, S.A.

34. On information and belief, SIS has infringed and/or is continuing to infringe at least claim 1 of the '222 patent by using, offering to sell, and/or selling DataCaptor™.

35. On information and belief, Capsule has infringed and/or is continuing to infringe at least claim 1 of the '222 patent by using, offering to sell, and/or selling DataCaptor™.

36. SIS is causing and will continue to cause Picis substantial damages and injury.

37. Capsule is causing and will continue to cause Picis substantial damages and injury.

38. Picis will suffer further damage and injury unless and until SIS is enjoined by this Court from continuing such infringement. The damage caused by SIS is irreparable and cannot be adequately compensated for in money damages.

39. Picis will suffer further damage and injury unless and until Capsule is enjoined by this Court from continuing such infringement. The damage caused by SIS is irreparable and cannot be adequately compensated for in money damages.

COUNT II  
INDUCING PATENT INFRINGEMENT

40. Picis realleges and incorporates by reference paragraphs 1 through 39 above.

41. On information and belief, SIS is actively inducing and will continue to induce others to infringe the '222 patent by distributing equipment, software, instructions, and information enabling, inducing and encouraging infringement of the '222 patent.

42. SIS has caused and will continue to cause Picis substantial damage and injury by virtue of its continuing active inducement of infringement of the '222 patent. *Picis will suffer further damage and injury unless and until SIS is enjoined by this Court from continuing such active inducement of infringement. The damage caused by SIS is irreparable and cannot be adequately compensated for in money damages.*



43. On information and belief, Capsule is actively inducing and will continue to induce other to infringe the '222 patent by distributing equipment, software, instructions, and information enabling, inducing and encouraging infringement of the '222 patent.

44. Capsule has caused and will continue to cause Picis substantial damage and injury by virtue of its continuing active inducement of infringement of the '222 patent. Picis will suffer further damage and injury unless and until Capsule is enjoined by this Court from continuing such active inducement of infringement. The damage caused by Capsule is irreparable and cannot be adequately compensated for in money damages.

#### PRAYER FOR RELIEF

WHEREFORE, Picis respectfully requests that Judgment be entered that SIS and Capsule have infringed valid claims of the '222 patent, and that Picis be granted the following relief:

(i) Entry of a preliminary injunction pending resolution of this action and a permanent injunction thereafter restraining SIS and Capsule, their officers, agents, servants, attorneys and all persons acting in concert with SIS and Capsule from further acts of infringement or inducement of infringement of the '222 patent;

(ii) An award of damages sufficient to compensate Picis for SIS's and Capsule's infringement and inducement of infringement of the '222 patent.

(iii) An award of prejudgment interest pursuant to 35 U.S.C. § 284, from the date of each act of infringement or inducement of infringement of the '222 patent by SIS and Capsule until the day a damages judgment is entered herein, and a further award of post judgment interest, pursuant to 28 U.S.C. § 1961, continuing thereafter until such judgment is paid;

(iv) An award of reasonable attorneys' fees, pursuant to 35 U.S.C. § 285, and Plaintiffs' costs of suit, pursuant to 35 U.S.C. § 284; and

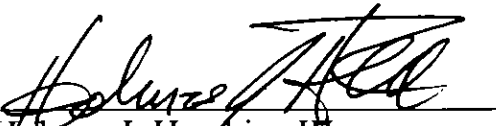
(v) Such other and further relief as this Court shall deem appropriate.

#### JURY DEMAND

Plaintiffs hereby demand a trial by jury.

Respectfully submitted this 28<sup>th</sup> day of June, 2004.

KING & SPALDING LLP

By:   
Holmes J. Hawkins III  
Georgia Bar No. 338681  
Christine Cox  
Georgia Bar No. 192229  
191 Peachtree Street, Suite 4900  
Atlanta, Georgia 30303-1763  
Ph: (404) 572-4600  
Fax: (404) 572-5145

Of counsel:

Michael D. Bednarek  
Daniel P. Westman  
Lawrence D. Eisen  
SHAW PITTMAN LLP  
1650 Tysons Boulevard  
McLean, Virginia 22102-4859  
Ph: (703) 770-7900  
Fax: (703) 770-7901

*Attorneys for Plaintiffs  
Picis, Inc. and Picis, S.A.*

**United States Patent** [19]

Montejo et al.

US005161222A

[11] **Patent Number:** **5,161,222**[45] **Date of Patent:** **Nov. 3, 1992**

[54] **SOFTWARE ENGINE HAVING AN ADAPTABLE DRIVER FOR INTERPRETING VARIABLES PRODUCED BY A PLURALITY OF SENSORS**

[75] **Inventors:** Leopoldo S. Montejo; Martine Pean, both of Paris, France

[73] **Assignee:** Human Microprocessing, Inc., Milwaukee, Wis.

[21] **Appl. No.:** 569,857

[22] **Filed:** Aug. 20, 1990

[51] **Int. Cl.<sup>3</sup>** ..... G06F 13/14

[52] **U.S. Cl.** ..... 395/500; 364/934.0; 364/934.2; 364/934.3; 364/935.42; 364/940.9; 364/940.1; 364/942.4; 364/DIG. 2

[58] **Field of Search** ..... 395/500, 325, 700; 364/146, 147, 188, 189, 138, 139, 200 MS File, 900 MS File

[56] **References Cited****U.S. PATENT DOCUMENTS**

4,481,574	11/1984	De Fino et al.	364/200
4,570,217	2/1986	Allen et al.	
4,589,063	5/1986	Shah et al.	395/275
4,649,479	3/1987	Advani et al.	395/700
4,663,704	5/1987	Jones et al.	364/188

4,672,532	6/1987	Jonge Vos	395/600
4,701,848	10/1987	Clyde	395/325
4,734,854	3/1988	Afshar	364/200
4,858,101	8/1989	Stewart et al.	364/131
5,014,185	5/1991	Saito et al.	364/188

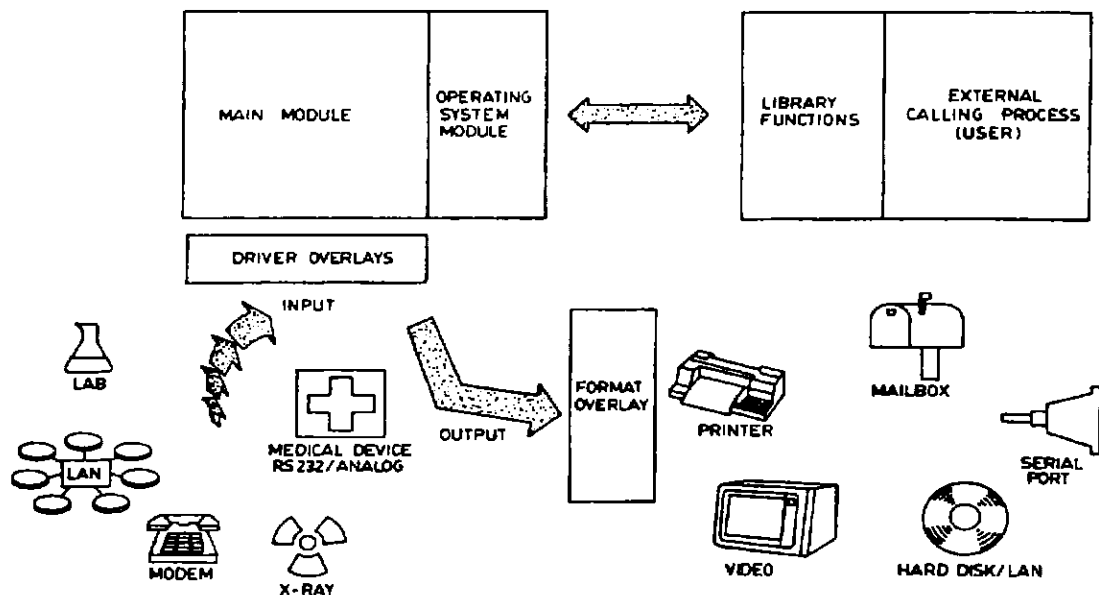
**OTHER PUBLICATIONS**

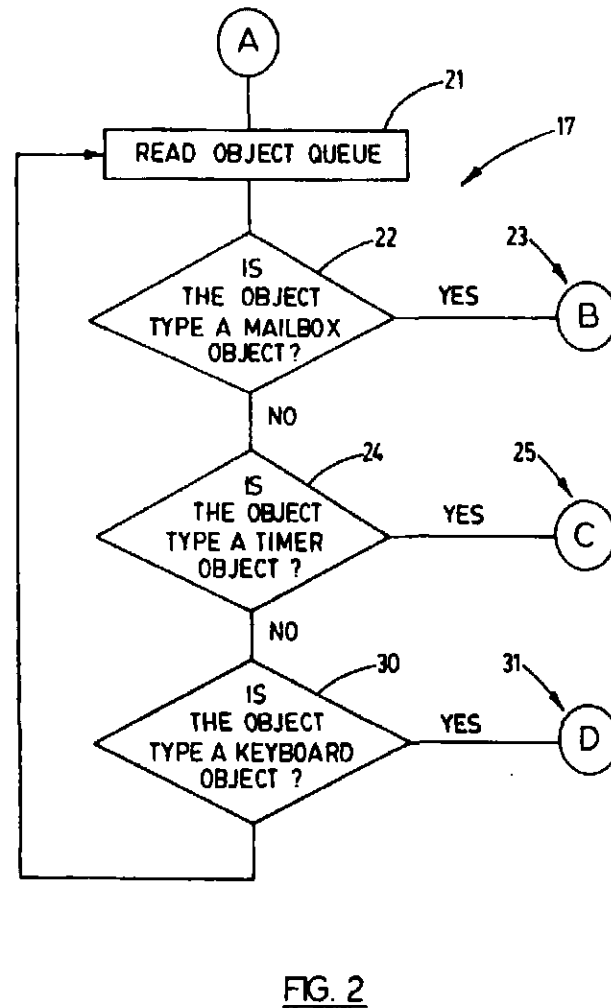
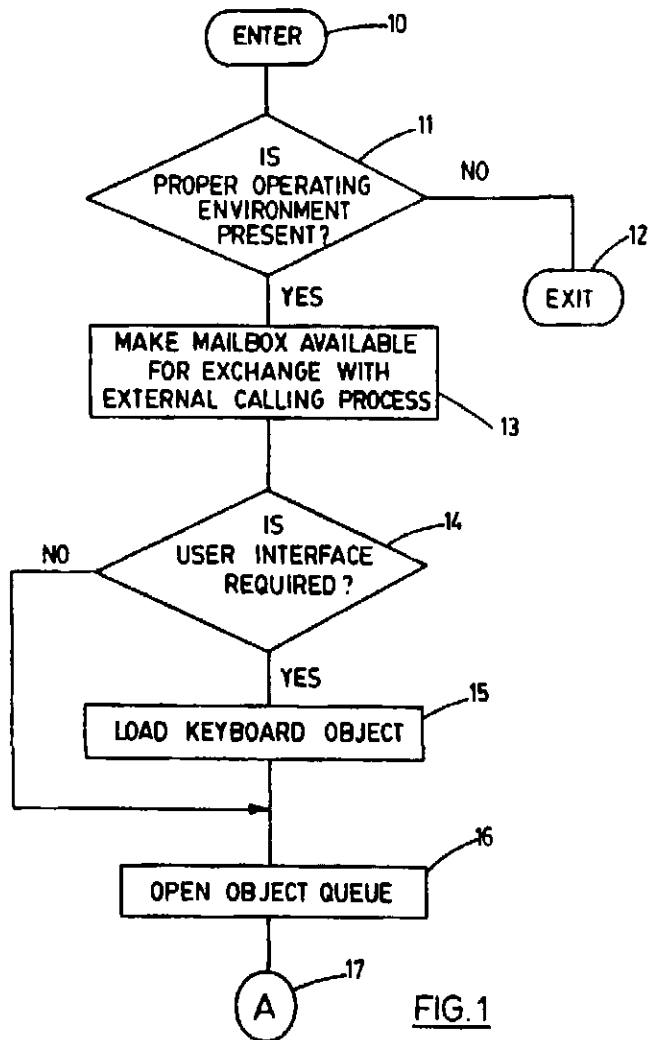
Armbrust et al., "Forward Looking DVI," *PC Tech Journal*, vol. 3, No. 9, Sep. 1985, pp. 42-55.

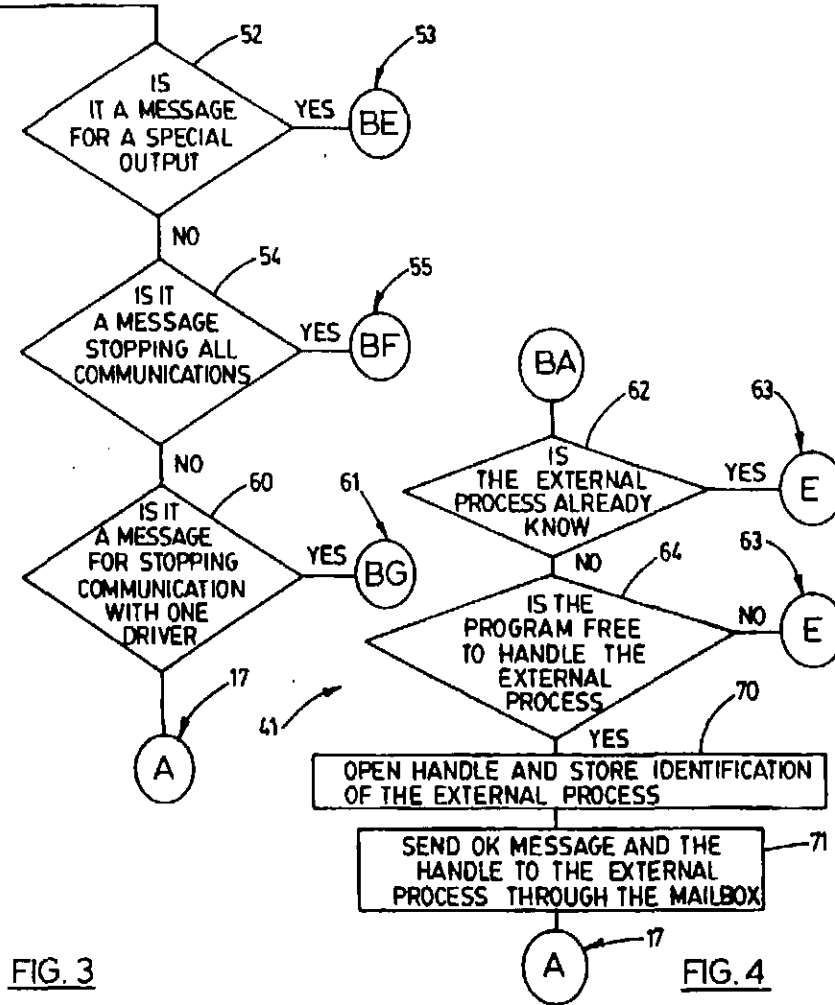
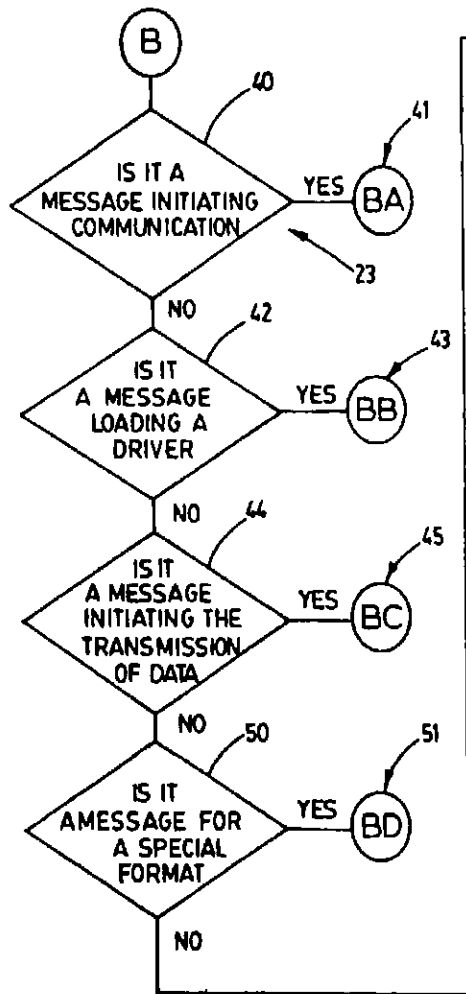
*Primary Examiner*—Thomas C. Lee  
*Assistant Examiner*—William M. Treat  
*Attorney, Agent, or Firm*—Godfrey & Kahn

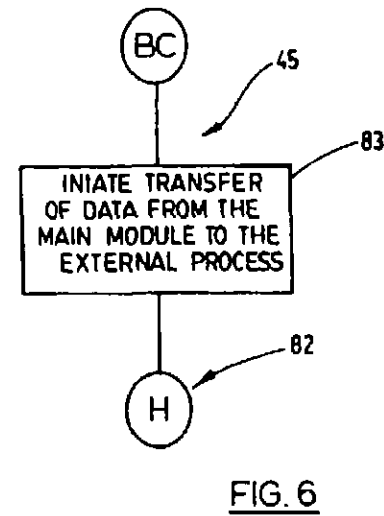
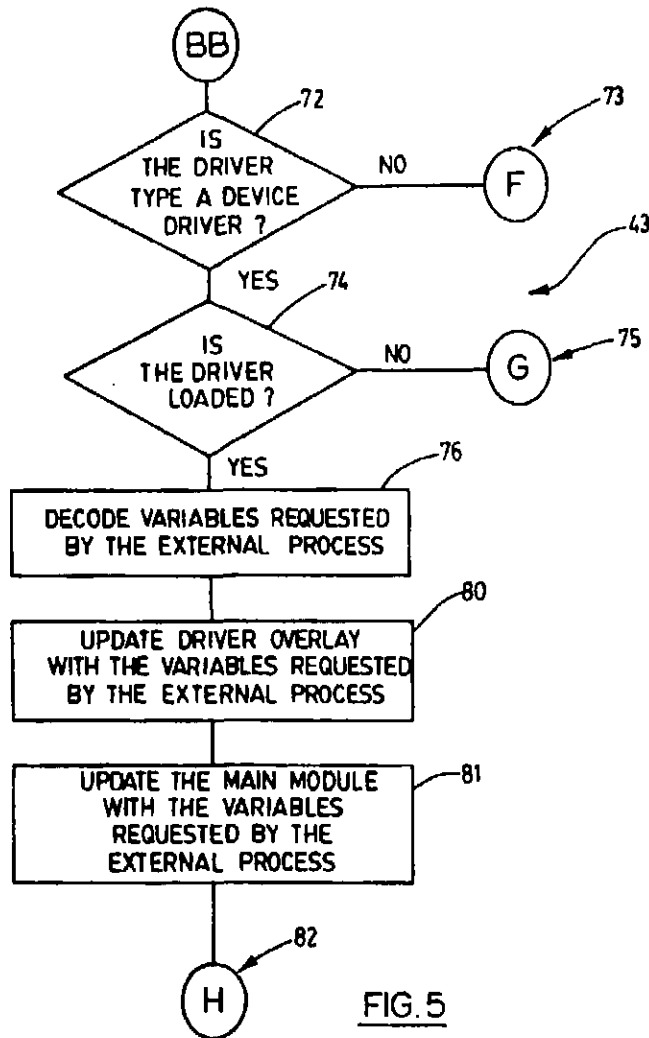
[57] **ABSTRACT**

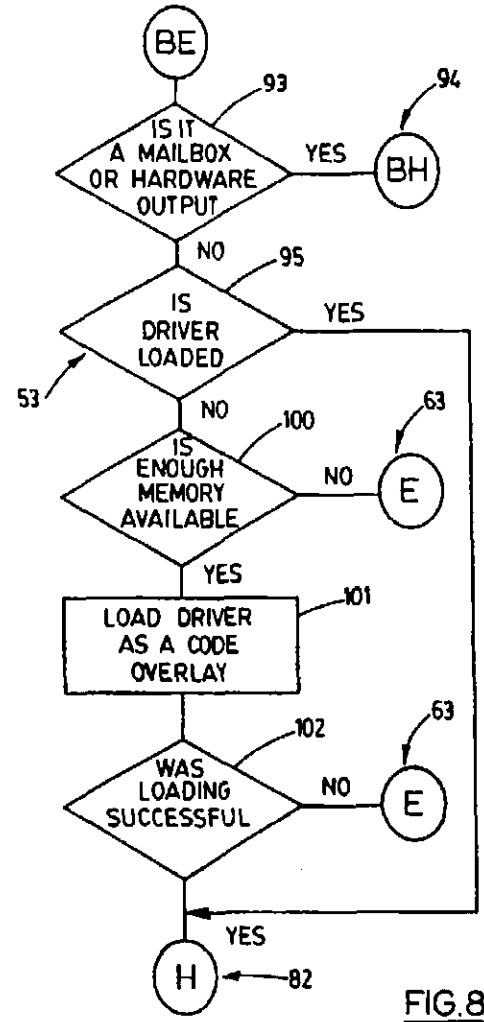
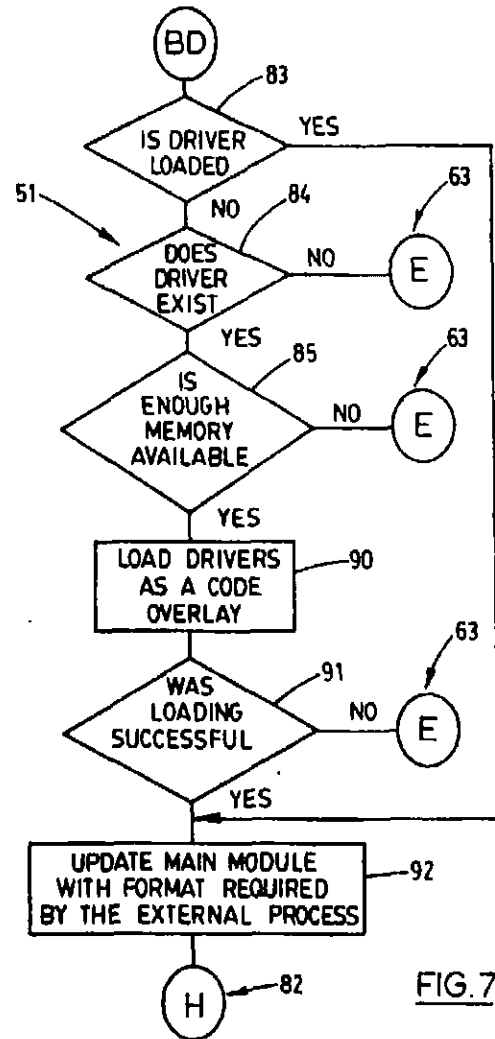
A method for interpreting variables produced by a sensor which communicates by a means of serial analog protocols including interpreting an external process request for data information from the sensor; overlaying a predetermined adaptable driver which when adjusted in a predetermined fashion corresponds to the characteristics of the sensor; polling or listening to the sensor thereby receiving the data information requested; and transmitting the information to a predetermined destination.

**14 Claims, 18 Drawing Sheets**











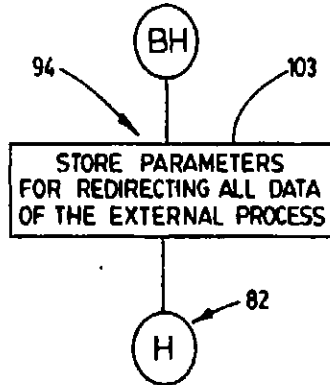


FIG. 9

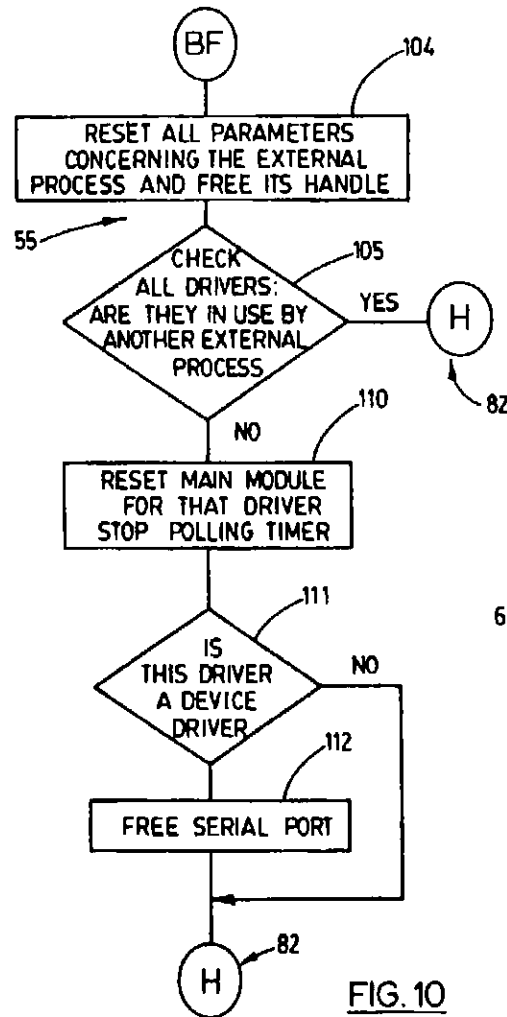


FIG. 10

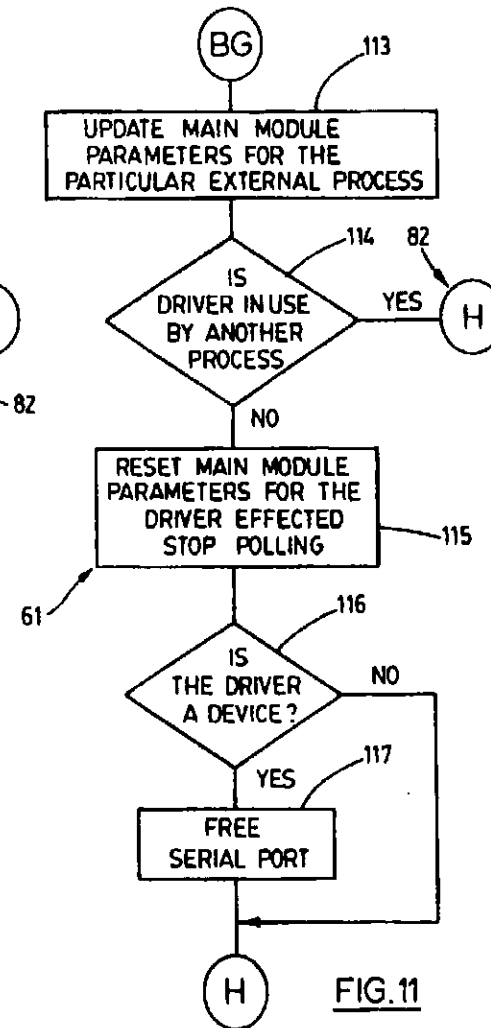


FIG. 11

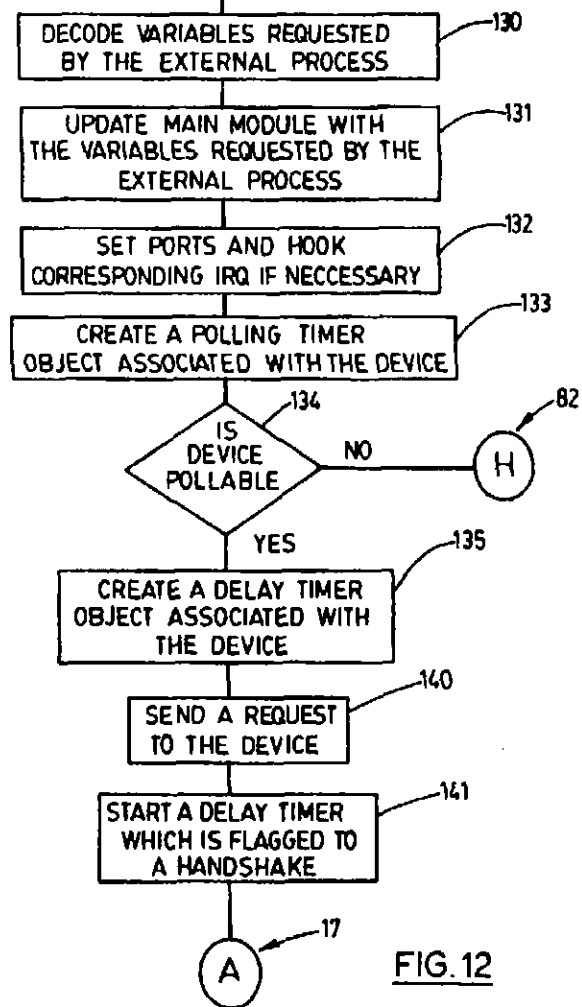
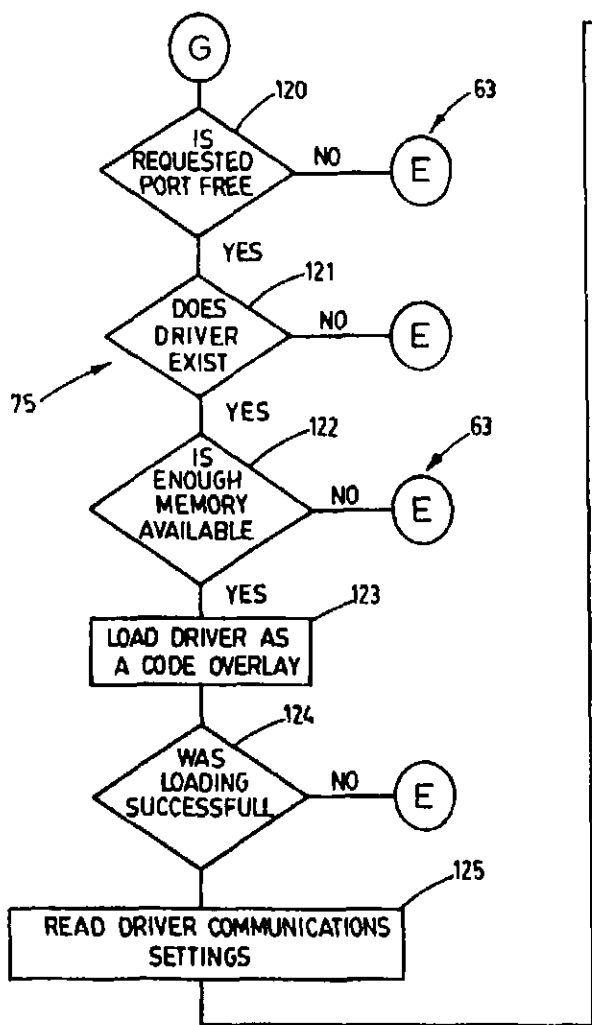


FIG. 12

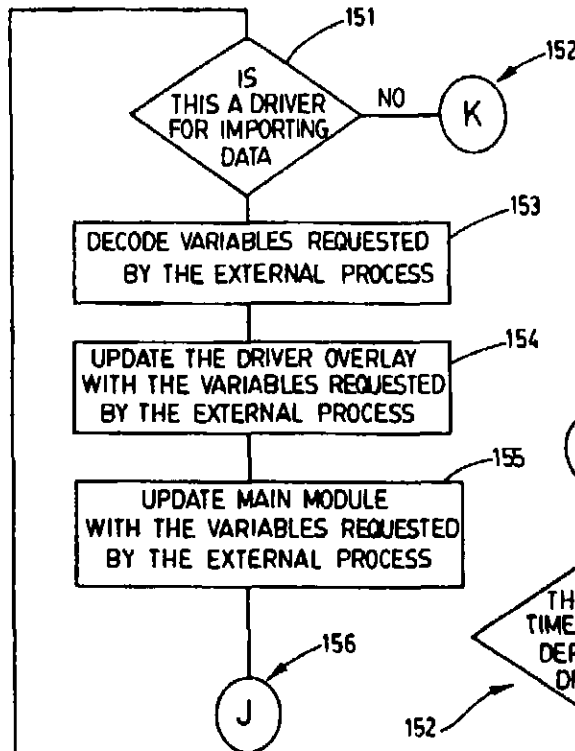
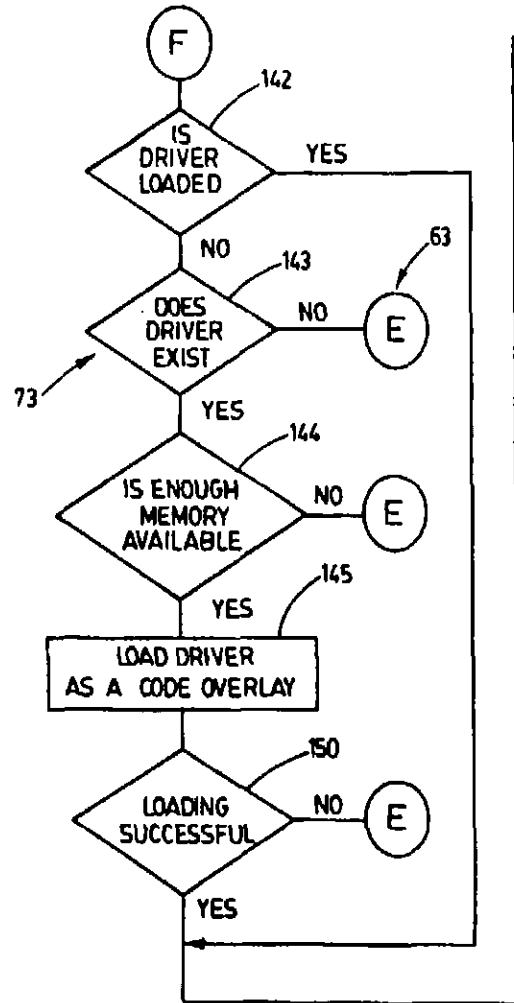


FIG. 13

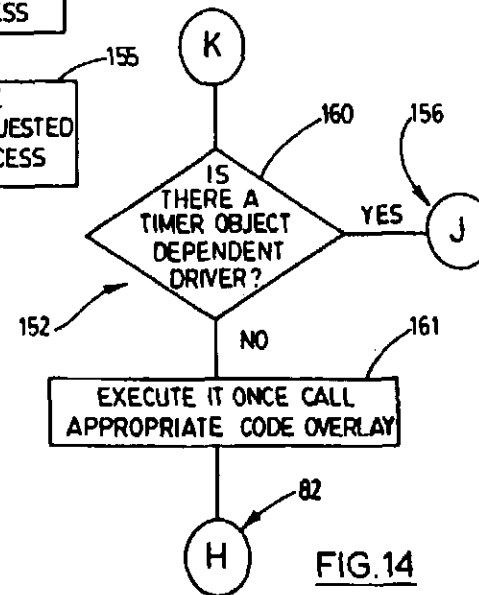
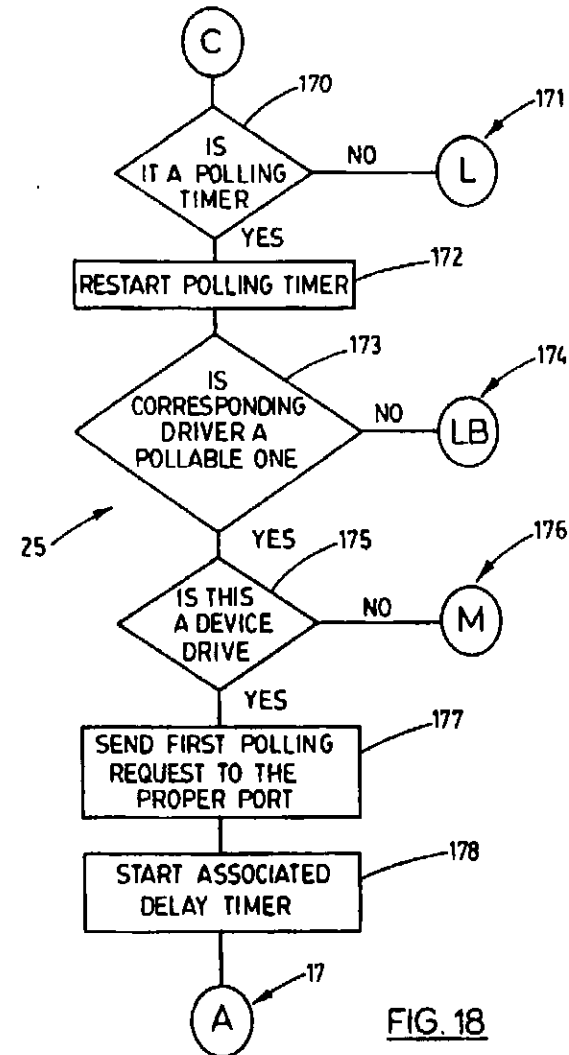
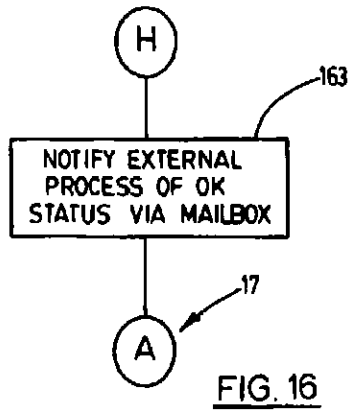
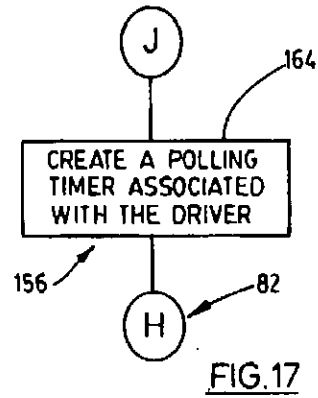
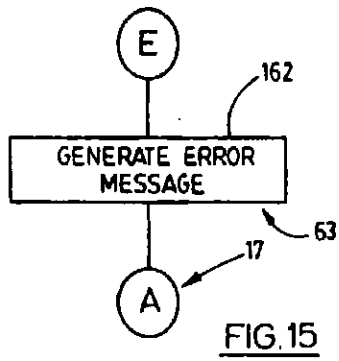


FIG. 14



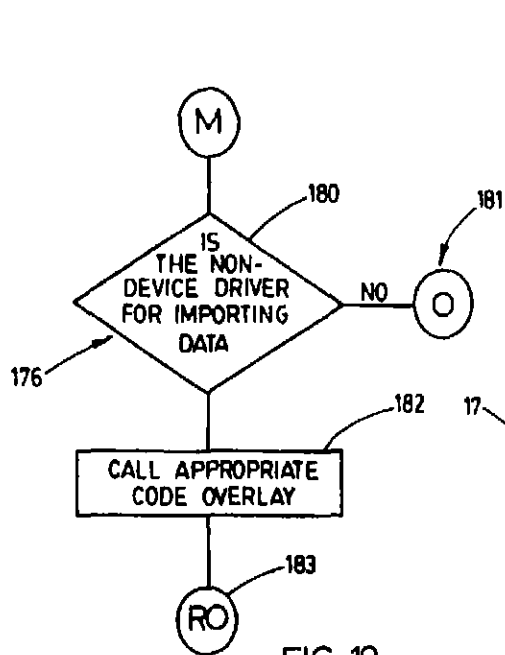


FIG. 19

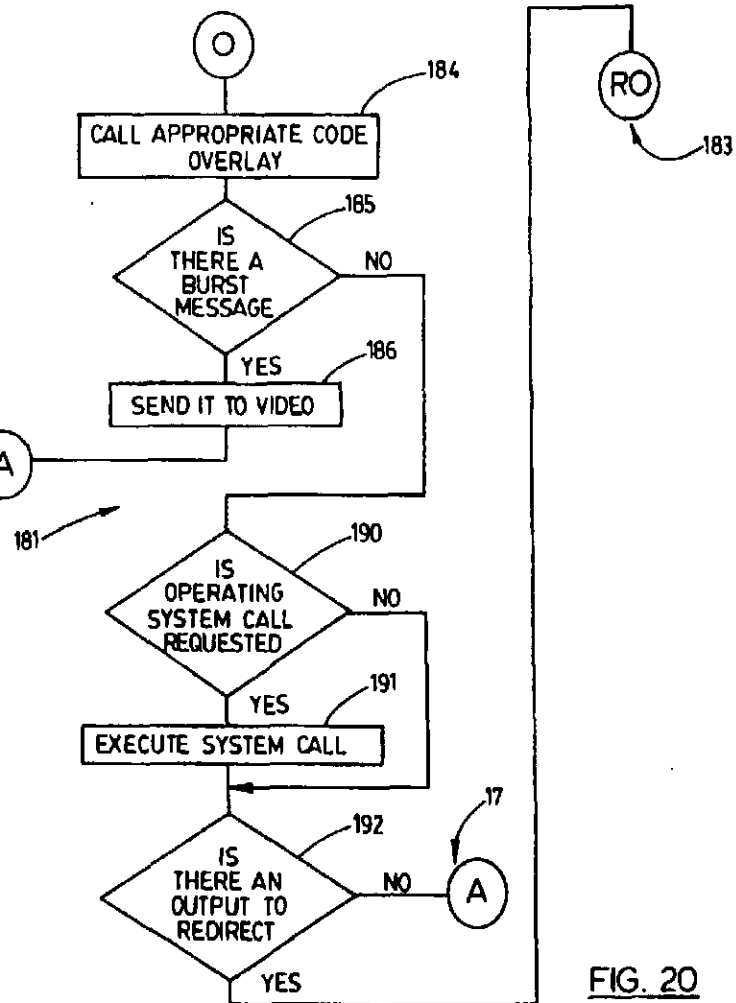


FIG. 20

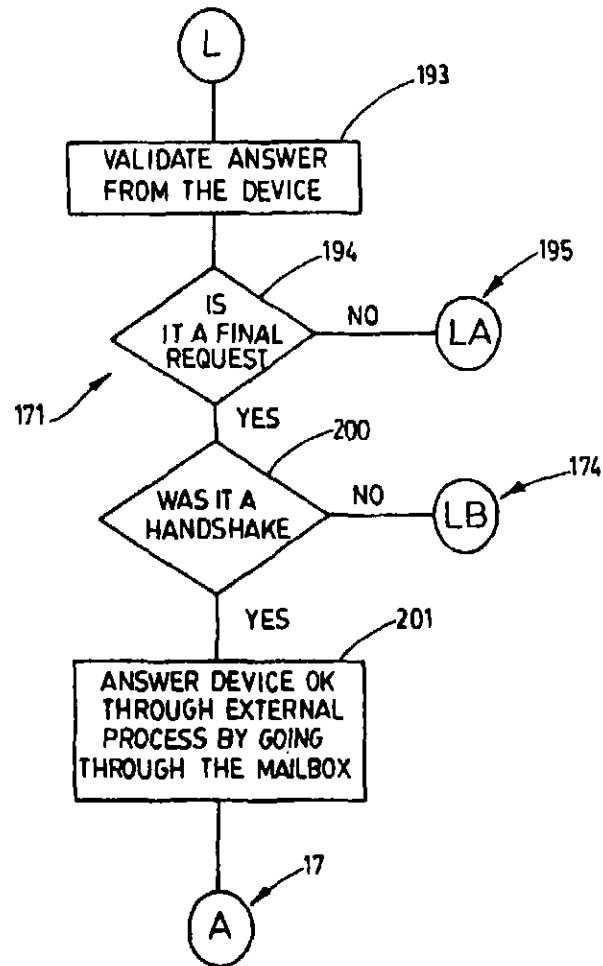


FIG. 21

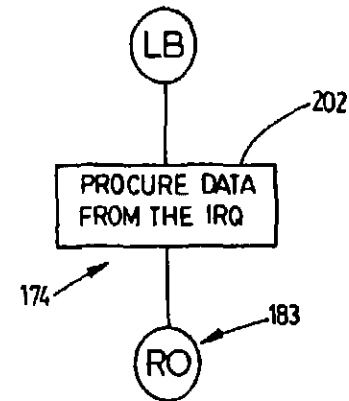
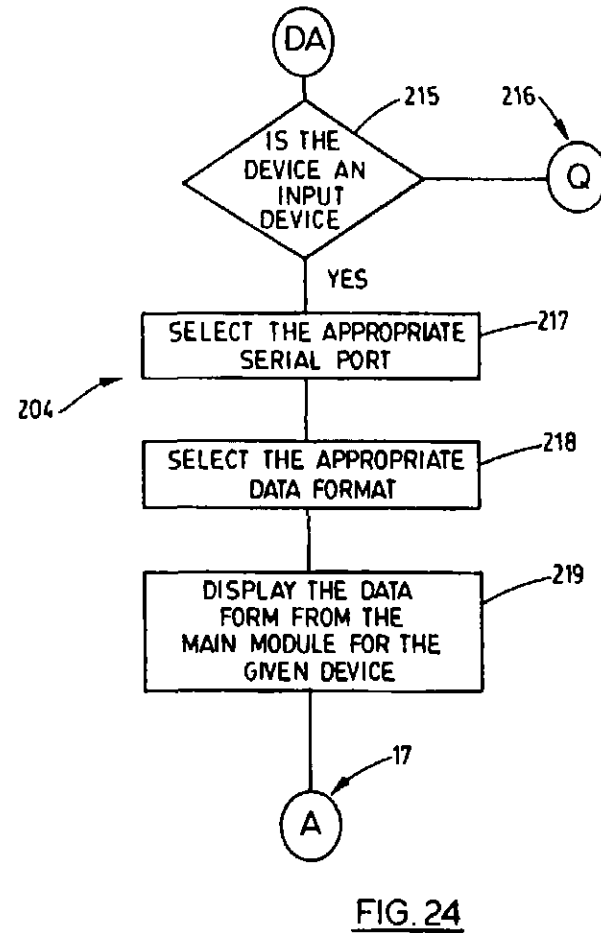
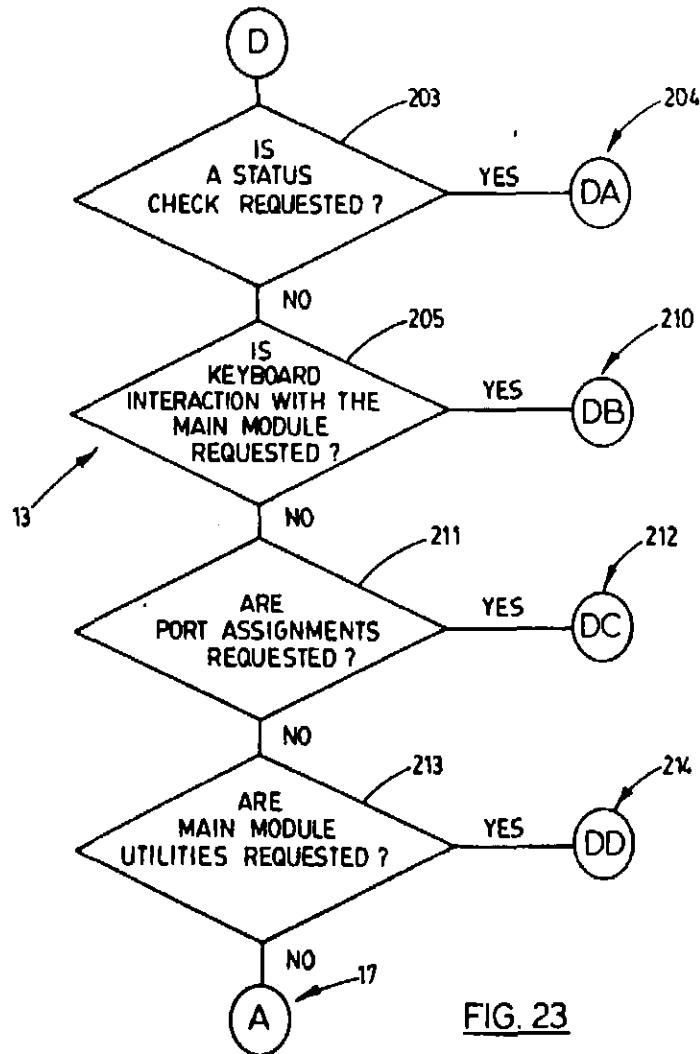


FIG. 22



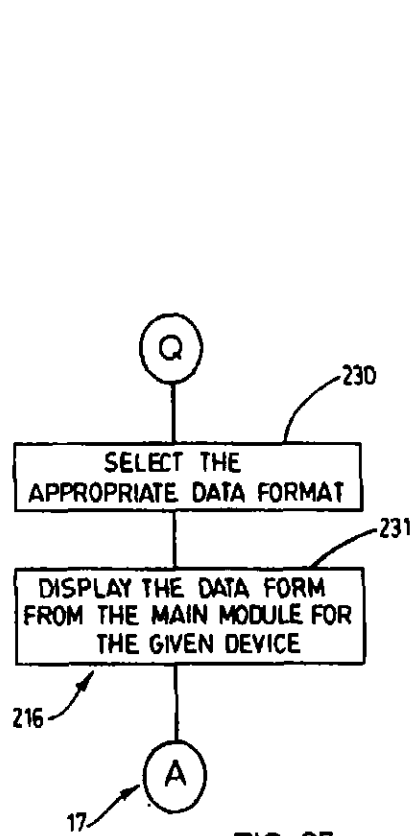


FIG. 25

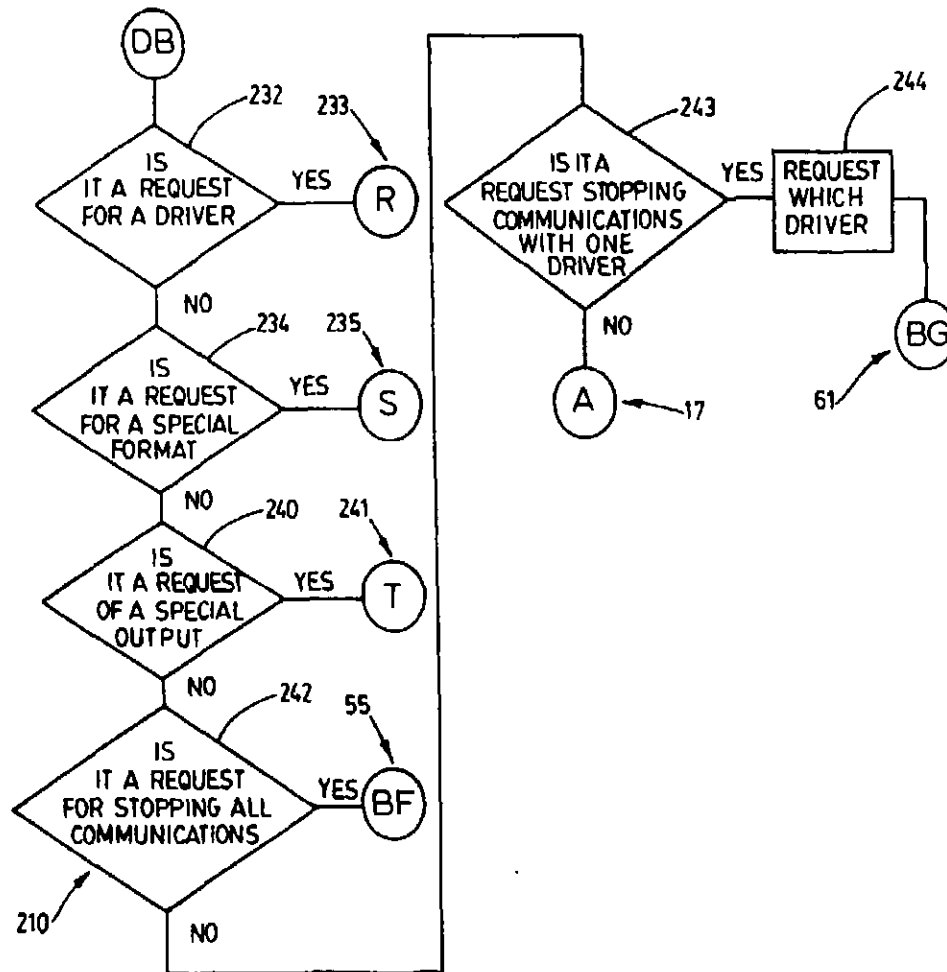
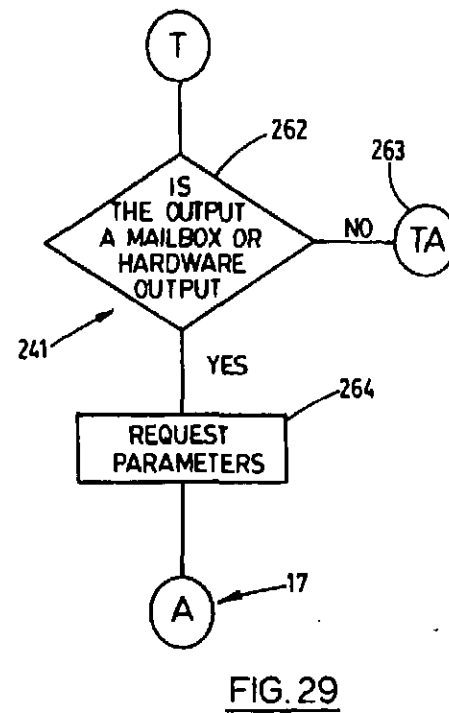
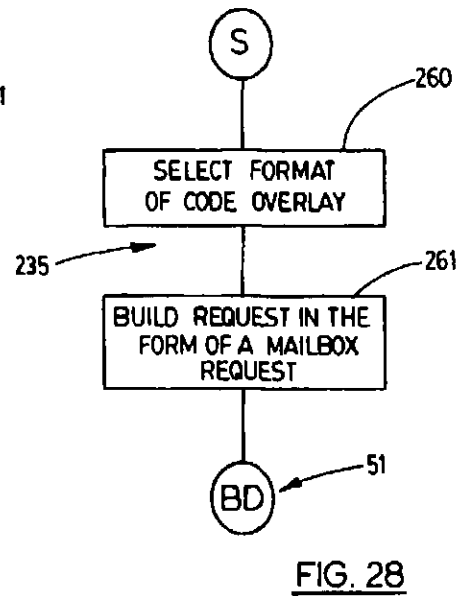
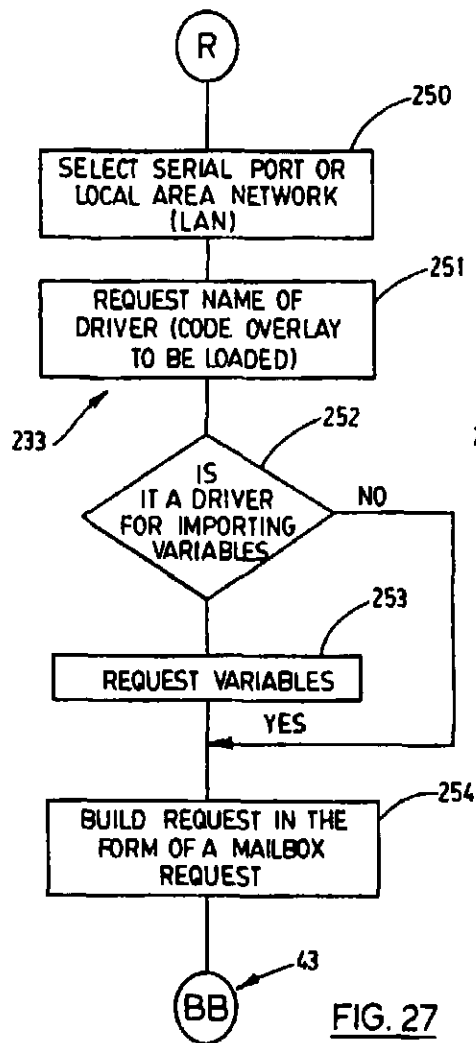
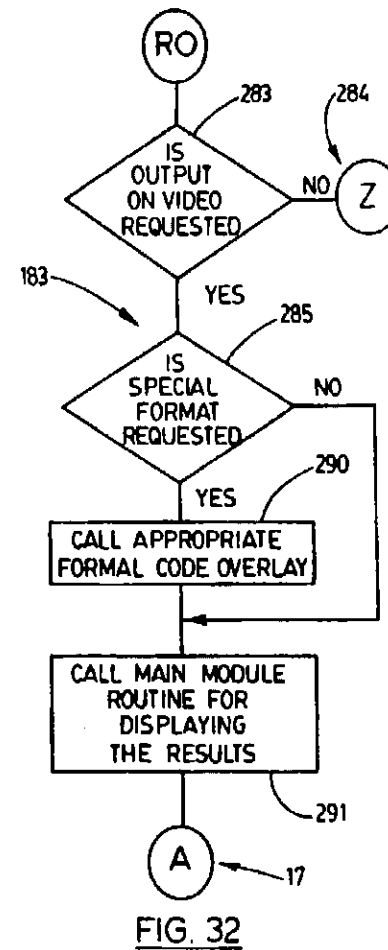
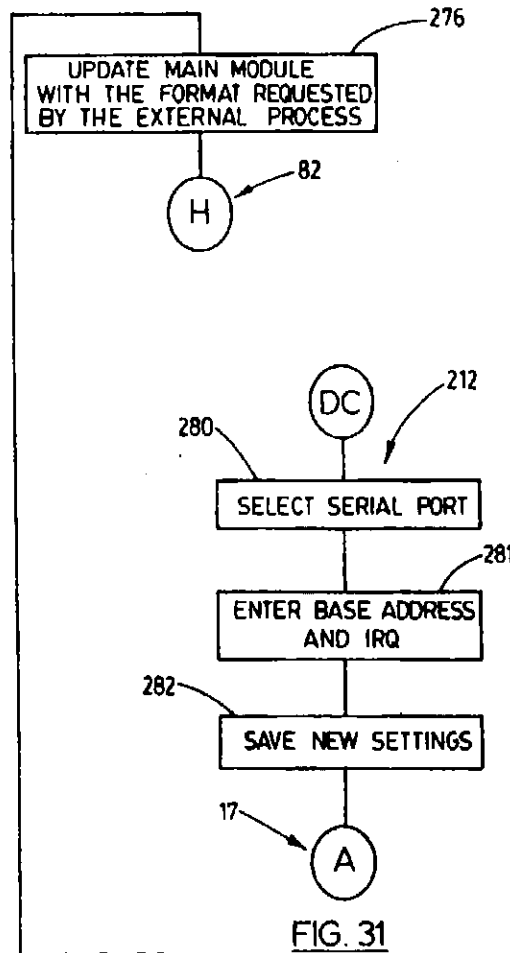
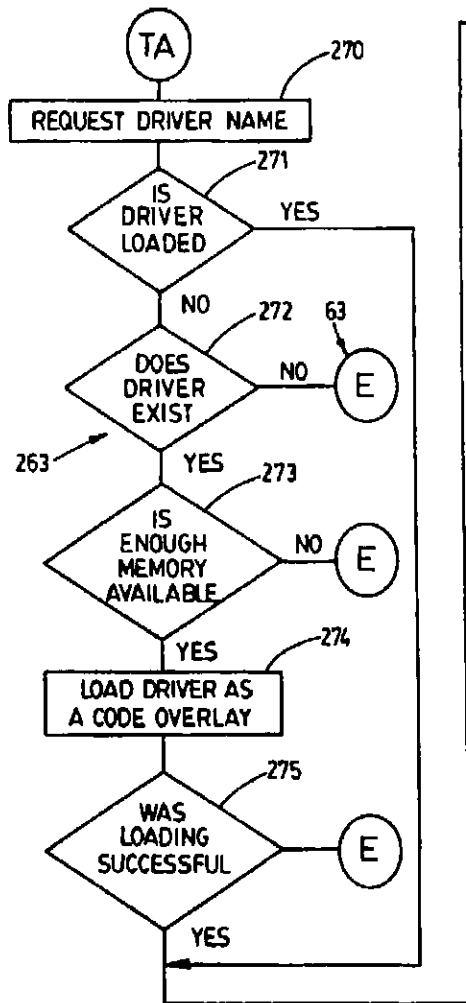
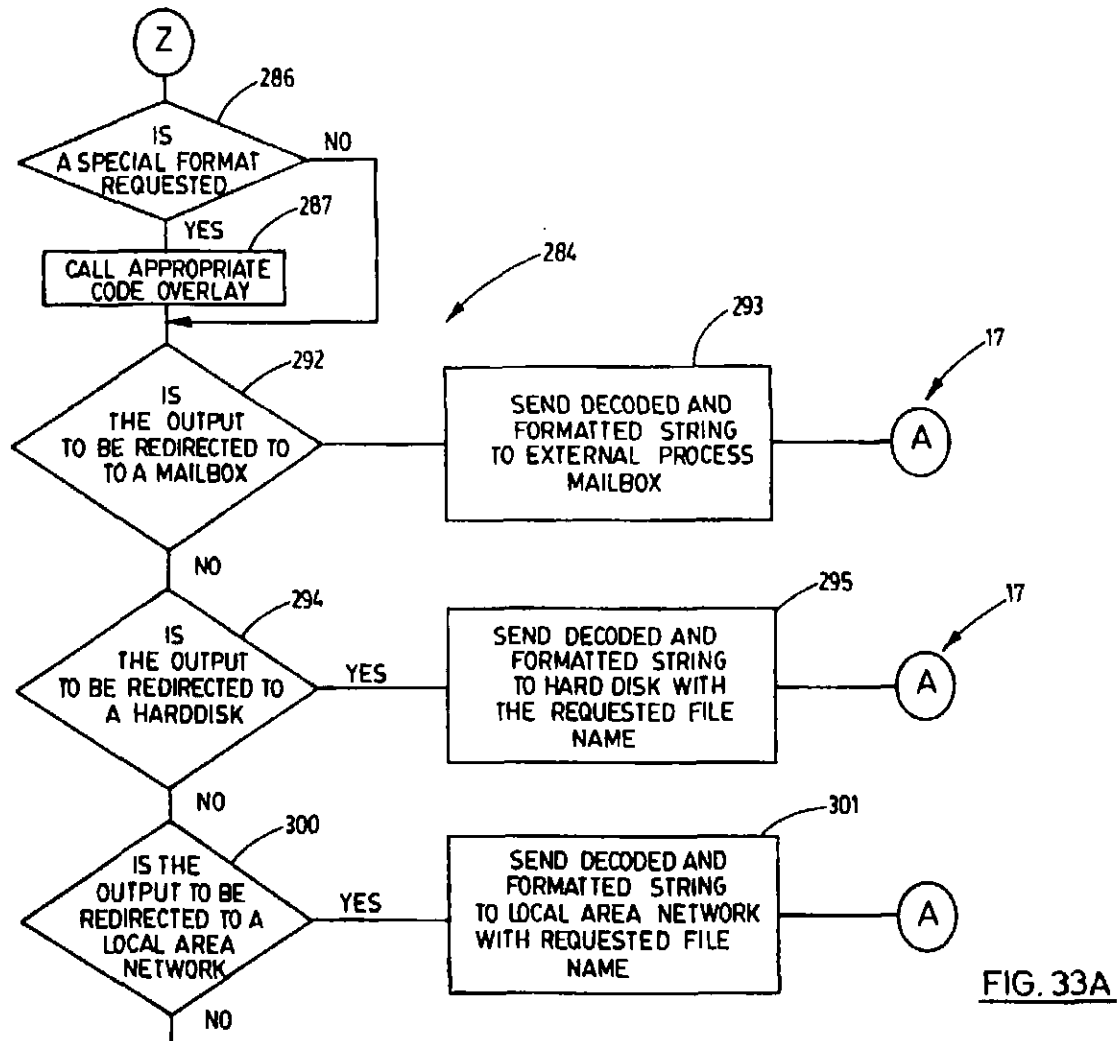


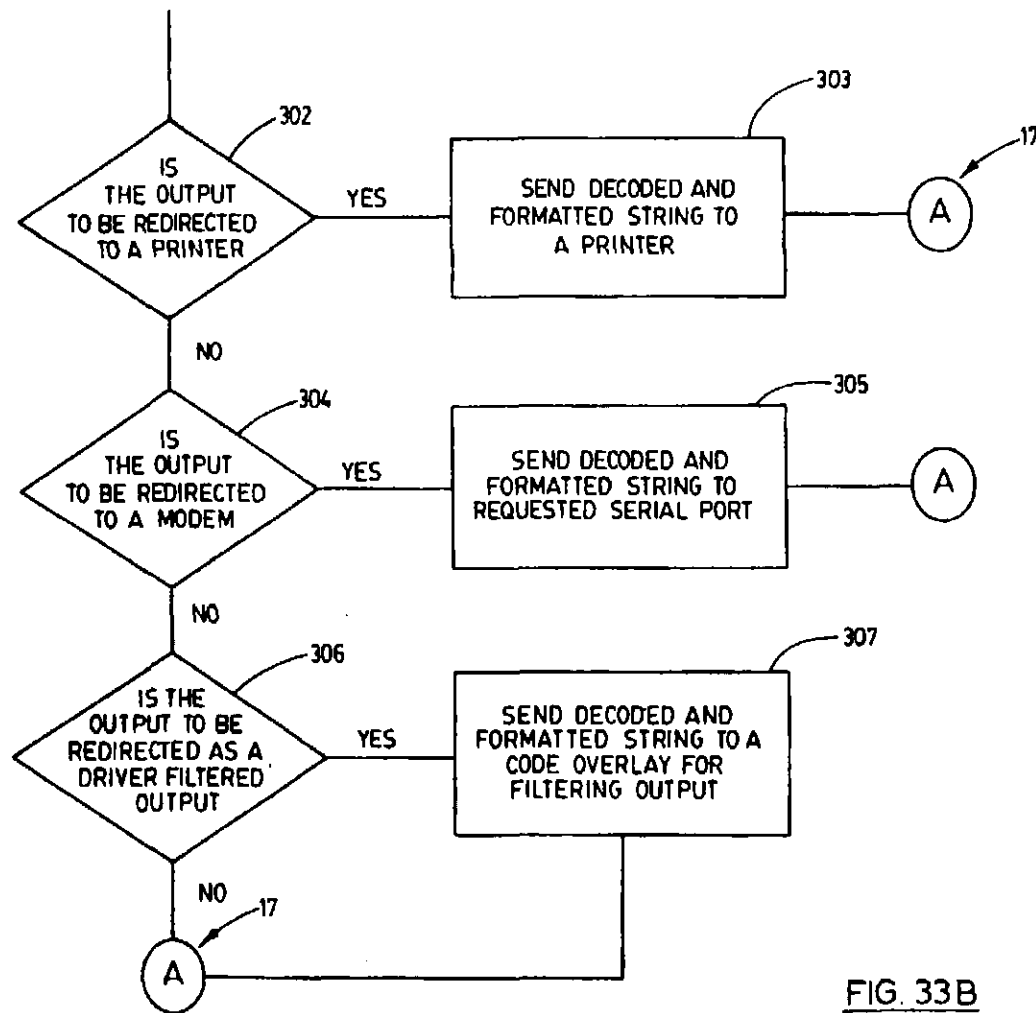
FIG. 26











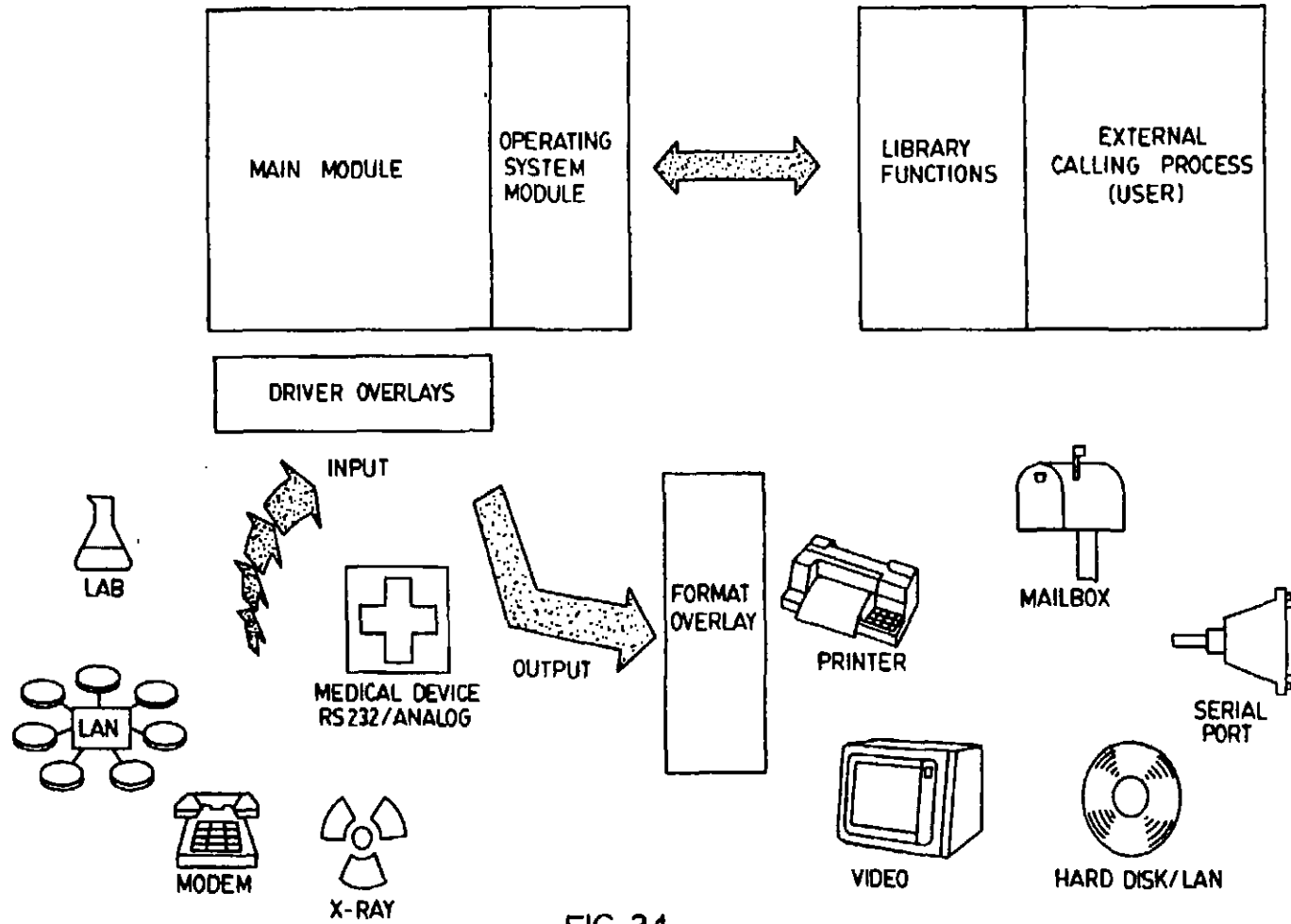


FIG. 34

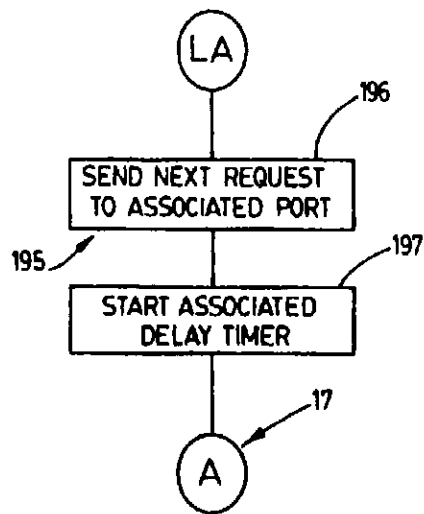


FIG. 35

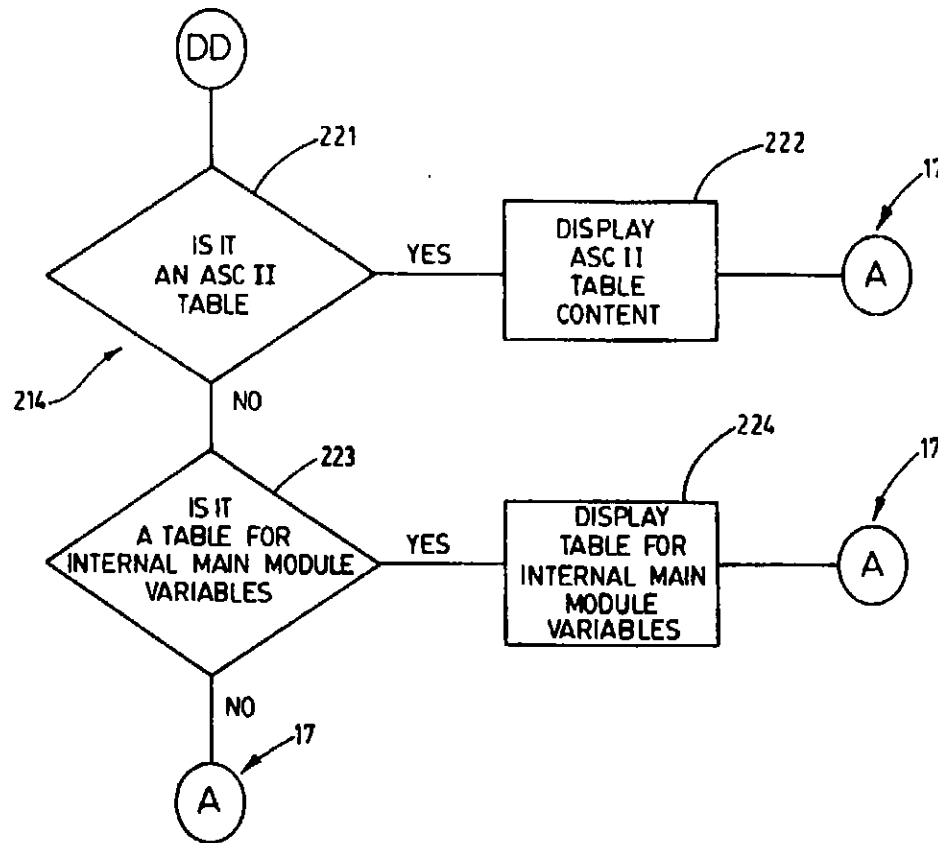


FIG. 36

5,161,222

1

## SOFTWARE ENGINE HAVING AN ADAPTABLE DRIVER FOR INTERPRETING VARIABLES PRODUCED BY A PLURALITY OF SENSORS

### BACKGROUND OF THE INVENTION

#### 1. Field of the Invention

The present invention relates to a method for interpreting variables produced by a plurality of sensors and more particularly to such a method adapted for use in communicating with sensors which are made integral with medical monitoring equipment, and which communicate by means of serial or analog protocols, and wherein the method provides a rapid means by which a user, such as a clinician, may interface with diverse monitors and thereafter receive the information produced by the monitors in a predetermined format and at a predetermined destination.

#### 2. Description of the Prior Art

The medical profession has long sought an effective method for receiving and thereafter evaluating clinical information which may be produced by various types of medical monitoring equipment or other diagnostic equipment and which could, for example be located in remote locations such as in a laboratory which is located in another area of a hospital. Further, the medical profession has long sought an effective method whereby this clinical information may be both freely exchanged to those individuals having a need to know same, such as hospital administrators, records clerks, and laboratory technicians, as well as have the information available for storage or transmission to any number of predetermined locations such as printers, video monitors, modems or local area networks, as conditions warrant. As should be understood the various medical monitoring equipment and other diagnostic devices used in laboratories and other clinical setting such as surgical suites, emergency rooms, etc. while having some similar component, have typically communicated with automated control assemblies, such as personal computers, by means of serial or analog protocols which have varied somewhat from one to the other. Therefore, software programs have been developed which have provided a means by which such information may be converted into a common format such as an interpretable ASCII data and thereafter exported or imported into another program for use therein. However, it should be readily recognized that the importation and exportation of interpretable ASCII data and the reformatting of same for use with a new software program is quite often time consuming and further may cause other problems which have detracted from the usefulness of this same technique.

The inconveniences and hardships occasioned to a patient by the delays which occur as a consequence of a clinician reviewing clinical data which may be produced by a plurality of medical sensors or other diagnostic equipment which may be positioned in remote location is readily apparent. Frequently, the information is received in piecemeal fashion and in various formats which require some time for evaluation. Further comparing and contrasting of the information is often necessary to determine the proper therapy for the malady under consideration. In addition to the foregoing the information has not, heretofore, been available to a clinician at a centralized location whereby the clinician can easily review same quickly and efficiently.

2

While the prior art has suggested various control assemblies including software programs which have provided a means by which particular clinical information such as medical history files, may be entered and thereafter used at diverse locations by clinicians and others for diagnostic and other purposes, such software programs have typically been cumbersome, complex and have not included a convenient means by which sensor information produced by a device which is foreign to the software program structure can be rapidly interfaced with the software program with a minimal amount of reprogramming of same.

Therefore it has long been known that it would be desirable to have a method and apparatus for interpreting variables produced by a plurality of sensors which communicate by means of serial or analog protocols and which can be employed in a wide variety of different institutional environments, and without the need for substantial alteration or rewriting of software programs which interface with same, and which can be manufactured and purchased at a relatively moderate cost, and which is both highly efficient in operation, and which further reduces to an absolute minimum the assorted problems associated with the monitoring and interpretation of clinical information which may be produced by a plurality of monitors and other diagnostic devices which may be positioned in remote locations.

### SUMMARY OF THE INVENTION

Therefore, it is an object of the present invention to provide an improved method for interpreting variables produced by a plurality of sensors, such as medical monitoring equipment and other diagnostic devices and the like.

Another object of the present invention is to provide a method for interpreting variables produced by a plurality of sensors, and particularly to a method which has utility when used in combination with a plurality of medical sensing devices, and which is adapted in response to an external process request to receive information from the medical sensors, and transmit same to a predetermined destination.

Another object of the present invention is to provide such a method which is operable to obtain the individual benefits to be derived from related prior art methods and devices while avoiding the detriments individually associated therewith.

Another object of the present invention is to provide such a method which is particularly well suited when used, in combination with medical monitoring equipment, to provide a clinician with assorted information relative to a patient's condition at a centralized location and which further may be easily reprogrammed in a fashion to permit the information to be transmitted to a plurality of predetermined destinations such as video monitors, printers, modems, local area networks, and the like.

Another object of the present invention is to provide such a method which is operable to deliver information regarding a patient's condition rapidly, dependably and efficiently while reducing to an absolute minimum the possibility of malfunction.

Another object of the present invention is to provide such a method which is of relatively moderate cost to purchase and maintain and which further is relatively inexpensive to operate.

Another object of the present invention is to provide such a method which is operating system independent,

5,161,222

3

and which is operable to interface with conventional software programs for purposes of further increasing the speed and efficiency with which the given information produced by a plurality of medical sensors can be produced during a predetermined time period.

Another object of the present invention is to provide such a method which is characterized by ease of employment, relative simplicity in its architecture, and which can be sold at a moderate price.

Another object of the present invention is to provide such a method which allows an operator to communicate with, and receive information from, a medical sensor foreign to the present invention, the present method including a means by which the operator may rapidly program the invention to receive the information produced by the foreign sensor.

Still another object of the present invention is to provide a method which may be manufactured as a software program or alternately manufactured as an integral subassembly of a piece of hardware such as in the nature of a computer chip which may be employed in a personal computer or which further may be manufactured in the manner of a retrofit.

Further objects and advantages of the present invention are to provide improved elements and arrangements thereof in a method for the purposes described which are dependable, economical, durable and fully effective in accomplishing its intended purposes.

These and other objects and advantages are achieved in a method for interpreting variables produced by a sensor which communicates by means of serial or analog protocols including interpreting an external process request for data information from the sensor, overlaying a predetermined adaptable driver which, when adjusted in a predetermined fashion, corresponds to the characteristics of the sensor, polling or listening to the sensor thereby receiving the data information requested, and transmitting the information to a predetermined destination.

#### BRIEF DESCRIPTION OF THE DRAWINGS.

FIG. 1 sets forth in detail the operation of loading the main module of the present invention into a personal computer including opening the object queue.

FIG. 2 sets forth in detail the operation of the object queue loop of the present invention.

FIG. 3 illustrates in detail the mailbox object process shown in FIG. 2.

FIG. 4 sets forth in greater detail the mailbox object process shown in FIG. 3.

FIG. 5 sets forth in greater detail the operation of the mailbox object process shown in FIG. 3.

FIG. 6 illustrates in greater detail the mailbox object process shown in FIG. 3.

FIG. 7 sets forth in greater detail the mailbox object process shown in FIG. 3.

FIG. 8 sets forth in greater detail the mailbox object process shown in FIG. 3.

FIG. 9 sets forth the subroutine for storing the parameters for redirecting all data of the external process and which is shown in FIG. 8.

FIG. 10 sets forth in greater detail the mailbox object process and more particularly the operation for stopping all communications and which is shown in FIG. 3.

FIG. 11 sets forth in greater detail the mailbox object process and more particularly the operation for stopping all communications with one driver and which is shown in FIG. 3.

4

FIG. 12 sets forth in greater detail the mailbox object process and more particularly the operation which is undertaken following a determination that a driver is not loaded and which is shown in FIG. 5.

FIG. 13 sets forth in greater detail the mailbox object process and more particularly the operation which is undertaken once it is determined that a device driver is being employed and which is shown in FIG. 5.

FIG. 14 shows in further detail the mailbox object process and more particularly the operation following a determination that the driver is operable to import data and which is shown in FIG. 13.

FIG. 15 shows in greater detail the mailbox object process and more particularly the operation of generating an error message and which is shown in FIG. 4.

FIG. 16 shows in greater detail the mailbox object process and more particularly the operation of notifying an external process of the status of the system by way of a mailbox and which is shown in FIG. 14.

FIG. 17 sets forth in further detail the mailbox object process and more particularly the operation for creating a polling timer associated with the driver of the present invention and which is shown in FIG. 14.

FIG. 18 sets forth in further detail the object timer process of the present invention and more particularly the operation of the object timer process and which is shown in FIG. 2.

FIG. 19 sets forth in further detail the object timer process of the present invention and more particularly the operation of determining whether the driver type is a device driver and which is shown in FIG. 18.

FIG. 20 sets forth in further detail the object timer process of the present invention and more particularly the operation following a determination that a non-device driver is not being utilized for importing data and which is shown in FIG. 19.

FIG. 21 sets forth in further detail the object timer process of the present invention and more particularly the operation following a determination that a message does not relate to a polling timer and which is shown in FIG. 18.

FIG. 22 sets forth in further detail the object timer process of the present invention and more particularly the operation following a determination that a driver is not a pollable one as shown in FIG. 18.

FIG. 23 sets forth in further detail of the keyboard object process shown in FIG. 2.

FIG. 24 sets forth in further detail the keyboard object process, and more particularly the operation following an affirmative status check request and which is shown in FIG. 23.

FIG. 25 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that a device is not an input device as shown in FIG. 24.

FIG. 26 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that keyboard interaction with the main module is requested and which is shown in FIG. 23.

FIG. 27 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that a message is a driver request and which is shown in FIG. 26.

FIG. 28 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that a message for a special



5,161,222

5

format has been received and which is shown in FIG. 26.

FIG. 29 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that a message is for a special output and which is shown in FIG. 26.

FIG. 30 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that an output is not a mailbox or hardware output as shown in FIG. 29.

FIG. 31 sets forth in further detail the keyboard object process, and more particularly the operation following a determination that port assignments have been requested and which is shown in FIG. 23.

FIG. 32 sets forth in further detail the keyboard object process, and more particularly the operation for calling an appropriate code overlay and which is best illustrated by reference to FIG. 19.

FIGS. 33A and 33B set forth in further detail the keyboard object process, and more particularly the operation following a determination that a video output has not been requested and which is best illustrated by reference to FIG. 32.

FIG. 34 is a pictorial representation of the various modules which make up the present invention.

FIG. 35 sets forth in further detail subroutine LA of the present invention.

FIG. 36 sets forth in further detail subroutine DD of the present invention.

#### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a method for interpreting variables produced by a plurality of sensors and which communicate by means of serial or analog protocols, and more particularly to such a method for receiving, decoding, and delivering at a predetermined location, data information which is produced by a plurality of medical sensing devices and other diagnostic equipment and which is used by a clinician in the diagnosis of various maladies which affect a patient. As should be understood the hardware components which employ the present invention include a personal computer, medical monitoring equipment, and other devices such as local area networks, modems, computer printers, video monitors and hard disk storage devices, which are pictorially illustrated in FIG. 34 and which are well understood by those skilled in the art.

In accordance with the primary aspect of the above-identified method and which is operable to interpret variables produced by a plurality of sensors, it should be understood that the present method is an operating system independent software engine which is operable to read the variables from any medical device which communicates by means serial or analog protocols. Due in large measure to its relatively open architecture, the present method is also capable of processing data information received from any local area network and then treat this same information as if it was in fact a device. In addition to the foregoing, the present method is operable to transmit data in a user defined format to a multiplicity of different devices which may include computer printers, video monitors and hard disk storage devices as conditions warrant.

The present invention is operable, in summary, to read, decode, reformat, and output information from any medical sensing device, modem, or local area network and then redirect this information, which is in a

6

predetermined user defined format, to any printer, software mailbox, serial port, hard disk, local area network or video monitor. This is graphically illustrated in FIG. 34. In short, the present method is operable to act as a bridge or as a link between a clinician and a multiplicity of devices or sensors and further is adapted to interface with medical devices that are even presently non-existent today. This is achieved by means of an adaptable code overlay or skeleton which will be discussed in greater detail hereinafter.

To achieve the beneficial results discussed above, the present method, which includes a software engine, has several features which will be discussed, in general, in the paragraphs which follow. More particularly the software engine of the present invention has several features which are noteworthy, for example,

The software engine of the present invention is:

1. Object oriented with respect to four critical elements, these include time, mailboxes, keyboard and object queue.

2. The software engine interprets requests from an external calling process.

3. The software engine in response to the external calling process overlays an appropriate driver in response to the external calling process request.

4. The software engine actuates, as appropriate, the hardware communication interrupts if necessary.

5. The software engine polls [or listens to] the device, local area network, modem or other diagnostic equipment, as appropriate.

6. The software engine is operable, upon receiving the information from the device, local area network, modem or other diagnostic equipment to prepare an answer, in a user definable way, by means of a software overlay thereby providing a clinician with information in a format which assists him/her in an appropriate diagnosis of the patient's malady.

7. The software engine is adapted to output, upon the request of the user, to a requested mailbox, serial port, printer, local area network, or video monitor as appropriate.

In the paragraphs which follow the general characteristics of the software engine will be discussed. Further, an explanation will be provided regarding how messages are received by the software engine, and how same are decoded, reformatted and transmitted to a predetermined destination based on user commands supplied by the external process.

As best illustrated by reference to FIG. 34 the method of the present invention is made up of several modules and more particularly a main module, library functions, driver overlays, and format overlays. The main module, which is made up of various steps which are shown in FIGS. 1 through 33 is written in assembly language and is operable to deal with communications, interpretation of user's requests from the external process, the loading and calling of driver overlays, the polling of devices, and the formatting of data and its output to a predetermined destination. In addition to the foregoing, an operating system module is provided and which is operable to permit the main module to operate with various operating systems which are commercially available and which are well-known in the art.

The library functions, and which are contained in a user's library, is an independent program, separate from the main module and which is used by the external process, and which includes the functions which permit the external process to gain access to the main module.

5,161,222

7

More particularly, the library functions provide a means for internally transforming a user demand and sending it to the main module described above. This library function is completely dependent on the operating system environment while in use. At linking time, the appropriate library is chosen for the operating system in question and one skilled in the art can readily construct a library of their own choice.

The present method includes drivers and driver overlays or skeletons and which will be discussed in greater detail hereinafter. However, it should be understood that the term drivers, when used in this context, means discrete pieces of software that decode the information coming from any device, local area network, or modem. These are separate executable programs which are resident on the hard disk of the computer. Those skilled in the art will readily recognize that drivers can be written in high level languages or in assembly language. In this regard there are several different types of drivers. More particularly, and as discussed hereinafter in greater detail, the drivers employed with the present invention may include serial or analog drivers which include two discrete parts. The first discrete part of a particular device which includes a driver, is the communication setting for the device, that is, the request strings, baud rates, parity, numbers of steps etc. The second discrete element of a driver are the instructions necessary for the decoding of the answer of the sensor being polled. In the case of an analog driver, the first discrete part discussed above is normally replaced by calibration values. Further, and in the case of a local area network driver, a LAN driver can be of three types. The driver can be of a type which retrieves data from a network server, in this case the driver knows the particular path for the data, as well as the instructions to decode it. Further the LAN driver can also emulate tasks, and can read data from a mainframe computer. Thereafter, the LAN can retrieve the data from a network server as noted above. Further, a LAN driver can operate, as described above, but may be further rendered operable to request data from a modem.

Specific output drivers which may be employed with the present invention include two discrete types, that is (1) a driver which can format an answer which is coming from a sensor or device in a user defined format; and (2) a driver which decides where an output is directed, that is, to a printer, serial port output, hard disk, mailbox, etc.

The method of the present invention is operable to store a library of drivers which allows it to communicate with a wide variety of existing medical devices and other diagnostic equipment. Further driver skeletons which may be written in C language and/or in assembly language are provided, and which permit a hospital technician or a programmer to write a proper interface between the main module and any new medical monitoring device which may not be resident in the library of drivers provided. The present invention provides a convenient means whereby, with as little as ten lines of code, an inexperienced programmer can add a driver to the main module library. In this regard, an example of a driver skeleton including the lines of code necessary to enter same are provided, in the example, which will be discussed in greater detail hereinafter.

As should be understood the main module of the present invention and which is illustrated in detail in FIGS. 1-33 is an object oriented system. In this regard the objects are of several types. These include input-

8

/output objects, timer objects, keyboard objects and an object queue. As best seen by reference to Figs. 2 and 34 the communications between the main module and an external calling process or calling task, such as a user [not shown] are achieved through mailboxes. As should be understood, however, the external calling process is a separate executable program which is operable to interface with the main module through the mailboxes and such external calling process programs are commercially available. Examples of such programs are the programs entitled "Atlantis", "Sensor", and "Helena" and which are commercially available through the HMP Corporation of Delaware. However it should be understood that any number of different external process programs may be developed and which are operable to gain access to the main module through the mailboxes. This, of course, permits a user to customize the present invention to provide various functions not described herein. In the present invention however, and when a user issues an external process request or call to the main module, the answer from the main module is returned to the library function issuing the call. Further the main module may respond, on some occasions, through software interrupts thereby preventing the calling task from waiting in loops until an answer is ready. With regard to the timer objects it should be understood that any polling of a sensor device, and which is repetitive, by nature, is associated with a first timer object. Further, and when an answer is expected, a second timer object is automatically activated and which informs the main module when it is suppose to begin decoding the answer returned from the sensor device. Also, and with respect to the keyboard objects, if an operator has a need to know the internal state of the main module, a series of menus and displays are provided which illustrate the actual state of same. Further, the keyboard objects transfer information between the end user and the main module. Finally, and as best understood by reference to FIG. 2 an object queue loop permits the main module to handle with utmost efficiency the different objects described above.

As discussed above, the main module is an interrupt driven software engine. As should be understood this means that any serial communications are hardware interrupt driven. Further, and when a user, including a process or task, requests data from a device or sensor through a specific serial port, the associated hardware interrupt is actuated. Further, the main module polls, if necessary, the device during, or in response to, the first timer object, and sets a delay, or second timer object, for obtaining an answer. Upon receiving this answer it is directly stored into a buffer by code which is executed by the actuated hardware interrupt. In addition to the foregoing, and when the delay timer is off, the main module reads the answer and calls the proper driver for the decoding of the string.

There are a multiplicity of different kinds of messages that a user, including a process or calling task, can transmit to the main module, these include messages which relate to the handling of communication messages, the request for data messages, and the request for specific output messages. Further, a user, that desires to communicate with the main module shall, open communications with the main module by way of a "handle", that is, a communications identification for all their future communications. Furthermore, the user must furnish to the main module an address of the routine for reception of the data. At the end of a session, a message can clear

9

5,161,222

all handles and terminate communications with a calling process or task. Messages may be of various types including a first type, which is used to request data from a specific device on a given serial port, a second type, which carries the name of a local area network driver to be loaded and which is to be operated by the main module, and a third type, which deals with burst messages which are to be read from a server or distant device.

As earlier discussed the main module is operable to provide a means whereby a user can request that a specific format be applied to the data information supplied by the sensor. In this regard, a special message provides to the main module the name of the driver that formats the data in the defined way. Further, a second message expresses the desire of the user to have the output sent to a specific predetermined destination such as a mailbox, serial port, a printer, a file on a hard disk or to a video monitor as conditions warrant. As earlier discussed the main module includes an object queue loop which provides a first in/first out handling of all requests for data information.

As best illustrated by reference to FIG. 1 the method of the present invention is loaded into a personal computer, which is not shown, and an operator initiates the present invention at the step labeled "enter" and which is generally indicated by the numeral 10. Upon executing the enter command, the present invention is operable to determine whether a proper operating environment is present 11. If a proper operating environment is absent, the program will exit 12 and will give an appropriate message to the operator. However, in the alternative, and if the proper operating environment is present, the present method makes a mailbox available for exchange with the external calling process 13 which may include such software programs as "Atlantis", "Sensor" and "Helena" and which were discussed above. The method next determines whether a user interface is required 14. If a user interface is required the system is operable to load a keyboard object 15. Upon loading the keyboard object the present method is operable to open the object queue loop 17 and which is set forth in greater detail in subroutine A. Further, and if the user interface is not required the present method is operable to return to the object queue loop which is illustrated in subroutine A in FIG. 2.

The object queue loop 17 and which is best illustrated by reference to FIG. 2, is operable to read the object queue 21 and determine if the object is a mailbox object 22. If it is a mailbox object 22, the object queue is operable to direct the request for data to subroutine B and which is generally indicated by the numeral 23. However, and if the request for information is not of the mailbox type, the object queue loop determines whether the object is a timer object 24. If it is a timer object, then in that event the object queue loop directs the request for information to subroutine C, and which is indicated by the numeral 25. Moreover, and if the object is not a timer object 24, the object queue loop determines whether the object type is a keyboard object 30. If it is a keyboard object, then, in that event, the information is directed to subroutine D and which is generally indicated by the numeral 31. As earlier discussed, the object queue loop is adapted to process the information received from the external process request and then transmit that information to the appropriate subroutine 23, 25, or 31 as appropriate. The object queue loop is the only loop in the present invention and

10

is operable to handle all external process requests on a first in first out basis.

As best illustrated by reference to FIG. 3 subroutine B 23 is operable to process an external process request which has been determined by the object queue loop to be a mailbox object. In this regard subroutine B which may be from time to time referred to as a mailbox object process, determines whether the message received is initiating communication 40. If indeed the external message is initiating communications with the mailbox then, in that event, the message is diverted to subroutine BA, and which is generally indicated by the numeral 41. If the message from the external process is not initiating communication with the mailbox then subroutine B determines whether it is a message from an external process which is loading a driver. If it is, then in that event, the message is diverted to subroutine BB and which is generally indicated by the numeral 43. If the message from the external process is not loading a driver, subroutine B next determines whether it is a message initiating the transmission of data 44. If this is the case then, in that event, the message is diverted to subroutine BC and which is indicated by the numeral 45. If the message does not initiate the transmission of data, then subroutine B next determines whether it is a message for a special format 50. If this is indeed the case then in that event the message is diverted to subroutine BD and which is generally indicated by the numeral 51. Further, and if the message does not request a special format then, in that event, subroutine B determines whether it is a message for a special output 52. If so, then the message is diverted to subroutine BE and which is generally indicated by the numeral 53. In the alternative, and if the message is not a message for a special output, then, in that event, subroutine B determines if it is a message stopping all communications 54. If so, then the message is directed to subroutine BF 55. If not, subroutine B determines if it is a message for stopping communications with just one driver 60. If so, then in that event, it is directed to subroutine BG and which is generally indicated by the numeral 61. If it is none of these messages then, in that event, subroutine B returns to the object queue loop and which is generally indicated by the numeral 17 in FIG. 2.

Subroutine BA, and which is generally indicated by the numeral 41 is best illustrated by reference to FIG. 4. As earlier discussed, and in response to an external process request which has been identified as a mailbox object type 22 and which has been diverted to subroutine B 23 and wherein the message has been determined by subroutine B to be initiating communications 40, subroutine BA is operable to first determine if the external process request is already known 62. If the external process is known, then, in that event the message is directed to subroutine E, and which is generally indicated by the numeral 63 and which is best illustrated by reference to FIG. 13. As shown therein, subroutine E generates an error message 162 to the end user. If the external process is not known, then, in that event, subroutine BA determines if the program is free enough to handle the external process 64. If not, subroutine BA is operable to divert the request for data to subroutine E 63 and thereby generate an error message to the end user. If the process is able to handle the external process, then, in that event, subroutine BA is operable to open a handle 70. As earlier discussed a handle is an identification tag, of sorts. Subroutine BA stores the identification or handle in memory. The step of opening

5,161,222

11

the handle and storing the identification of the external process is indicated by the numeral 70. Upon completing the step of identifying the external process, subroutine BA sends an "okay" message and the identification "handle" to the external process through the mailbox 71. Upon completion of this task, subroutine BA returns to the object queue loop and which is generally indicated by the numeral 17.

As earlier discussed the mailbox object process, and more particularly subroutine B, and which is generally indicated by the numeral 23 in FIG. 3, is operable to determine if an external process message is loading a driver 42. If the external process is loading a driver, the message is diverted to subroutine BB and which is generally indicated by the numeral 43 in FIG. 5. As shown therein, subroutine BB is initially operable to determine if the driver type is a device driver 72. If this is not the case, then, in that event, the message is diverted to subroutine F and which is generally indicated by the numeral 73. If the message is for a device type driver, then, in that event, subroutine B determines if the driver is loaded 74. If the driver is not loaded, then, in that event, the message is directed onto subroutine G 75 and which will be discussed in greater detail hereinafter. If the driver is loaded, then, in that event, subroutine B is operable to decode the variables requested by the external process 76, update the driver overlay with the variables requested by the external process 80, and update the main module with the variables requested by the external process 81. Upon completion of this last task subroutine BB, 43 returns to subroutine H 82 and which will be discussed in greater detail hereinafter.

The mailbox object process and which is generally indicated by the numeral 23, is operable to determine if a message from the external process was initiating the transmission of data 44. If this answer is in the affirmative then, in that event, the message is diverted to subroutine BC and which is generally indicated by the numeral 45 in FIG. 6. As shown therein subroutine BC is operable to initiate the transfer of data from the main module to the external process 83 and then return to subroutine H and which is generally indicated by the numeral 82.

The mailbox object process and more particularly subroutine B 23 is operable to determine if a message from an external process is requesting a special format 50. If this answer is in the affirmative, then, in that event, the message is diverted to subroutine BD and which is generally indicated by the numeral 51 in FIG. 7. As shown therein, subroutine BD is operable to determine if the driver is loaded 83. If the response to same is in the affirmative, then, in that event, subroutine BD is adapted to update the main module with the format required by the external process 92 and return to subroutine H and which is generally indicated by the numeral 82. However, and if the driver is not loaded, subroutine BD is operable to next determine if the driver exists 84. If the driver does not exist, an error message is generated by means of subroutine E and which is generally indicated by the numeral 63. In the alternative, and if the driver does exist subroutine BD next determines if enough memory is available for processing the request 85. If a response to same is in the negative, then, in that event, subroutine E is executed and which is generally indicated by the numeral 63. In the alternative, and if sufficient memory is available to process the request 85 then, in that event, subroutine BD is operable to load the driver as a code overlay 90.

12

Subroutine BD then determines whether loading was successful 91. If not, an error message through subroutine E 63 is executed. In the alternative and if loading was successful, then, in that event, subroutine BD is operable to update the main module with the format required by the external process 92 and then return to subroutine H 82, as earlier discussed.

The mailbox object process 23 is operable, as earlier discussed, to determine if a message received from an external process is a message for a special output 52. If this answer is in the affirmative, then, in that event, the message from the external process is diverted to subroutine BE and which is generally indicated by the numeral 53 in FIG. 8. As shown therein, subroutine BE is operable to first determine if the message from the external process is a mailbox or hardware output 93. If the answer is in the affirmative, then, in that event, the message is diverted to subroutine BH and which is generally indicated by the numeral 94. In the alternative and if the answer is a negative response, then, in that event, subroutine BE next determines if the driver is loaded. If the driver is loaded, then the subroutine is operable to determine if loading was successful 102. If not, an error message through subroutine E 63 is generated. If loading was successful, then, in that event, the message is transmitted to subroutine H, and which is generally indicated by the numeral 82. If the driver is not loaded then subroutine B is operable to thereafter determine if enough memory is available for processing the request 100. If sufficient memory is not available, an error message by means of subroutine E 63 is provided. If indeed sufficient memory is available, subroutine BE loads the driver as a code overlay 101 and thereafter determines whether loading was successful 102. If not, subroutine E 63 is executed thereby generating an error message. If loading was successful, then, in that event, the message is transmitted to subroutine H and which is generally indicated by the numeral 82.

Subroutine BH and which is generally indicated by the numeral 94 is operable, upon receiving a message indicating that a message is a mailbox or hardware output 93 from subroutine BE 53, to execute and store the parameters for redirecting all data of the external process 103 and return the information to subroutine H and which is generally indicated by the numeral 82.

Subroutine BF and which is generally indicated by the numeral 55 is best illustrated by reference to FIG. 10, and is operable to receive a message from the external calling process which is directing the present invention to stop all communications 64. In this regard, subroutine BF is adapted to reset all parameters concerning the external process and free its handle 104 and thereafter determine if each of the drivers previously requested by the external process are in use by another external process 105. For the drivers which are in use, nothing occurs. For those drivers not in use then, in that event, subroutine BF is operable to reset the main module for that driver and stop any polling timers 110. Further, and following step 110, subroutine BF is adapted to determine if the driver is a device driver 111. If not, the information is returned to subroutine H 82. However, and if it is a device driver, then, in that event, subroutine BF executes a command to free a serial port 112 as best seen by reference to FIG. 10.

As earlier discussed, the mailbox object process and which is generally indicated by the numeral 23 in FIG. 3 is operable to determine if the message from the external process is stopping communications with one driver

13

5,161,222

14

60. If the answer to the message is in the affirmative, then, in that event, the information is sent to subroutine BG and which is generally indicated by the numeral 61 in FIG. 11. As shown therein, subroutine BG is operable upon receiving the information, to update the main module parameters for the particular process 113 and thereafter inquire whether the driver is in use by another process 114. If this determination is in the affirmative, the information is directed to subroutine H 82. If the answer is in the negative, then, subroutine BG is operable to reset the main module parameters for the driver effected and stop polling 115. Further, subroutine BG is operable to determine if the driver is a device 116. If the answer is in the negative, then, in that event, subroutine BG is operable to direct the information to subroutine H 82. If the answer is in the affirmative, subroutine BG frees up a serial port 117 and then proceeds to direct the information to subroutine H. This is best understood by a study of FIG. 11.

FIG. 12 illustrates subroutine G 75 and which receives information following a determination in subroutine BB, and which is generally indicated by the numeral 43, that a driver is not loaded to memory 74. Upon receiving information, subroutine G determines if the requested port is free 120. If the answer is in the negative then, in that event, an error message is generated by means of subroutine E 63. If the answer is in the affirmative, subroutine G next determines if the driver exists 121. If not, then, in that event, subroutine E is utilized and an error message is generated. In the alternative, and if the driver does exist, subroutine G determines if enough memory is available to process the information 122. If sufficient memory is not available an error message is generated by means of subroutine E. Alternatively and if the answer is in the affirmative, subroutine G is operable to load the driver as a code overlay 123 and determine if loading was successful by means of step 124. If loading was not successful then, in that event, subroutine E 63 generates an error message. However, and if loading was successful, subroutine G sequentially reads the driver communication settings 125, decodes the variables requested by the external process 130, updates the main module with the variables requested by the external process 131, sets the ports and hooks the corresponding IRQ of the hardware, if necessary, 132 and creates a polling timer object associated with the device 133. Further and upon the successful creation of a polling timer 133, subroutine G determines if the device is pollable 134. If not, the information is passed to subroutine H. However and if the device is pollable, then, in that event, subroutine G creates a delay timer object associated with the device 135 and then sends a request to the device 140 and starts a delay timer which is flagged to a handshake 141. Following the handshake, which initiates communication, subroutine G returns to the object queue loop 17 as best illustrated by reference to FIG. 2.

As earlier discussed the mailbox object process 23 is operable to determine if a message received from an external process is loading a driver 42. Further and if the driver is not a device driver 72 as determined by subroutine BB 43 then, in that event, subroutine F is operable to process the information and which is generally indicated by the numeral 73 in FIG. 13. Initially subroutine F determines whether the driver is loaded 142. If subroutine F determines that the driver is not loaded, then, in that event, it determines whether the driver exists in memory 143. If the driver does not exist

in memory, then, an error message by means of subroutine E 63 is generated. However, and if the driver does not exist in memory, subroutine F subsequently determines if enough memory is available to handle the external process request 144. A negative determination of this query generates an error message by means of subroutine E 63. However, and if enough memory is available, then, in that event, subroutine F loads the driver as a code overlay 145 and thereafter determines whether loading was successful 150. Failure to load successfully generates an error message by means of subroutine E 63. Successful loading results in subroutine F thereafter determining if the driver under consideration is importing data 151. If not, the information is directed to subroutine K and which is generally indicated by the numeral 152 in FIG. 14. However, and if the driver is for importing data, then, in that event, subroutine F decodes the variables requested by the external process 153, updates the driver overlay with the variables requested by the external process 154, and updates the main module with the variables requested by the external process 155. Following the last step, subroutine F returns the information to subroutine J and which is indicated by the numeral 156.

Subroutine K and which is generally indicated by the numeral 152 in FIG. 14 is operable to process information from subroutine F following a determination that a driver is not being utilized for importing data 151. In this regard, subroutine K is operable to determine if the message is a timer object dependent driver 160. Following an affirmative determination, subroutine K is operable to transmit the information to subroutine J and which is generally indicated by the numeral 156. A negative determination causes subroutine K to call the appropriate code overlay 161 and then transmit the information to subroutine H and which is generally indicated by the numeral 82. Further, and as best illustrated by reference to FIG. 15, subroutine E is operable to generate an error message at various points in the present method. As best illustrated by reference to FIG. 16, subroutine H is operable upon receiving information to notify the external process 163 of an okay status via a mailbox and then return to the object queue loops 17 and which was earlier discussed in greater detail. Further, and as best illustrated by reference to FIG. 17, subroutine J and which is generally indicated by the numeral 156, upon receiving information, is operable to create a polling timer associated with the driver 164 and then return the information to subroutine H and which is indicated by the numeral 82 in FIG. 16.

Subroutine C and which is generally indicated by the numeral 25 illustrates, in general, the object timer process and which is generally indicated by the numeral 25 in FIG. 2. The object timer process C is operable to initially determine whether the external process request is a polling timer request 170. If this answer is in the negative then, in that event, the external process request is directed to subroutine L and which is generally indicated by the numeral 171. Subroutine L will be discussed in greater detail hereinafter. Further, and if the message is a polling timer request, then, in that event, subroutine C is operable to restart the polling timer 172 and determine if the corresponding driver is a pollable one 173. As should be understood, some drivers are not pollable. If the driver is not pollable then, in that event, the information is directed to subroutine LB and which is generally indicated by the numeral 174. In the alternative, and following a determination that the driver is

5,161,222

15

pollable, subroutine C next determines if the driver is a device driver 175. If the device is not a device driver, then, in that event, the information is directed to subroutine M, and which is generally indicated by the numeral 176. Further, and if the device is a device driver, subroutine C sends a first polling request to the proper port 177 and then starts the associated delay timer 178. Following completion of the start the associated delay timer, 178, subroutine C returns to the object queue loop and which is generally indicated by the numeral 17 in FIG. 2.

FIG. 19 illustrates subroutine M 176 and which receives information from the object timer process and which is generally indicated by the numeral 25 in FIG. 18. Subroutine M receives information from subroutine C and more particularly the information that the device under consideration is not a device driver 175. Following receipt of this information, subroutine M initially determines whether the non-device driver is for importing data 80 and if it is not, then, in that event, the information is directed onto subroutine O, and which is generally indicated by the numeral 181. If the device is a non-device driver, then, in that event, subroutine M calls the proper code overlay 182 and directs the information onto subroutine RO, and which will be discussed hereinafter as numeral 183.

Subroutine O, and which is generally indicated by the numeral 181, in FIG. 20, is operable to determine, following a negative determination that the non-device driver is for importing data 180 to call an appropriate code overlay 184 and then determine if the message being received is a burst message 185. If the message is a burst message, then, in that event, the information is directed to a video screen 186. The subroutine then returns to the object queue loop 17. However, and in the alternative, if the information is not a burst message, then, in that event, subroutine O determines if the message is an operating system call request 190. If indeed, it is an operating system call request, then, in that event, subroutine O executes the system call and inquires if there is an output to redirect. If indeed there is an output to redirect, then, in that event, the information is directed to subroutine RO and which is generally indicated by the numeral 183. However, and if there is no output to redirect, subroutine O returns to the object queue loop A and which is generally indicated by the numeral 17.

As best illustrated by reference to FIG. 21, subroutine L, and which is generally indicated by the numeral 171, is operable to receive information from the object timer process 24, and which is generally indicated by the numeral 25 in FIG. 18. Subroutine L is operable, upon receiving information from subroutine C, to validate the answer from the device 193 and determine if it is a final request 194. If it is not a final request 194, then, in that event, the information is transmitted to subroutine LA, and which is generally indicated by the numeral 195 in FIG. 21. If the information received is a final request, then, subroutine L is operable to determine if it was a handshake initiating communications 200. If it was not, then, in that event the information is directed to subroutine LB and which is generally indicated by the numeral 174 in FIG. 22. If, in the alternative, the final request was a handshake initiating communication, then, in that event, subroutine L initiates an answer to the device with an okay through the external process by going through the mailbox 201 and then

16

returns to the object queue loop which is generally indicated by the numeral 17.

Subroutine LA and which is generally indicated by the numeral 195 in FIG. 35 is operable, upon receiving information to send the next request for information to an associated port 196, and then start the associated delay timer 197. Subroutine LA then returns to the object queue loop 17.

Subroutine LB and which is generally indicated by the numeral 174, in FIG. 22, is operable to receive information from the object timer process and which is generally indicated by the numeral 25 in FIG. 18, and more specifically to receive information regarding whether the corresponding timer is a pollable one as provided at 173. If the driver is not a pollable one, then, in that event, the information is processed through subroutine LB. More specifically, subroutine LB, procures the data from the IRQ of the hardware and then returns to subroutine RO, and which is generally indicated by the numeral 183 and which is illustrated with more particularly in FIG. 32.

FIG. 23 illustrates in more detail the keyboard object process and which is generally indicated by the numeral 31 in FIG. 2. Subroutine D is operable initially to determine whether the request from the external process is a status check request 203, and if this is the case, the information is directed to subroutine DA and which is generally indicated by the numeral 204. Further, and if the information from the external process is not a status check request 203, then, in that event, subroutine D is operable to determine if keyboard interaction with a main module is requested 205. If a response to this question is in the affirmative, then, in that event, the information is directed to subroutine DB and which is generally indicated by the numeral 210 in the alternative and if a response is in the negative, then, in that event, subroutine D is operable to determine if port assignments are requested 211. If this is the case, then, in that event, the information is directed onto subroutine DC 212. If port assignments are not requested, then, in that event, subroutine D is operable to determine whether the main module utilities are requested. If the answer is in the affirmative, then, in that event, the information is directed to subroutine DD 214. As should be understood subroutine DD and which is shown in FIG. 36, is operable to determine if an ASCII table is requested 221, or if a main module variable table is requested 223. Upon displaying the requested information 222, and 224, respectively, subroutine DD returns to the object queue loop 17 and which is best illustrated in FIG. 2.

FIG. 24 illustrates subroutine DA, and which is generally indicated by the numeral 204. Subroutine DA is operable to receive the information from the keyboard process request, and which is generally indicated by the numeral 31, in FIG. 23. Subroutine DA is operable, upon receiving the information regarding whether the external process is requesting a status check 203 and thereafter determine if the device is an input device 215. If the device is not an input device, then in that event the information proceeds to subroutine Q and which is generally indicated by the numerals 216. However, and if the device is an input device, then, in that event, subroutine DA is operable to select the appropriate serial port 217 select the appropriate data format 218 and; thereafter display the data form 219 from the main module for the given device 219. Upon completion of the last step of displaying the data, subroutine DA 204

5,161,222

17

returns to the object queue loop, and which is generally indicated by the numeral 17.

FIG. 25 illustrates subroutine Q, and which receives information from subroutine DA in FIG. 24. Upon a determination that a device is not an input driver, subroutine Q is operable to receive the information, and select the appropriate data format 230 from the main module for the given device 231. Upon completion of the last step, 231, subroutine Q returns to the object queue loop and which is generally indicated by the numeral 17 in FIG. 2.

Subroutine DB and which is generally indicated by the numeral 210, receives information from the keyboard object process and which is generally indicated by the numeral 31 in FIG. 23. Subroutine DB, upon receiving information from the keyboard object process D, is operable to determine if the request is a message for a driver 232. If the request is for a driver, then, in that event, the information is directed to subroutine R and which is generally indicated by the numeral 233. Further, and if the foregoing answer is in the negative, then, in that event, subroutine DB next determines if the request is for a special format 234. If indeed it is, then, in that event, the information is directed onto subroutine S, and which is generally indicated by the numeral 235. In the alternative and if the request is not for a special format, then, subroutine DB determines if it is a request for a special output 240. If so, then, in that event, the information is directed to subroutine T and which is generally indicated by the numeral 241. However, and if the information is not for a special output 240 then, in that event, subroutine DB next determines if it is a request for stopping all communications 242. If the answer to the last query is in the affirmative, then, in that event, the information is directed to subroutine BF, and which is generally indicated by the numeral 55. Alternatively, and if the request does not stop all communications, then in that event, subroutine DB determines if it is a request for stopping communications with only one driver 243. If the request is in the negative, then in that event, subroutine DB returns to the object queue loop, and which is generally indicated by the numeral 17. Alternatively, and if it is a request stopping communications with only one driver, then, in that event, subroutine DB requests which driver communications is to be stopped, and thereafter directs the information to subroutine BG and which is generally indicated by the numeral 61 and which was discussed in detail earlier.

Subroutine R and which is generally indicated by the numeral 233 in FIG. 27 is operable to receive information from subroutine DB and which is indicated by the numeral 210 in FIG. 26. Subroutine R is operable, upon receiving information, to select the serial port or local area network as appropriate 250 and execute a request for the name of the driver (code overlay to be loaded) 251. Upon completion of step 251, subroutine R determines if the information is a driver for imparting variables 252. If indeed it is, then, in that event, subroutine R executes the request for variables 253 and thereafter builds the request in the form of a mailbox request 254. Upon completion of the step 254, subroutine R returns to subroutine BB and which is generally indicated by the numeral 43 and which was discussed in the above paragraphs.

As best illustrated by reference to FIG. 28, subroutine S 235 receives information from subroutine DB and is operable to select the format of the code overlay as

18

appropriate 260 for the message received and thereafter executes a command to build a request in the form of a mailbox request 261. Subroutine DB subsequently returns to subroutine BD 51 as indicated in FIG. 28. In addition to the foregoing, subroutine T, and which is generally indicated by the numeral 241 in FIG. 29, is operable to receive the special output message 240 from subroutine DB 210 and thereafter determines if the output is a mailbox or hardware output 262. If the output is not a mailbox or hardware output, then, in that event, the information is diverted to subroutine TA and which is indicated by the numeral 263 in FIG. 30. If, in the alternative, the output is a mailbox or hardware output, then, in that event, subroutine T is operable to request the parameters of same 264 and then return to the object queue loop, and which is generally indicated by the numeral 17 in FIG. 2.

Subroutine TA, and which is indicated by the numeral 263 in FIG. 30, is operable to receive information from subroutine T in FIG. 29 and is further operable upon receiving that information to request a driver name 270 and then determine if the driver is loaded 271. If the driver is loaded, then in that event, subroutine TA is operable to update the main module with the format requested by the external process 276 and then return to subroutine H 82. Alternatively, and if the driver is not loaded, subroutine TA is operable to determine if the driver exists, and if the answer is in the negative, then in that event it generates an error message by means of subroutine E 63. However, if the driver does exist, subroutine TA next determines if sufficient memory is available to handle the request 273, and if not, an error message is generated by means of subroutine E 63. Alternatively, and if sufficient memory is available, then, in that event, subroutine TA loads the driver as a code overlay 274, and determines if loading was successful by means of step 275. An error message is generated in the event that loading was not successful 63. If loading was successful, then, in that event, an update of the main module proceeds with the format requested by the external process 276 and thereafter subroutine TA returns to subroutine H, and which is generally indicated by the numeral 82.

As best illustrated by reference to FIG. 31, subroutine DC, and which is indicated by the numeral 212 in FIG. 23, receives information from the keyboard object process 31 and is operable, upon receiving the information, to sequentially select the serial port 280 requested, enter the base address and IRQ of the information 281, and then save the new settings 282 as appropriate. Upon completing the last step 282, subroutine DC returns to the object queue loop A, and which is generally indicated by the numeral 17 in FIG. 2.

As best illustrated by reference to FIG. 32, subroutine RO is adapted to receive information from subroutine M (FIG. 19) and then determines whether the information is an output on video 283. If the request for the information is not an output on video, then, in that event, the information is directed to subroutine Z, and which is generally indicated by the numeral 284 in FIG. 32. However, and if it is a video output request, then in that event, subroutine RO determines whether the information is a special format request 285. In the event that the information received is a special format request, then, in that case, subroutine RO executes a call appropriate for the format code overlay 290 and thereafter calls the main module routine for displaying the results 291, as appropriate. Upon achieving the last step 291,

5,161,222

19

subroutine R0 returns to the object queue loop A and which is generally indicated by the numeral 17 in FIG. 2.

Subroutine Z, and which is generally indicated by the numeral 284 in FIGS. 3A-B is operable, upon receiving information from subroutine R0, to determine if the request is for a special format 286. If the answer is in the affirmative then, in that event subroutine Z is operable to call the appropriate code overlay 287 and thereafter determine if the output is to be redirected to a mailbox 292. If this determination is in the affirmative, then, in that event, subroutine Z sends the decoded and formatted string to the external process mailbox 293, and then returns to the object queue loop 17. If, in the alternative the determination is negative, then, in that event, subroutine Z next determines whether the output is to be redirected to a hard disk 294. If the determination is affirmative, then, in that event, subroutine Z sends the decoded and formatted string to the hard disk with a requested file name 295 and thereafter returns to the object queue loop, and which is generally indicated by the numeral 17 in FIG. 2. If the information is not to be redirected to a hard disk, subroutine Z next determines if the output is to be redirected to a local area network 300. If the response to this query is in the affirmative, then, in that event, subroutine Z sends the decoded and formatted string to a local area network with the requested file name 301 and then returns to the object queue loop 17. Alternatively, and if the information is not to be redirected to a local area network then, subroutine Z determines if the output is to be redirected to a printer 302. If this last determination is in the affirmative, then, in that event, the decoded and formatted string is sent to a printer 303, and subroutine Z returns to the object queue loop 17. Alternatively, and if the determination to the previous query, is negative, then, subroutine Z determines if the output is to be redirected to a modem. If the information is to be directed to a modem then the decoded and formatted string is transmitted to the requested serial port 305. Similarly and upon transmission to the requested serial port, subroutine Z then returns to the object queue loop 17. Further, and if the information is not to be redirected to a modem, then, in that event subroutine Z next deter-

20

mines if the output is to be redirected as a driver filtered output. If the answer to the previous query is in the affirmative, then, the subroutine sends the decoded and formatted string to a code overlay for filtering the output, and then returns to the object queue loop 17.

As best illustrated by reference to FIG. 34 it should be understood that an external process, not shown, and which may include such programs as "Atlantis", "Sensor", and "Helena" are operable to be accessed by a user, not shown, and which are operable to provide instructions regarding a particular sensor to the library function, which, in turn, transmits the request for information through an operating system module which converts this same information into a format which is understood by the main module and which is set forth in greater detail in Figs. 1 through 33, respectively. The main module is operable to interpret the variables produced by a sensor, and which communicates by means serial or analog protocols and is further operable to interpret the external calling process request for data information from the sensor and thereafter overlay a predetermined adaptable driver which, when adjusted in a predetermined fashion, corresponds to the characteristics of the sensor. As earlier discussed, an adaptable driver is an executable program which is resident on the hard disk of the computer and which may be called up into memory based upon a user command and thereafter employed to provide the requested information from the sensor and which may then be used by the main module. Upon overlaying a predetermined adaptable driver the main module polls, or listens to, the sensor thereby receiving the data information requested, and thereafter transmits the information to a predetermined destination in accordance with the earlier subroutines which were discussed in greater detail in the paragraphs above. As best illustrated in the example which follows, the driver skeleton for a Nellcor N200 pulse oxymeter is provided and which illustrates the type of driver skeleton which is resident within the main module and which provides a means by which a relatively unskilled programmer may use the same driver skeleton and thereafter rapidly interface the main module with a sensor which is foreign to the present device with very minimal amounts of reprogramming.

45

/\*

## HMP MEDICAL INTERFACE BUS

## C - DRIVER SKELETON

```
Driver ID: Nellcor N-200
Version:
Date Last Worked on:
By:
Comments: This example adds to the main module a driver for
the Nellcor N200 pulse oxymeter.
```

\*/

```
int_actused = 0;
```

/\*



21

5,161,222

22

## C SKELETON EQUATES

(do not alter)

```
#include<drv_equ.h>
#include<var_equ.h>
```

/\*

## STRUCTURE DEFINITION TO DEVICE

(do not alter)

\*/

```
struct REQUEST {
    char mon_req[10]; /* request to mon (10 chars max) */
    char mon_lreq; /* real length of request */
    char mon_ans[5]; /* answer expected from mon */
    char mon_lans; /* length of answer to check */
    char mon_fl4ans; /* answer to wait for */
    char mon_filler[2]; /* filler for 20 char in structure */
}; /* length in mib_equ */
```

/\*

## COMMUNICATIONS POLLING DEFAULTS

(Enter here information)

(do not reorder)

```
char mon_name[8] = {"n200c"}; /* name of file (DOS max=8 char) */
int mon_poll_int = 200; /* tm int for poll freq in 1/100'ths sec */
int mon_delay_int = 40; /* tm delay between polling and response */
char mon_baud = BAUD_9600; /* baud rate from drv_equ.h */
char mon_parity = DBB_5B1_NOP; /* port settings from drv_equ.h */

char mon_language = 'C'; /* this driver is in C language */
char mon_flavor = 1; /* 1 = monitoring device */
char mon_pollable = 1; /* 1 = mon pollable, 2 = not pollable */
char mon_spec_comdef = 0; /* see manual for explanation */

char mon_step4hello = 1; /* see manual for explanation */
char mon_numof_steps = 2; /* see manual for explanation */
```

/\*

## STRUCTURE FOR REQUEST TO DEVICE

(Enter here information)

\*/

```
char mon_pos4req[8] = {"REQ_SOT"}; /* string header for request */
struct REQUEST mon_request[2] = {
    {"S", 1, "S", 1, 1, " "},
    {"R", 1, "R", 1, 1, " "}
```

```

23                                     5,161,222                                     24

/*

                                VARIABLES TO READ FROM DEVICE

                                (Enter here information)

                                (do not reorder)

                                                                */

char var_pos4req[8] = {"VAR_bot"}; /* string header variables/id's

int vars_numof      = 2;
int var_present[2]  = { PULSE, SAO2 };
char vars_id[2]     = {"RS"};

#include,cdrv_fn.h>                                /* c-function library for main module */

*/

                                BEGIN CODE TO DECODE STRING FROM DEVICE

                                (Enter here code...)

INPUT:
                                int function_type: 1 = decoding
                                int mon_str_len:   Length of str_2decode
                                int var_rank:      rank of var to decode eg. SAO2 = 1. (0 based)
                                char *str_2decode: answer from device

OUTPUT:
                                char result[20]: decoded string to return to main module

FORMAT: 'vvvvv'
                                variable value in 5 chars
                                trailing blank for future use

EQUATES: BUF4ANS_LEN = max length of string from device

                                                                */

char *ov0fn(int function_type,int mon_str_len,int var_rank,char *str_2decode)
{

char *tab[BUF4ANS_LEN];
char result[20];
int i,j;

/* Passing of Arguments */

tab[0] = str_2decode;
for (i = 1; i < mon_str_len && i < BUF4ANS_LEN; i++){
    tab[i] = tab[i-1] + 1;

/* DECODING CODE FOR STRING COMING FROM MON. DEVICE BEGINS HERE */
for (i = 0; i < mon_str_len && i < BUF4ANS_LEN; i++){
    if (*tab[i] == vars_id[var_rank])
        break;
}

}

```

25

5,161,222

26

```

if ( i == mon_str_len) return(buf_error);

result[0] = '0';
result[1] = '0';

for (j=0; j<3; j++)
    result[j+2] = *tab[i+1+j];

result[j+2] = '\0';

return(result);
}

```

As best understood following a study of the example provided above, the step of overlaying and updating a predetermined adjustable driver includes the steps of entering the name of the adaptable driver format in the driver skeleton as provided; providing a polling timer interval for the sensor; entering a timing delay interval for the sensor and which is operable to await the answer of the sensor for each request; providing a baud rate and parity for the sensor being polled; providing the name of the language which is employed by the driver; entering the type of sensor device being polled; determining whether the sensor can be polled or listened to and providing instructions as appropriate; providing special communications instructions to the sensor, if necessary; entering the number of polling requests which must be completed to insure receipt of all data information from the sensor; providing the external process requests to be made to the sensor for each of the steps; providing the variables which are to be read from the sensor; decoding the variables received from the sensor, and delivering same to a predetermined destination. As should be understood, the step of providing the variables which are to be read from the sensor includes providing the number of variables which are to be read from the sensor, and identifying the variables which are to be read from the sensor. In the present case, the pulse and SA02 variables are being received from the present device which is shown in the example.

It should be understood that the inventors have discovered that with respect to medical monitoring devices of the type anticipated to be used with the present invention that, as a general matter, the general format or skeleton of the data information has a basic structure which is quite similar to the skeleton noted in the example above. Therefore, a programmer utilizing the skeleton can very quickly, using only a few lines of code, successfully interface a foreign sensor with the main module. In the present example the inventors have provided the driver skeleton for a Nellcor N200 pulse oxymeter which provides SA02 and pulse to a clinician for diagnostic purposes.

Referring to the example noted above, and more specifically to that portion of the example which is entitled "Communication Polling Defaults" it should be understood that the skeleton is written in C language, although those skilled in the art will readily recognize that assembly language could be used with equal success. In this regard the skeleton of the above-identified Nellcor N200 pulse oxymeter has a means for providing a request to the sensor and which is indicated by the first line of commands in that portion of the skeleton labelled "Structure Definition to Device". Further, the skeleton provides a location where the name of the sensor, and in the present example the designation N200C is entered. The next command provides a means

whereby a monitor polling interval may be established. As should be understood, and depending upon the type of sensor involved, a monitor may be available for polling at predetermined intervals as a result of their own inherent engineering characteristics. In this regard, and based upon the characteristics of the sensor, an operator may enter the polling interval frequency for the monitor in question. Further, the next command provides a means for setting a timing delay between the polling request and the response from the monitor. In this regard it should be understood that, in certain instances, and based in large measure on the characteristics of the monitor, or sensor being polled, the monitor may not provide the data instantaneously and thus there may be a time delay. In this instance the time delay would be entered at this location in the skeleton. Further the next two lines of commands includes standard communication settings for both the baud rate of the sensor being polled, as well as the parity for the sensor. These communications parameters are well understood by those skilled in the art. The driver skeleton in the example above provides a means whereby the language of the driver which is providing the communications data may be entered such that the main module can understand how it will call the code overlay. In this regard, and in the example provided, the sensor (Nellcor N200) produces data in C language, and this information is entered at this location in the skeleton.

The driver skeleton provides a means whereby an operator may enter information regarding whether the present sensor under consideration is a monitoring device; a local area network; or further, whether the driver produces information either in a serial or analog format. By entering the appropriate characters as provided in this portion of the skeleton, the operator may adjust the skeleton for those features of the sensor. The next line of commands in the skeleton provide a means by which the operator can enter special communications definitions, and more particularly those communications definitions related to whether the sensor being polled is pollable or non-pollable. As should be understood some sensors produce information at periodic time intervals and cannot be polled by an external calling process. By inserting the appropriate character at this portion of the skeleton the operator may adjust the skeleton as appropriate for these communications parameters. The next line of command relates to special communications definition, and more particularly to the initialization of the sensor being polled. In this regard, it should be understood, that some sensors, based upon their inherent design characteristics, may require specific instructions prior to providing the data information requested. The skeleton therefore provides an operator with a means for adjusting same for those sensors having these characteristics. In addition, the next com-

27

5,161,222

mand in the skeleton provides a means by which an operator may compensate for a multiplicity of steps which may be required in order to cause the same sensor to provide the data information requested. As should be understood, some sensors, based upon their inherent design characteristics, may need to exchange several bits of data and/or provide or receive various instructions from an external calling process prior to transmitting the requested information. In this regard, an appropriate character may be entered at this point in the skeleton to compensate for those shortcomings in the sensor. Finally the last line of instructions provides a means by which the operator may set the number of steps required in order to receive enough information for the handshake. As should be understood certain sensors may be required to be polled a number of times in order to receive all the information requested. In this regard the Nellcor N-200 must receive two requests for information prior to divulging all the information that is, pulse, and SA02, but, as a general matter one request is usually sufficient to complete the handshake.

Referring more particularly to that portion of the skeleton labelled "Structure for request to device", it should be understood that means are provided for requesting data from the sensor. One skilled in the art will recognize the use of the structure described in that portion of the skeleton labelled "Structure request to device". This includes a set of information for requesting data from a sensor. In the present example, the skeleton includes two requests. The first set of information requests the string to send to the sensor. In this regard and in the skeleton of the Nellcor shown above, the string is a "S". The next set of information is its length. In the present example it is a "1". The remainder of the characters are for validating the answer, followed by their number. In the above example, it is an "S", and one character. The "1" following the "S" specifies that the main module will expect an answer for that request.

Referring more particularly to that portion of the example provided above, and which is directed to the variables to read from the device portion, it should be understood that means are provided for reading the variables received from the sensors, and wherein the operator may designate the number of variables to be read. Further the operator may also designate the variables by type. In this instance, the example provides that the variables being read are pulse, and SA02, and which are set forth in the text above. The character identification are designated as R and S, respectively. Finally the example above provides an example of decoding the variables received from the sensor. As should be readily apparent upon a close study of the example, an operator may only be required to enter as little as ten lines of programming in order to complete a foreign sensor driver as appropriate for the main module.

Therefore the method for interpreting variables produced by a sensor and which communicates by means of serial or analog protocol can be employed in a wide variety of operative environments, and can be manufactured and purchased at moderate cost when compared with related prior art software devices, is different sensors having different communications parameters and which further is highly efficient in operation and reduces to an absolute minimum the problems associated with the assimilation of clinical information from diverse sources and which are designed for substantially identical purposes.

Although the invention has been herein shown and

28

described in what is conceivably the most practical and preferred embodiment, it is recognized that departures may be made therefrom within the scope of the invention which is not to be limited to the illustrative details disclosed.

Having described our invention, what we claim is new and desire to secure by Letters Patent is:

1. A software engine for interpreting variables produced by a plurality of sensors which communicate by means of serial, analog or local area network protocols, the software engine comprising:

means for interpreting an external process request for data information which includes the variable produced by the sensor, and wherein the external process request is an independently executable software program which includes a piece of software code which generates a message in a predetermined format requesting from the software engine the delivery of data which is in a specific format and syntax, and wherein the independently executable software program in otherwise unfamiliar with the software engine and which further includes:

a. means for interfacing with an operating system to determine if an appropriate operating system is present and to execute the necessary memory and disk management functions such that the software engine can operate, the software engine otherwise being operating system independent, and wherein the interface means is included in a main module;

b. means for creating a mailbox which is adapted to exchange information with the external process request, and

c. means for opening an object queue loop, and wherein the object queue loop identifies an object type and initiates a predetermined process with the individual sensors requested by the external process;

means for overlaying a predetermined adaptable driver, which, when adjusted in a predetermined fashion, corresponds to the data characteristics of the individual sensors and wherein the adaptable driver decodes the variables requested by the external process request;

means for polling or listening to the individual sensors, thereby receiving the data information requested by the external process request, and wherein the data information received is in a predefined format, and in any syntax; and

means for transmitting the data information generated by the individual sensors to a predetermined destination.

2. A software engine as claimed in claim 1 and wherein the object type includes mailbox, timer, and keyboard objects, and wherein the software engine includes predetermined processes which correspond with the mailbox, timer, and keyboard objects.

3. A software engine as claimed in claim 2 and wherein the mailbox process includes,

means for initiating communications with the external process;

means for determining the characteristics of the driver employed by the sensor;

means for overlaying and updating the predetermined adaptable driver format to substantially correspond to the characteristics of the sensor;

means for polling or listening to the sensor thereby

29

5,161,222

30

receiving the data information requested;  
 means for decoding the variable requested by the external process;  
 means for transmitting the decoded information to a predetermined destination directed by the external process; and  
 means for returning to the object queue loop.

4. The software engine as claimed in claim 2 and wherein the timer process includes a polling timer process and a delay timer process.

5. A software engine as claimed in claim 4 and wherein the polling timer process includes,  
 means for starting the polling timer;  
 means for determining the characteristics of the driver associated with the polling request;  
 means for executing the first polling request received from the external process;  
 means for actuating a delay timer;  
 means for transmitting the information received from the first polling request to a predetermined destination which corresponds to the external process request upon deactivation of the delay timer; and  
 means for returning to the object queue loop.

6. A software engine as claimed in claim 4 and wherein the delay timer process includes,  
 means for determining if conditions for polling the sensor have been met;  
 means for continuing communication with the sensor;  
 means for transmitting the information received from the sensor to a predetermined destination; and  
 means for returning to the object queue loop.

7. A software engine as claimed in claim 2 and wherein the keyboard process includes a status process, and a keyboard interaction process, and wherein the status process includes,  
 means for determining the type of input or driver;  
 means for selecting an appropriate data format which corresponds with the type of input or driver;  
 means for displaying the given data format for the given input or driver; and  
 means for returning to the object queue loop.

8. A software engine as claimed in claim 7 and wherein the keyboard interaction process includes,  
 means for determining the type of request from a user as an external process;  
 means for identifying the type of driver which corresponds to the user request;  
 means for loading a driver code overlay which corresponds with the type of driver identified;  
 means for requesting a direction and format for the output; and  
 means for returning to the object queue loop.

9. A method as claimed in claim 2 and wherein the means for overlaying the predetermined adaptable driver includes,  
 means for loading the adaptable driver;  
 means for ascertaining the communication parameters of the sensor; and  
 means for modifying the predetermined adaptable driver with the variables which substantially correspond with the variables requested by the external process and which correspond with the sensor being polled.

10. A software engine as claimed in claim 2 and wherein the means for overlaying the predetermined adaptable driver includes,  
 means for entering the name of the adaptable driver format;

means for providing a polling timer interval for the sensor;  
 means for entering a timing delay interval for the polling timer and which is operable to delay the polling requests to the sensor;  
 means for providing a baud rate and parity for the sensor being polled;  
 means for providing the name of the language which is employed by the sensor;  
 means for entering the type of sensor device being polled;  
 means for determining whether the sensor can be polled, or listened to, and providing instructions as appropriate;  
 means for providing special communication instructions to the sensor, if necessary;  
 means for entering the number of polling requests which must be completed to ensure receipt of all the data information from the sensor;  
 means for providing the variables which are to be read from the sensor; and  
 means for decoding the variables received from the sensor and delivering same to a predetermined destination.

11. A software engine as claimed in claim 10 and wherein the means for providing special communications instructions includes,  
 means for initialization of the sensor being polled;  
 means for establishing communication with the sensor being polled; and  
 means for polling the sensor a predetermined number of times to receive all the data information produced by same.

12. A software engine as claimed in claim 11 and wherein the means for providing the variables which are to be read from the sensor include,  
 means for providing the number of variables which are to be read from the sensor; and  
 means for identifying the variables which are to be read from the sensor.

13. A software engine for interpreting data information which includes variables produced by a plurality of sensors which communicate by means of serial, analog or local area network protocols, and wherein the software engine operates at run time, in combination with an operating system, to execute the necessary memory and/or disk management functions such that the software engine can operate, the software engine comprising:  
 means for interpreting an external process request for data information from the sensors and wherein the external process request is an independently executable software program which runs simultaneously and independently with respect to the software engine, and which further has an operating system interface module which executes the necessary memory and disk management functions such that the software engine can operate, the software engine otherwise being operating system independent, and wherein the independently executable software program includes a piece of software code which generates a message in a predetermined format requesting from the software engine the delivery of data which is in a predetermined format and syntax, and wherein the interpreting means includes a module disposed in data receiving relation relative to the external process request, and which is operable to initiate an object queue

31

5,161,222

loop which identifies different object types including a mailbox through which data information is exchanged with the external process, and a timer which regulates the frequency of data exchanges with the individual sensors requested by the external process, and wherein the module further processes the data exchanges with the individual sensors;

means for overlaying a predetermined adaptable driver which, when adjusted in a predetermined fashion by employing specific functions of the operating system, corresponds to the data characteristics of the individual sensors, and wherein the predetermined adaptable driver to be overlayed includes a driver skeleton which has a decoding area, and wherein an operator, prior to overlaying the predetermined adaptable driver, supplies data to the driver skeleton which includes,

- a name for the adaptable driver;
- a polling timer interval for the individual sensors;
- a timing delay interval which is provided to the module and which determines the time interval with which the module must return to the individual sensors to receive all the data requested by the external process;
- a baud rate, and parity, for the individual sensors being polled by way of serial communications;
- a name of the language which is employed by the individual sensors;
- the type of sensor device being polled or listened to, including serial, analog, or local area networks;
- instructions regarding whether the individual sensors can be polled, or listened to;
- any special communication instructions to the individual sensors;
- the number of polling requests which must be completed to ensure receipt of all the data information from the individual sensors;
- the variables which are to be read from the individual sensors; and
- the decoding area of the driver skeleton which decodes the variables read from the individual sensors;

means for polling or listening to the individual sensors thereby receiving the data information requested by the external process request, and wherein the data information received is in a predetermined protocol, and in any syntax, and wherein the driver decoding area decodes the variables requested by the external process request; and

means for routing the variables generated by the individual sensors, and which have been decoded by the driver decoding area, to a predetermined destination, or to the requesting external process, as appropriate.

14. A software engine which operates in combination with a computer having a disk operating system, and wherein the software engine is operable to interpret variables produced from a plurality of remote sensors which communicates by serial, analog or local area network protocols, and wherein an external process requests data information from selected sensors and the software engine directs the requested information to a predetermined destination, the software engine comprising:

- a library function which can be accessed from the external process, and wherein the library function

32

executes a plurality of commands which can be requested by an operator, and wherein an operator issues commands through the library function regarding the particular remote sensor from which data information is requested;

- a main module including an operating system interface module, and wherein the operating system interface module is operating system dependent, and disposed in data receiving relation relative to the library function, and wherein the main module includes a memory which stores a plurality of drivers which are operable to interpret the external process request for information, and wherein the library function transmits the external process request for information and converts this external process request into a format which is understood by the main module, and wherein the main module is operable upon the request of the external process, to determine whether an appropriate operating system is present to execute the necessary memory and disk management functions such that the software engine can operate but which is otherwise operating system independent, create a mailbox through which data information is exchanged with the external process, and initiate an object queue loop which identifies predetermined object types, and wherein the main module initiates a process unique to each of the object types, and the particular remote sensor requested by the external process;

means for overlaying a predetermined adaptable driver which, when adjusted in a predetermined fashion by employing specific functions of the operating system, corresponds to the characteristics of the particular sensor, and wherein the adaptable driver called up from the memory of the main module and overlayed includes a driver skeleton which has a decoding area, and wherein the driver skeleton may be adjusted by an operator, and the decoding area employed to decode the variables produced by the particular remote sensor, the operator supplying information to the driver skeleton which includes,

- a name for the adaptable driver;
- a polling timer interval for the particular sensor;
- a timing delay interval which is provided to the main module and which determines the time interval with which the main module must return to the particular sensor to receive all the data requested by the external process;
- a baud rate, and parity, for the particular sensor being polled by way of serial communications;
- a name of the language which is employed by the operator in the decoding area of the particular sensor;
- the type of sensor being polled or listened to;
- instructions whether the particular sensor can be polled or listened to, as appropriate;
- any special communication instructions to the particular sensor, as appropriate;
- the number of polling requests which must be completed to ensure receipt of all data information from the particular sensor;
- the variables which are to be read from the particular sensor; and
- the decoding area of the driver skeleton and which decodes the variables read from the particular sensor;

5,161,222

33

means for polling or listening to the particular sensor  
thereby receiving the variables which have been  
decoded by the decoding area of the updated driver  
skeleton; and

34

means for routing the decoded variables to a prede-  
termined destination, or to the requesting external  
process, as appropriate.

\* \* \* \* \*

5

10

15

20

25

30

35

40

45

50

55

60

65

UNITED STATES PATENT AND TRADEMARK OFFICE  
CERTIFICATE OF CORRECTION

PATENT NO. : 5,161,222

DATED : November 3, 1992

INVENTOR(S) : Montejo et al.

It is certified that error appears in the above—identified patent and that said Letters Patent is hereby corrected as shown below:

Column 13, line 47, after the numeral 133,  
insert --.--;

Column 15, line 5, after the numeral 176,  
insert --.--;

Column 19, line 35, after the numeral 17,  
insert --.--;

Column 27, line 61, after the words "devices, is"  
insert --highly efficient in operation and is operable  
to facilitate the rapid assimilation of sensor  
information from a variety of--;

Column 28, line 13, delete the word --variable--  
and insrt --variables--.

Signed and Sealed this

Twenty-eighth Day of September, 1993

Attest:



BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks





## Medical Device Connectivity

Search: [SOLUTIONS](#)[DATACAPTOR](#)[NEWS](#)[COMPANY](#)[RESOURCES](#)[USERS' AREA](#)**Who We Are**

Capsule Technologie is the world's leading provider of medical device connectivity solutions. The company is composed of software and hardware specialists with extensive experience in low-level transport protocols, device firmware and advanced software programming. Our technical team focuses exclusively on the challenge of integrating data from diverse medical devices—from manufacturers around the world—with clinical and hospital information systems.

**What We Provide**

Capsule Technologie's flagship product, DataCaptor™, provides an interface between standalone and/or networked medical devices and clinical/hospital information systems. DataCaptor collects, decodes and distributes all the medical data that is made available at the digital communication port of the medical device. This generic software solution is available to clinical software vendors, system integrators and device manufacturers, as well as to hospitals and other end-users. With built-in support for more than 250 diverse medical devices, DataCaptor includes the largest library of medical device interfaces available. DataCaptor's distributed architecture makes it a highly scalable and flexible way to interface all network and/or serial devices. Capsule Technologie is ISO9001 certified and DataCaptor is FDA 510(k) cleared.

**Why We Do It**

For application providers, system integrators and device manufacturers alike, creating interfaces for medical device connectivity is a challenging and resource-hungry activity. Effectively, the best way to attain reliability is for all industry players—application providers and device manufacturers alike—to share a common, generic platform. Capsule Technologie takes advantage of recent advancements in programming technologies enabling the development of middleware that is flexible and scalable enough to become a generic medical device interface solution.

More and more, the world's leading application providers and device manufacturers are choosing DataCaptor, our extensive library of DataCaptor Device Interfaces™ (DDIs), as their connectivity solution. DataCaptor makes

possible to benefit from *increased availability, reliability and cost-effectiveness* of device connectivity while le  
the digital value of medical devices for improved patient care.

Copyright © 2004



**NEW version 4.4**

## The Universal Connectivity Solution for Medical Devices

### BUILT-IN SUPPORT FOR DEVICES FROM THESE AND OTHER LEADING MANUFACTURERS:

*Over 250 different bedside  
medical devices supported.*

Abbott Diagnostics  
Alaris Medical Systems  
Aspect Medical Systems  
Bard  
Baxter  
B. Braun  
Bios Medical  
Beckman Coulter  
B. Braun  
Cobe Cardiovascular  
Cohn  
DataScope  
Durex, Ohmco  
Dräger  
Ereschius Vial  
GE Medical Systems  
Gravety  
Johnson & Johnson  
Jorha  
Kontron  
Mallinckrodt  
NovoMatrix  
Novartis  
Oxford Instruments  
Philips Medical Systems  
Radiometer  
Roche Diagnostics  
Siemens  
Spacelabs  
Tasma  
Welch Allyn

**DataCaptor allows you to connect any medical device to your clinical application and collect data in real time.**

With built-in support for more than 250 diverse bedside devices, DataCaptor includes the largest library of medical device interfaces available anywhere. New DataCaptor Device Interfaces™ (DDIs) are continuously added to support new versions, models and manufacturers around the world. DataCaptor makes it possible to benefit from increased availability, reliability and cost-effectiveness of device connectivity while leveraging the digital value of medical devices for improved patient care. Capsule Technologie is ISO9001 certified and DataCaptor is US FDA 510(k) cleared and CE certified.

More and more, leading application providers and device manufacturers are choosing DataCaptor. DataCaptor is the universal connectivity solution for:

#### ► Clinical Information System providers

Integrate DataCaptor into your applications to collect data from diverse medical devices. DataCaptor can also be used to interface applications to the ADT or lab systems.

#### ► System Integrators

Include DataCaptor in your toolset to deliver integration solutions and make full, universal device connectivity simpler and more cost effective for you and your clients.

#### ► Medical Device Manufacturers

Bundle DataCaptor with your products in order to provide data in common and industry-standard output formats including HL7 and XML.

#### ► Hospitals & Healthcare providers

Install DataCaptor and you can seamlessly collect data from virtually any medical device and transfer it to your information systems, Excel worksheets or SQL databases.



ISO 9001 Certified  
AOQC Moody International  
No. 200005190



US Food & Drug Administration  
FDA 510(k) cleared



In North America call toll free 1 (800) 130-0837 • In Europe call +33 (0) 1 3324 1430 • [info@capsuletech.com](mailto:info@capsuletech.com)  
[www.capsuletech.com](http://www.capsuletech.com)

Copyright © 2004 Capsule Technologie



## The smart solution to medical device connectivity for clinical application providers and system integrators

### ► Complete Connectivity

The ever-increasing number and diversity of medical devices on the global market makes it difficult to provide data collection interfaces for every model. Capsule Technologie offers a comprehensive, universal solution. Check [www.capsuletech.com](http://www.capsuletech.com) for a complete list of supported devices—either the device interface you need is available or Capsule Technologie will develop it.

### ► Tested and Up-to-date Device Interfaces

For application providers and system integrators, creating interfaces for medical device connectivity is a challenging and resource-hungry activity. Manufacturers are continuously upgrading medical devices with new functions, making drivers obsolete. All of Capsule Technologie's interfaces are rigorously tested before release and are updated when device manufacturers upgrade device protocols.

### ► Future-proof Connectivity

New technologies will result in continuous and dramatic changes to customers' connectivity needs. DataCaptor is based on an open and distributed architecture, is easily scalable, and compatible with most networks and terminal servers on the market. DataCaptor can interface all types of devices that send data, including wireless devices.

### ► Certified Solutions

As regulations move towards certification requirements for all software in the critical care area, most CIS providers will need to redo their entire driver libraries. Capsule Technologie, an international medical connectivity provider, is up-to-date on local regulation issues. DataCaptor has FDA 510(k) clearance, CE certification and ISO 9001 certification.

### ► Automatic Device Identification

Simply collecting data from devices is not enough; data needs to be clearly assigned to its source. DataCaptor includes Automatic Device Identification enabling Plug-and-Play connectivity to medical devices without software reconfiguration. Ideal for exchanging devices at the bedside.

### ► Generic Connectivity

Using DataCaptor provides customers with a single connectivity platform interfacing devices and systems, such as ADT or lab systems, simultaneously.



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

**DataCaptor™**

Connect any medical device to your clinical application.

DataCaptor allows you to collect data from an extensive library of bed medical devices in real time, and access the data in your clinical application.

**new version 4.4**

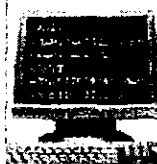
with Automatic Device Identification

→ Learn More

“ ”

Read how Philips, Siemens, Epic and others have used DataCaptor to solve their medical device connectivity needs. → Read More

Attend a live online demonstration and learn about the features and functions of the most advanced solution for medical device connectivity.



→ Sign Up



Automatic Device Identification  
DataCaptor now features connectivity. → Learn More

**ECLIPSYS**

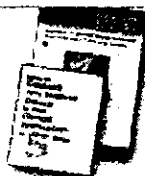
The Outcomes Company

Eclipsys selects DataCaptor as the Universal Device Connectivity engine for SunriseX™.

→ Press Release

The Device Connectivity Information You Need.

Get this report, with case studies and testimonials, as well as a step-by-step guide to collecting device data. → Free Upon Request



Join our mailing list for DataCaptor updates and valuable information about Device Connectivity issues.

→ Sign Up Now



No. 200005190  
ISO 9001 Certified  
AOQC Moody International



FDA 510(k) cleared



Copyright © 2004 Capsule Technologie



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

**Innovative device connectivity solutions for the healthcare industry**

Capsule Technologie is the World Leader in Medical Device Connectivity and offers the largest library of device interfaces on the market. We specialize in device protocols, data transfer, and system integration.

Whether you are a Clinical Information System Provider, a System Integrator, a Medical Device Manufacturer or a Healthcare provider, there is a solution for you.

**Clinical Information System Providers**

Integrate DataCaptor into your CIS and seamlessly access the largest library of device interfaces available on the market.

[➔ Learn more](#)**System Integrators**

Offer your customers complete connectivity to all new and legacy devices. Add DataCaptor to your toolbox and provide your customers with automated charting of vital signs from medical devices, allowing you to complete the Electronic Patient Record, increase nurse productivity and reduce human error.

[➔ Learn more](#)**Medical Device Manufacturers**

Provide data in industry standard formats, not just via a proprietary protocol. Add your devices to the DataCaptor Device Interface library and bundle a DataCaptor CD-ROM with all your devices to provide your customers with data in XML or HL7.

[➔ Learn more](#)**Healthcare Providers**

Use DataCaptor to facilitate your research projects by collecting data from a wide variety of devices and sending it to an Excel worksheet, SQL database, or leverage your Clinical Information System by extending its connectivity to all devices present in the unit.

[➔ Learn more](#)**Clinical Trial Centers**

Collect data in real time from all your monitors and build a complete and automatic Electronic Patient Record system.

[➔ Learn more](#)

Copyright © 2004



## Supported Devices

### Medical Devices Supported by DataCaptor

Updated April 27, 2004

**DataCaptor includes built-in support for more than 250 diverse medical devices from every major manufacturer. New DataCaptor Device Interfaces™ (DDIs) are continuously added to support new versions, models and manufacturers.**

Check the Capsule Technology Web site (<http://www.capsuletech.com>) for the most up-to-date list of DDIs and supported devices.

MANUFACTURER	DEVICE	DEVICE TYPE
<b>Abbott Diagnostics</b>	Oximetrix 3	Blood Gas Monitor, Oxygen
	Plum XL	Infusion Pump
	Plum XL3	Infusion Pump, Multichannel
<b>Agilent (see Philips)</b>		
<b>Alaris Medical Systems</b>	Asena CC	Infusion Pump, Syringe
	Asena GH	Infusion Pump, Syringe
	Asena GS	Infusion Pump, Syringe
	Asena TTVA	Infusion Pump, Syringe
	Asena GW	Infusion Pump, Volumetric
	IMED Gemini PC-1	Infusion Pump
	IMED Gemini PC-2	Infusion Pump, Multichannel
	IMED Gemini PC-2TX	Infusion Pump, Multichannel
	IMED Gemini PC-4	Infusion Pump, Multichannel
	IVAC TTVA	Infusion Pump, Syringe
	IVAC TTVA/TCI	Infusion Pump, Syringe
	IVAC P1000	Infusion Pump, Syringe
	IVAC P2000	Infusion Pump, Syringe
	IVAC P3000	Infusion Pump, Syringe
	IVAC P4000	Infusion Pump, Syringe
	IVAC P6000	Infusion Pump, Syringe
	IVAC P7000	Infusion Pump, Syringe
	IVAC Signature Edition GOLD 7000	Infusion Pump
	IVAC Signature Edition GOLD 7100	Infusion Pump
	IVAC Signature Edition GOLD 7101	Infusion Pump
	IVAC Signature Edition GOLD 7130	Infusion Pump
	IVAC Signature Edition GOLD 7131	Infusion Pump
	IVAC Signature Edition GOLD 7200	Infusion Pump, Multichannel
	IVAC Signature Edition GOLD 7201	Infusion Pump, Multichannel
	IVAC Signature Edition GOLD 7230	Infusion Pump, Multichannel
	IVAC Signature Edition GOLD 7231	Infusion Pump, Multichannel
<b>Aspect Medical Systems</b>	A-2000 BIS	Bispectral Index Monitor
	A-2000 XP BIS	Bispectral Index Monitor
<b>Bard</b>	CritiCore	Unimeter
<b>Baxter</b>	Vigilance	Cardiac Monitoring System, Non-invasive
<b>B.Braun</b>	Fluid Manager/fm computer	Infusion Pump
<b>Bear (see Viasys Healthcare)</b>		
<b>Beckman Coulter</b>	STKS	Analyzer, Laboratory, Hematology

Updated April 27, 2004

MANUFACTURER	DEVICE	DEVICE TYPE
bioMérieux	Vitek 120	Analyzer, Laboratory, Microbiology
	Vitek 240	Analyzer, Laboratory, Microbiology
	Vitek 30	Analyzer, Laboratory, Microbiology
	Vitek 32	Analyzer, Laboratory, Microbiology
	Vitek 60	Analyzer, Laboratory, Microbiology
	Vitek Junior	Analyzer, Laboratory, Microbiology
	Vitek 2	Analyzer, Laboratory, Microbiology
	Vital	Analyzer, Laboratory, Hematology
Bird (see Viasys Healthcare)		
Cobe Cardiovascular	Stockert STII	Heart-Lung Bypass Unit
	Stockert Compact	Heart-Lung Bypass Unit
Colin	BP308 Mark I	Blood Pressure Monitor, Non-Invasive
	BP308 Mark II	Blood Pressure Monitor, Non-Invasive
	BP308 Mark III	Blood Pressure Monitor, Non-Invasive
	BP408 Mark I	Blood Pressure Monitor, Non-Invasive
	BP408 Mark II	Blood Pressure Monitor, Non-Invasive
	BP408 Mark III	Blood Pressure Monitor, Non-Invasive
Critikon (see GE Medical)		
Datascope	Expert	Physiological Monitoring System, Acute Care
	Passport 2	Physiological Monitoring System, Acute Care
	Passport XG	Physiological Monitoring System, Acute Care
	Accutorr Plus	Physiological Monitoring System, Acute Care
	System 95	Circulatory Assist Unit, Intra-Aortic Balloon
	System 96	Circulatory Assist Unit, Intra-Aortic Balloon
	System 97	Circulatory Assist Unit, Intra-Aortic Balloon
	System 97e	Circulatory Assist Unit, Intra-Aortic Balloon
	System 98	Circulatory Assist Unit, Intra-Aortic Balloon
	System 98XT	Circulatory Assist Unit, Intra-Aortic Balloon
Datex-Ohmeda	7100	Ventilator, Anesthesia
	7900	Ventilator, Anesthesia
	Modulus CD	Ventilator, Anesthesia
	Aestiva	Ventilator, Anesthesia
	AS/3	Physiological Monitoring System, Acute Care, Anesthesia
	Capnomac	Ventilation Monitor
	Capnomac Ultima	Ventilation Monitor
	Cardiicap II	Airway Gas Monitor
	Cardiicap/5	Physiological Monitoring System, Acute Care, Anesthesia
	CS/3	Physiological Monitoring System, Acute Care
	S/5	Physiological Monitoring System, Acute Care
	S/5 Light	Physiological Monitoring System, Acute Care
	OSP-200 SATLITE Trans	Oximeter, Pulse
	Babylog 8000	Ventilator, Neonatal
	Babylog 8000 SC	Ventilator, Neonatal
Dräger	Babytherm 8004	Incubator, Infant
	Babytherm 8010	Incubator, Infant
	Capnosat	Oximeter, Pulse
	Cato	Ventilator, Anesthesia
	Caleo	Incubator, Infant
	Cicero B	Ventilator, Anesthesia
	Cicero C	Ventilator, Anesthesia, Physiological Monitoring System
	Cicero EM	Ventilator, Anesthesia
	Evita	Ventilator, Intensive Care
	Evita 2	Ventilator, Intensive Care
	Evita 2 dura	Ventilator, Intensive Care
	Evita 4	Ventilator, Intensive Care
	Evita XL	Ventilator, Intensive Care
	Fabius GS	Ventilator, Anesthesia
	Fabius Tiro	Ventilator, Anesthesia



Copyright © 2010 Capsule Technologies. [www.capsuletech.com](http://www.capsuletech.com)

Updated April 27, 2004

MANUFACTURER	DEVICE	DEVICE TYPE
Graseby	3400	Infusion Pump, Syringe
	3500	Infusion Pump, Syringe
Hamilton Medical	Amadeus	Ventilator, Intensive Care
	Veolar	Ventilator, Intensive Care
	Galileo	Ventilator, Intensive Care
	Raphael	Ventilator, Intensive Care
Hellige (see GE Medical)		
Hewlett Packard (see Philips)		
Hospal	Prisma	Hemofiltration Unit
Infrasonics (see Tyco Healthcare)		
Johnson & Johnson	Vitros 250	Analyzer, Laboratory, Clinical Chemistry
	Vitros 500	Analyzer, Laboratory, Clinical Chemistry
	Vitros 700	Analyzer, Laboratory, Clinical Chemistry
	Vitros 700c	Analyzer, Laboratory, Clinical Chemistry
	Vitros 950	Analyzer, Laboratory, Clinical Chemistry
	Vitros EC	Analyzer, Laboratory, Immunoassay
Jostra	HL-20	Heart-Lung Bypass Unit
Kontron	ABT 4300	Ventilator, Anesthesia
	ABT 5300	Ventilator, Anesthesia
	Kolomon	Physiological Monitoring System, Acute Care
	Micromon	Physiological Monitoring System, Acute Care
	Minimon	Physiological Monitoring System, Acute Care
	Supermon	Physiological Monitoring System, Acute Care
	Trakmon	Physiological Monitoring System, Acute Care
Mallinckrodt (see Tyco Healthcare)		
Marquette (see GE Medical)		
Medelec (see Oxford Instruments)		
Nelcor Puritan Bennett (see Tyco Healthcare)		
Novamatrix	Nico	Physiological Monitoring System, Cardiac, Non-invasive
Novartis	Compat-Handy	Infusion Pump, Syringe
Ortho-Clinical Diagnostics (see Johnson & Johnson)		
Oxford Instruments	Neurostar MS92B	Electromyograph
	Sonicaid FM6	Cardiotocograph
	Sonicaid FM7	Cardiotocograph
	Sonicaid Menden	Cardiotocograph
	Sonicaid Team	Cardiotocograph
	CMS (Merlin)	Physiological Monitoring System, Acute Care
Philips Medical Systems	IntelliVue	Physiological Monitoring System, Acute Care
	Viridia M3	Physiological Monitoring System, Acute Care, Portable
	Viridia M4	Physiological Monitoring System, Acute Care, Portable
	Viridia 24	Physiological Monitoring System, Acute Care
	Viridia 26	Physiological Monitoring System, Acute Care
	Series 50 A (M1351A)	Cardiotocograph
	Series 50 JP (M1353A)	Cardiotocograph
	Series 50 IX (M1350A)	Cardiotocograph
	Series 50 XM (M1350B)	Cardiotocograph
	Series 50 XMD (M1350C)	Cardiotocograph
	Philips Clinical Data Server	Physiological Monitoring System, Gateway
	Alligent Connect	Physiological Monitoring System, Gateway
Protocol Systems (see Welch Allyn)		
Radiometer	ABL 5	Analyzer, Laboratory, Blood Gas
	ABL 50	Analyzer, Laboratory, Blood Gas
	ABL 500	Analyzer, Laboratory, Blood Gas
	ABL 505	Analyzer, Laboratory, Blood Gas
	ABL 510	Analyzer, Laboratory, Blood Gas
	ABL 520	Analyzer, Laboratory, Blood Gas
	ABL 555	Analyzer, Laboratory, Blood Gas
	ABL System 600	Analyzer, Laboratory, Blood Gas

Updated April 27, 2004

MANUFACTURER	DEVICE	DEVICE TYPE
Radiometer, cont'd	ABL System 610	Analyzer, Laboratory, Blood Gas
	ABL System 620	Analyzer, Laboratory, Blood Gas
	ABL 725	Analyzer, Laboratory, Blood Gas
	ABL 735	Analyzer, Laboratory, Blood Gas
	BPH 5	Analyzer, Laboratory, Blood Gas
	EML 100	Analyzer, Laboratory, Blood Gas
Roche Diagnostics	Hitachi 917	Analyzer, Laboratory, Immunoassay
	Elecsys 2010	Analyzer, Laboratory, Immunoassay
	Elecsys 1010	Analyzer, Laboratory, Immunoassay
Siemens	SC5000	Physiological Monitoring System, Acute Care
	SC6000	Physiological Monitoring System, Acute Care
	SC6002	Physiological Monitoring System, Acute Care
	SC6002XL	Physiological Monitoring System, Acute Care
	SC7000	Physiological Monitoring System, Acute Care
	SC8000	Physiological Monitoring System, Acute Care
	SC9000	Physiological Monitoring System, Acute Care
	SC9000XL	Physiological Monitoring System, Acute Care
	Servo 300	Ventilator, Intensive Care
	Servo 900 C/D	Ventilator, Intensive Care
	Servo <sup>i</sup>	Ventilator, Intensive Care
	Sirecust 960	Physiological Monitoring System, Acute Care
	Sirecust 961	Physiological Monitoring System, Acute Care
	Sirecust 1280	Physiological Monitoring System, Acute Care
	Sirecust 1281	Physiological Monitoring System, Acute Care
	Sigma 8000	Infusion Pump
Sigma International Spacelabs	PC1	Physiological Monitoring System, Acute Care
	PC2	Physiological Monitoring System, Acute Care
	Ultraview 1700	Physiological Monitoring System, Acute Care
	Ultraview 1500	Physiological Monitoring System, Acute Care
	Ultraview 1050	Physiological Monitoring System, Acute Care
	Universal Clinical Workstation	Physiological Monitoring System, Acute Care
	PCMS Gateway (Spacegate)	Physiological Monitoring System, Gateway
	Ultraview Gateway	Physiological Monitoring System, Gateway
Stockert (see Cobe Cardiovascular)		
Taema	Alys	Ventilator, Anesthesia
	Horus	Ventilator, Intensive Care
Tyco Healthcare	N-180	Oximeter, Pulse
	NPB 190	Oximeter, Pulse
	NPB 195	Oximeter, Pulse
	NPB 395	Oximeter, Pulse
	7200 A	Ventilator, Intensive Care
	7200 AE	Ventilator, Intensive Care
	840	Ventilator, Intensive Care
	Infant Star	Pediatric Ventilator
	Infant Star ISV 500	Pediatric Ventilator
	Infant Star ISV 950	Pediatric Ventilator
	Infant Star ISV 950+	Pediatric Ventilator
Viasys Healthcare	Bear 1000	Ventilator, Intensive Care
	Bear 1000t/es	Ventilator, Intensive Care
	Bear Cub 750vs	Ventilator, Intensive Care, Neonatal
	Bear Cub 750psv	Ventilator, Intensive Care, Neonatal
	Bird VIP (Standard)	Pediatric Ventilator
Welch Allyn	Propaq 102	Physiological Monitoring System, Acute Care, Portable
	Propaq 104	Physiological Monitoring System, Acute Care, Portable
	Propaq 106	Physiological Monitoring System, Acute Care, Portable



## Medical Device Connectivity

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS AREA

### Easy device connectivity is here at last!

DataCaptor is a data acquisition and distribution software enabling you to collect data from all types of medical devices and use it in your clinical application.

**4.4** DataCaptor 4.4 has built-in support for more than 250 diverse medical devices and includes many new features and enhancements. The DataCaptor Diagnostics package gives you the ability to troubleshoot different points in the data collection process. The new DataPortal and DMMServer Add-Ins allow you to connect to information systems using sockets and to customize the quantity and identification of data you collect.

#### On this page:

- ✚ About DataCaptor
- ✚ What's New in DataCaptor 4.4
- ✚ DataCaptor Architecture

#### About DataCaptor

DataCaptor is a software-only data acquisition tool that runs on Windows NT4 SP6, 2000 SP3, XP SP1 and Server 2003 systems, and sends data to clinical applications. The architecture—based on the Distributed Component Object Model (DCOM)—is highly scalable and modular. DataCaptor's easy-to-use graphical user interface makes it simple to configure and connect medical devices to your clinical application.

DataCaptor collects all variable data from bedside medical devices through any type of communication hardware or through a direct connection to a bedside computer. DataCaptor retrieves and delivers real-time data from more than 250 different medical devices. DataCaptor facilitates software creation by merging data flows and creating a common protocol for many different data sources, thus optimizing software development and application management.

#### What's New in DataCaptor 4.4

DataCaptor 4.4 provides new and enhanced features, including:

- **Remote DataCaptor Device Interface (DDI) Start and Stop Functionality**

DataCaptor 4.4 allows the client to start or stop a DDI remotely, without the use of the DataCaptor control panel.

- **Automatic Device Identification / Plug-and-Unplug Functionality**

The Automatic Device Identification feature of DataCaptor has been extended to support hardware detection for any device in the DataCaptor DDI Library. This feature relies on a specialized hardware component called a Device ID Module. Using the new Device ID Manager, users can program Device ID Modules to identify specific devices. To enable Automatic Device Detection, simply plug a device with its Device ID Module into the concentrator or serial port. This will trigger instant recognition of the device and startup of the associated DDI. Unplugging the device and plugging a different one (with an associated Device ID Module) will cause the DDI corresponding to

the newly plugged device to be started, without any changes to DataCaptor's configuration.

- **Chart on Change Data Management Module (DMM)**

The Chart on Change DMM allows for all values of monitored variables of a device to be sent to the client application only when one or more value of those monitored variables has changed.

- **Data Sampling DMM Improvements**

The DataSampling DMM has been completely redesigned and rewritten to provide significant performance gains and to provide consistency to the common DMMs user interface for easier configuration and use.

The DataSampling DMM samples data received from DataCaptor and sends it to the client application at regular intervals. This DMM enables:

- The client application to receive all data provided by DataCaptor grouped at regular intervals;
- Sampling based on last value, average value and other criteria;
- The ability to stop other DMMs from continuously processing data if the client application does not request a real-time data stream, thus reducing the CPU usage needed to process the device's data.

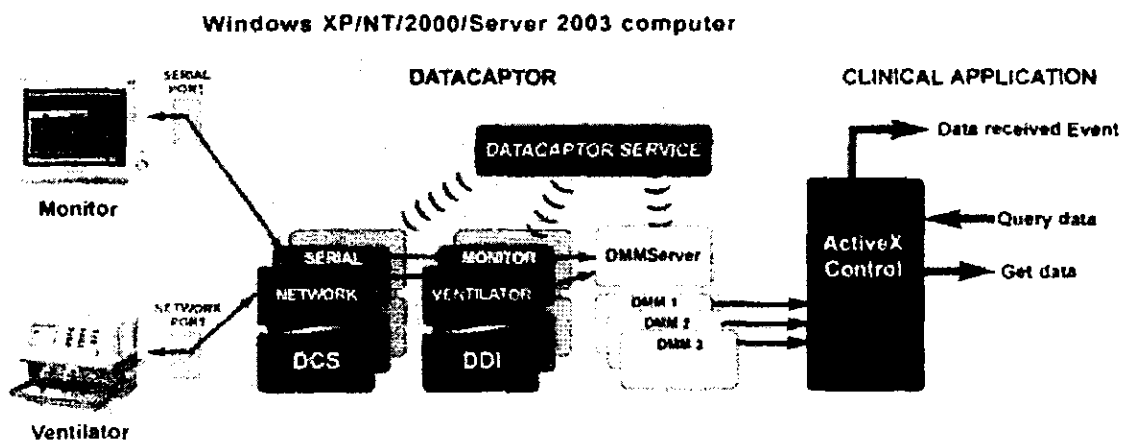
- **DMMAdmin Expert Mode**

DMMAdmin provides a new expert mode in the user interface. The expert mode displays information such as the events sent from DMMServer, the client status and client statistics.

- **DataCaptor Architecture Improvements**

Architectural improvements include a new connection from DataPortal to DataCaptor without a intermediary connection to DMMServer. This setting can be made on a per-data source basis, giving more flexibility for the configuration of production environments that concentrate several data sources in one DataPortal access.

### DataCaptor Architecture



- **DataCaptor Communication Servers (DCS)** control physical device ports. Each DCS module corresponds to a communications type such as serial, Ethernet, or proprietary network.

- **DataCaptor Device Interfaces (DDI)** manage communications with medical devices. Each DDI module interprets and utilizes the device protocol, managing low-level communications via the DCS.
- **DataCaptor Data Management Unit** receives the data acquired by each DDI and transmits it to the DataCaptor ActiveX Control.
- **DataCaptor ActiveX Control**, the programmers' interface to DataCaptor, acts as a data source. After receiving data from the Data Management Unit (either local or remote), the control makes it available to the calling application.
- **DataCaptor Service** oversees the whole system, loading and controlling the modules and correcting any errors, ensuring proper functioning.

**Additional DataCaptor Resources:**[!\[\]\(950a62bbddad88d64435fd35607dfc42\_img.jpg\) Library of Supported Devices](#)[!\[\]\(5a132f13505a6571904d622757b7a8f0\_img.jpg\) Case Studies & Testimonials](#)[!\[\]\(10f8862fc183b400327470ea85afe9ae\_img.jpg\) Medical Device Connectivity Report](#)[!\[\]\(e1d6102fe77919492c04879c8450f1f5\_img.jpg\) DataCaptor Information Sheet](#)[!\[\]\(73002692dd5e7a64e60946be3158e719\_img.jpg\) Automatic Device Detection](#)[!\[\]\(d5d7044e5caf6907399af2dced8d6ff8\_img.jpg\) Contact Capsule Technologie](#)[!\[\]\(35dc653d59570f8f891c312eeece91a2\_img.jpg\) DataCaptor Solutions](#)[!\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60\_img.jpg\) Request a Demo](#)

Copyright © 2003



## Medical Device Connectivity

Search: [SOLUTIONS](#)[DATACAPTOR](#)[NEWS](#)[COMPANY](#)[RESOURCES](#)[USERS' AREA](#)

DataCaptor System Interfaces, or DSIs, are modular components of DataCaptor that allow information systems to collect data from any disparate applications. Data from laboratory, admission, discharge and transfer, and other applications in the hospital can be sent to information systems in industry standard formats such as XML and HL7.

*DataCaptor System Interfaces are not currently available for download. If you are interested in interfacing to a laboratory, hospital, or clinical system, please contact us for more details.*

Copyright © 2000



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

## DataPortal™

## A new way to collect vital patient data

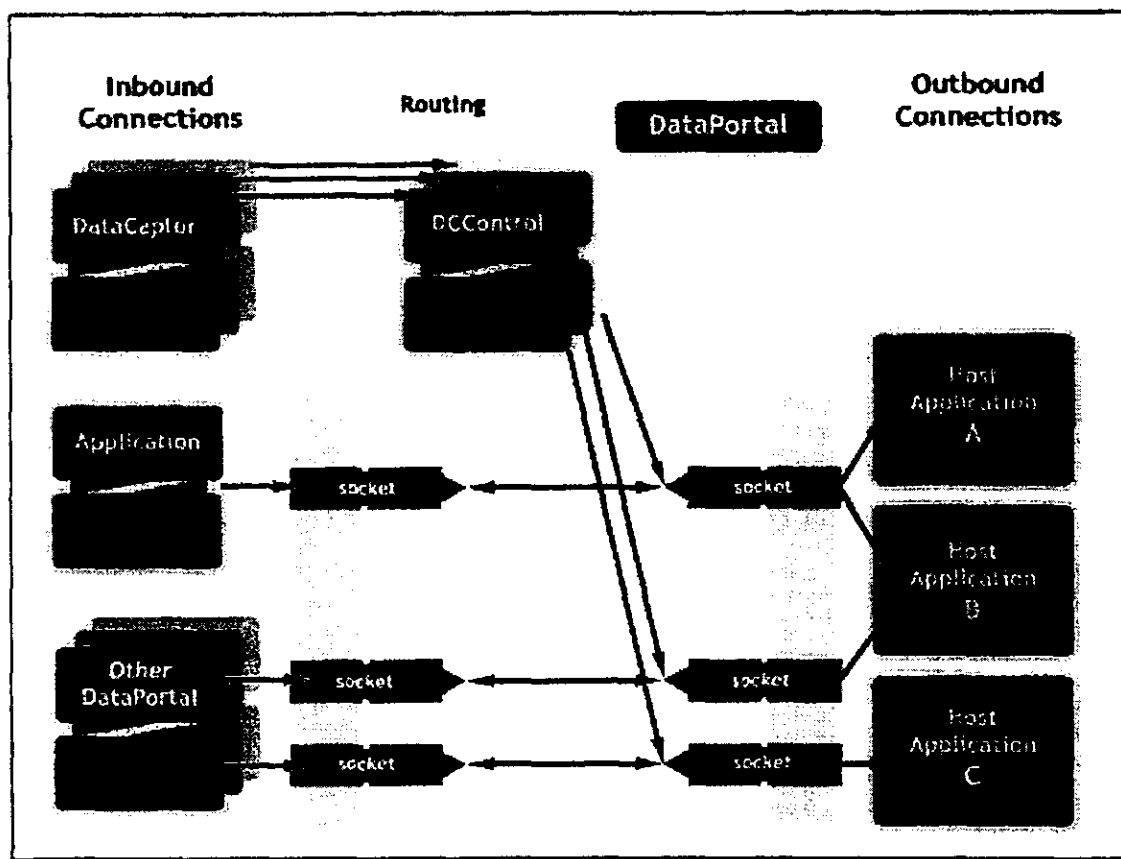
The DataPortal Add-In allows you to use data from DataCaptror without integrating the DCControl (ActiveX control) into your program. DataPortal collects data from DataCaptror and sends it to information systems via sockets. DataPortal provides access to data from DataCaptror when:



- The information system collecting data via DataCaptror does not run on Windows platforms
- It is more appropriate to collect data from DataCaptror via sockets than via the DCControl.

## How DataPortal works

DataPortal extends the functionality of DataCaptror by sending device data to any clinical information system regardless of the platform on which it runs.








DataPortal receives data from DataCaptor by using the ActiveX Control. The data is then routed through a socket connection to send the data on to the host application. DataPortal is used together with DMMServer to deliver data that is formatted exactly how the host application requires it.

Connections not originating from DataCaptor can also connect through sockets. Incoming data from socket connections can enter DataPortal in XML or HL7. Outbound connections transfer data from DataPortal to the host application through the network. Multiple data sources can be routed to any single outbound socket. However, all data that leaves any single socket must be in only one of two available formats: XML or HL7.

DataPortal can be installed on the DataCaptor server or on another PC on the network.

 [Download a DataCaptor product information sheet](#)  
This PDF document requires the free Adobe Acrobat Reader

 [Learn more about the benefits of DataCaptor](#)

 [Request a demo of the DataCaptor connectivity solution](#)

Copyright © 2004



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

Which devices do you support?

How do I request an interface to a device not currently in your interface library?

How long does it take to get delivery of a requested DDI?

What is the pricing structure of DataCaptor?

Who distributes DataCaptor in the United States and Europe?

What kind of technical support is offered with DataCaptor?

What kind of quality certification do you have?

Do you sell or recommend hardware for use with DataCaptor?

Which devices do you support?

With DataCaptor, you can receive data from all kinds of measuring devices with a data output port. We support all types of serial and network devices. DataCaptor does not treat DICOM images. See Device Interfaces for a complete list.

How do I request an interface to a device not currently in your interface library?

We can write interfaces to any device you need. Just let us know the manufacturer, model and version numbers, and we will get the necessary information to create the new interface and add it to our library. Please contact us for more information.

How long does it take to get delivery of a requested DDI?

It takes an average of three months between the day of your initial request and the delivery date. The steps are as follows: first, we contact the device manufacturer and review the protocol specifications, then we develop the DDI and finally we perform our validation procedure. The actual delay is largely dependent on the aid of the device manufacturer and the availability of the machine for testing purposes.

You will be kept informed of the target delivery date and progress.

What is the pricing structure of DataCaptor?

The full DataCaptor product is available at no cost for integration into your existing software. In order to collect data from devices, you must purchase "connections", referring to the physical connection to each

device. For example, if an ICU system is collecting data from 2 HP heart monitors and a Siemens respirator, that is equal to 3 connections. Changing a device does not affect the number of connections necessary. In the above example, changing one of the HP monitors to a GE monitor would not require purchasing a new connection. The interfaces to these machines are downloadable from our Web site, are quality assured and easily installed for immediate use.

#### Who distributes DataCaptor in the United States and Europe?

DataCaptor is conceived to be downloadable over the Internet, thus removing the need for resellers. We have direct contact with our clients that are located in several European countries and in the US.

#### What kind of technical support is offered with DataCaptor?

Our technical support philosophy is very simple: unlimited. The conception of the product does not lend itself to very many potential problems, but in the case that you do need support, we are willing to do everything in our power to help you. Sending an email to support@capsuletech.com will get you an immediate response to your technical questions. Our office is open from 9-6 Monday-Friday GMT +1. Any patches or fixes that we make available will be freely downloadable on our website, and our offer of support extends to those as well.

#### What kind of quality certification do you have?

Capsule Technologie is ISO 9001 certified by Moody International. DataCaptor and many of its device



No. 200005190  
ISO 9001 Certified  
AOQC Moody International



FDA 510(k) cleared



interfaces are FDA 510(k) cleared, with new device interfaces regularly being submitted for clearance. Capsule Technologie is fully committed to delivering total quality product and services. We believe that quality control and flexibility are the determining factors to a successful solution. Today, successful software providers have all been able to leverage their core business through intelligent use of third party solution providers.

#### Do you sell or recommend hardware for use with DataCaptor?

Capsule Technologie has certified several hardware options allow your application to fully leverage the use of a multi-serial port card, a concentrator, a multiplexer, an Ethernet port or of course a regular serial port.

For more information about products that can be used with DataCaptor, please see Hardware.



## Medical Device Connectivity

Search: [SOLUTIONS](#)[DATACAPTOR](#)[NEWS](#)[COMPANY](#)[RESOURCES](#)[USERS' AREA](#)

How do I access the data collected by DataCaptor?

On which operating systems does DataCaptor run?

Is there a limit to the number of devices I can interface?

How long does it take to configure a specific device?

Can I access my data from a remote location?

Can I keep my existing drivers?

Is a connection license good for more than one PC?

Can I control devices with DataCaptor interfaces?

---

How can I access the data collected by DataCaptor?

DataCaptor is an integration engine that collects data from medical devices and makes them accessible via an ActiveX Control. You can use this control in one of two ways:

- A. The ActiveX Control can be integrated into a clinical application that runs on Windows. The engineering effort required to do this depends largely on whether the clinical application is already capable of displaying patient trend data.
- B. The ActiveX Control can also be accessed with the DataPortal application. This application sends the data collected by DataCaptor to clinical applications via sockets, in the case that the clinical application is not Windows-based, or if you prefer not to integrate the ActiveX Control directly into the application code.

On which operating systems does DataCaptor run?

DataCaptor runs on Windows NT, 2000 and XP operating systems. If DataCaptor is used with DataPortal, it can send data to non-Windows applications.

Certified Platforms - DataCaptor 4.4.1 has been thoroughly tested for release against the following Windows platforms:

- Windows 2000 Server Service Pack 4 - US version
- Windows XP Professional Service Pack 1 - US version
- Windows Server 2003 - US version

Supported Platforms - Capsule Technologie will provide technical support for DataCaptor 4.4.1 if running

on the following Windows platforms:

- Windows NT4 Service Pack 6 – US version
- Windows XP Professional Service Pack 2 Release Candidate 1 – US version
- Windows 2000 Professional Service Pack 4 – US version

Please note that support for Windows NT will be retired with the next major or minor release of DataCaptor.

**Is there a limit to the number of devices that I can interface?**

No. DataCaptor can interface simultaneously to an unlimited number of devices. The only limits are due to the hardware platform that you intend to use for data acquisition. DataCaptor is a low level service and as such is extremely frugal with resources. On a standard PC sold today, you can easily run up to 100 device interfaces.

**How long does it take to configure a specific device?**

A few minutes are needed to configure a device. The DataCaptor Control Panel will guide you through this procedure and allow you to change devices on-line.

**Can I access my data from a remote location?**

Yes, thanks to the distributed architecture of DataCaptor, you can easily access the data from any station on your LAN. The same is true for the Internet, DataCaptor will work transparently across any LAN or WAN configuration.

**Can I keep my existing drivers?**

DataCaptor can coexist with most existing data acquisition solutions.

**Is a connection license good for more than one PC?**

A connection license can only be used on one PC at a time. Each installation of DataCaptor will create a unique Site Code that is only good for that PC. In the case of changing PCs, you can export a license from one PC to another.

**Can I control devices with DataCaptor Interfaces?**

DataCaptor does not control devices. DataCaptor does send requests for data, but it does not send commands that affect the device in any way.

Copyright © 2004



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

What is the output format of DataCaptor?

How can I access example source code?

Where can I find a Programmer's Guide?

Can I access the data kept in the DataStore?

Do you have a control for viewing data that can be plugged into an application?

How often does DataCaptor send data?

Can I use my own application to configure device connections?

How many variables are collected from each device? Are alarms collected?

Can I get a list of possible variables in my application?

Can I get a list of commands in my application?

---

What is the output format of DataCaptor?

The DataCaptor ActiveX Control outputs data in XML format. Other output formats are available by using DMMServer.

How can I access example source code?

By choosing "Full" at the time of the DataCaptor installation, example Visual Basic programs that use the ActiveX Control and their source code are installed on your PC. You can find the application code in the Capsule Technologie\DataCaptor\Example Source directory. This code is only to be used to illustrate how the ActiveX Control is integrated into a Visual Basic application - it is not to be used as a clinical application.

If you did not choose "Full" at installation, you can redo the setup and choose "Modify" to add the files that you are missing.

Where can I find a Programmer's Guide?

The Programmer's Guide is installed on your PC when you chose the "Full" setup option during the DataCaptor installation. You can then access it from the DataCaptor shortcuts on your Start menu or in the Capsule Technologie\DataCaptor directory.

If you did not choose "Full" at installation, you can redo the setup and choose "Modify" to add the files that you are missing.

You may also find a PDF version of the Programmer's Guide on our Web site (See Documentation Library)

**Can I access the data kept in the DataStore?**

The DataStore is used as a buffer, passing data from the DataCaptor Communication Servers on to the ActiveX Control. There are no historic data capabilities today.

**Do you have a control for viewing data that can be plugged into an application?**

At this time, we do not provide charting controls. Your application needs to have the capacity to view trend data in order to visualize the data DataCaptor is transmitting to it.

**How often does DataCaptor send data?**

DataCaptor transmits all parameter, alarm and device status data as often as it is sent by the device. You can determine in your application which variables you want to accept and how often, depending on your requirements.

**Can I use my own application to configure device connections?**

With DataCaptor version 4.2 and below, the DataCaptor Control Panel is the only way to configure devices.

**How many variables are collected from each device? Are alarms collected?**

DataCaptor collects every variable that is sent by a device - patient demographics, alarms, parameters, device status, etc. Alarms are collected for patient charting use, but cannot be used for remote monitoring purposes.

A complete list of variables sent by the device can be found in the device's Help file. (See Documentation Library)

**Can I get a list of possible variables in my application?**

The list of variables is sent in the XML message at connect.

**Can I get a list of commands in my application?**

As there are few commands compared to variables, there is not a way to receive a list of commands within your application. A complete list of commands is available in the devices' Help file. (See Documentation Library.)

Copyright © 2003



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

I have a message that says I am not authorized to use DataCaptor. Why?

I am not receiving data from DataCaptor. What should I check first?

When trying to run the DataCaptor Control Panel, I get an error that says a file is missing. Why?

I have a message that says I am not authorized to use DataCaptor. Why?

1. If this is the first time you are running DataCaptor, this means that you need to authorize your copy of the software. Open the DataCaptor Control Panel and go into Actions/Authorization. You can then either choose to run DataCaptor for 30-days or enter a connection license immediately. Contact Capsule Technologie with your Site Code, to obtain a Site Key for the number of connections you have ordered.
2. If you have been running the 30-day trial, your trial period may have expired. Go to Actions/Authorization in the DataCaptor Control Panel, to verify this. If you have reinstalled the software, you may be prompted to start your 30-day trial again, and then receive a message that says your trial period has already expired. A trial period is not renewed upon reinstallation. Contact Capsule Technologie with your Site Code, to obtain a Site Key for the number of connections you have ordered.
3. If you have deleted or moved any files from the Capsule Technologie\DataCaptor directory, you may have corrupted your license. If this is the case, contact Capsule Technologie.

I am not receiving data from DataCaptor. What should I check first?

Check that:

1. Check that the cable is well attached to both the device and the PC, checking all intermediary hardware that you may be using as well.
2. Be sure that the cable matches the cable specifications in the device's Help file.
3. Check that the connection license is still valid in the DataCaptor Control Panel in Actions/Authorization. It is normal that the site key field will be blank and you should see the number of connections you are authorized to use.
4. Check that you have not exceeded the number of connections you are authorized.
5. Make sure that you are running the required versions of Windows, Service Pack and Internet Explorer.
6. Check that the device is configured to send data via its data port. You may find information on the configuration needed in the device's Help file, or in the device's user manual provided by the manufacturer.



7. Check that DataCaptor is running by verifying that the icon appears in the system tray.

8. Verify that you can connect to DataCaptor with XMLDump.exe.

When trying to run the DataCaptor Control Panel, I get an error that says a file is missing. Why?

1. Versions 4.0 and below: If the file is an .OCX file, double check that you have run the DCOM for NT and 2000 setup BEFORE the DataCaptor setup. If the setups were done in the wrong order, simply repair the DCOM installation, then repair the DataCaptor installation.
2. Other files, such as configlib.dll may not "be found" if you are running versions of NT Service Pack or Internet Explorer that are inferior to the version specified by the system requirements. The file is not actually missing, but this is the error message given by the operating system. If this is the case, upgrade the program in question, then repair both the setups used to install your version of DataCaptor.

Copyright © 2003



## Medical Device Connectivity

Search

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

## What are the installation steps?

### What are the installation steps?

What are the systems requirements for DataCaptor and DataPortal?

Do I lose my connection license when I reinstall?

Can I use DataCaptor version 3 DDIs with DataCaptor version 4?

### What are the installation steps?

DataCaptor Core software:

In order to install DataCaptor, the installer must have administrative rights on the machine.

If a previous version of DataCaptor is installed on your machine, the DataCaptor setup will ask you if you wish to upgrade your installation. Clicking YES will take you through the normal installation procedure and all previous configurations will be saved for use with the upgraded version.

1. Run the CoreSoftwarePackage.exe.
2. DataCaptor offers you three types of installations. Select the one that best serves your requirements:
  - ☐ Run-time
  - ☐ Full
  - ☐ Custom

When you run the CoreSoftwarePackage.exe, Install Shield will install the following files onto your system:

#### Run-time setup option:

Support Files  
 Microsoft .NET Framework 1.0  
 DataCaptor  
 DataCaptor Communication Servers  
 Automatic Device Detection  
 DataCaptor ActiveX Control  
 DataCaptor Help (Online Help, Programmer's Guide, Generic Device Help)

#### Full setup option:

All the above, plus:  
 Sample Host Applications (VB App, VB XML App plus source code, XMLDump plus source code)  
 DCAPI  
 Generic Devices 1, 2, 4, 5, and Monitor  
 Diagnostics

**Custom setup option:**

Allows selecting any of the above files

**DataCaptor Device Interfaces:**

Device Interfaces may be installed from the Device Interface Setup Package, or from individual setups. The Device Interface Setup includes a custom setup option to choose the interfaces you are interested in installing.

In order to install the Device Interface Setup Package, the installer must have administrative rights on the machine.

To install DataCaptor Device Interfaces, run the DDI Setup.exe. Individual DDI setups are also available on the Capsule Technologie Web site.

**DataCaptor Add-ins:**

DataCaptor Options can be installed on a DataCaptor computer for a simple configuration, or on other network computers for a distributed configuration. If DataPortal is used, one or more DMM servers must be installed on the network. To use Diagnostics, DataCaptor and any options that are used must also be installed on the local machine, but not necessarily licensed.

In order to install the DataCaptor Options Package, the installer must have administrative rights on the machine.

If you are reinstalling the DataCaptor Options Package, make sure that neither DataCaptor Service nor its Options are running. In the DataCaptor Control Panel, click on Stop Service if a previous version of DataCaptor is running on your machine, and repeat the process for all other running programs from their respective control panels.

If a previous version of DataCaptor Options is installed on your machine, the DataCaptor setup will ask you if you wish to upgrade your installation. Clicking YES will take you through the normal installation procedure and all previous configurations will be saved for use with the upgraded version.

1. If the options are to be installed with DataCaptor on a computer, first install DataCaptor.
2. Run SetupOptionPackage.exe.
3. Select the installation that best serves your needs. The DataCaptor Options Package offers you three types of installations:
  - ☐ Run-time
  - ☐ Full
  - ☐ Custom

When you run the SetupOptionPackage.exe, Install Shield will install the following files onto your system:

**Run-time setup option:**

Support Files

Microsoft .NET Framework 1.0

DMM Server

DMM Administrator

Data Management Modules:

Device Type

Variable and Unit Mapping

Identifier Mapping  
Aperiodic  
Variable Type  
Data Sampling  
Variable Selection  
Unit Translation  
HL7 (ORU) Translation  
DataCaptor Help (Online Help, Programmer's Guide, Generic Device Help)  
Sample Host Application (DataClient)

**Full setup option:**

All the above, plus:  
Diagnostics

**Custom setup option:**

Allows selecting any of the above files

**What are the system requirements for DataCaptor?**

DataCaptor is designed to use system resources sparingly. Actual requirements will vary depending on the number of devices connected to each computer as well as various other factors.

To run DataCaptor and its add-ins, your computer must meet the following requirements:

- Internet Explorer 6 with Service Pack 1
- 500 MHz Pentium III processor, 256MB RAM
- The device connections require the appropriate physical connectors. When a direct serial connection from the medical device to a PC is used, it is recommended to use an optical isolating device connected to the serial cable.

If data is to be analyzed on remote machines, a network with a throughput equivalent to at least a 10 Mbps Ethernet is required. Actual network requirements will vary depending on the number of devices connected to each computer.

DataCaptor runs on Windows NT, 2000 and XP operating systems. If DataCaptor is used with DataPortal, it can send data to non-Windows applications.

**Certified Platforms** - DataCaptor 4.4.1 has been thoroughly tested for release against the following Windows platforms:

- Windows 2000 Server Service Pack 4 - US version
- Windows XP Professional Service Pack 1 - US version
- Windows Server 2003 - US version

**Supported Platforms** - Capsule Technologie will provide technical support for DataCaptor 4.4.1 if running on the following Windows platforms:

- Windows NT4 Service Pack 6 - US version
- Windows XP Professional Service Pack 2 Release Candidate 1 - US version
- Windows 2000 Professional Service Pack 4 - US version

Please note that support for Windows NT will be retired with the next major or minor release of DataCaptor.

Do I lose my connection license when I reinstall?

Your license will not be lost in any of the following cases:

- *Uninstall and reinstall DataCaptor*
- *Repair of a DataCaptor installation*
- *Modification of a DataCaptor installation to add new components*

If you are in your 30-day trial period, a reinstallation will not disrupt the days remaining for evaluation.

However, if you delete or move the Capsule Technologie system folder, you will lose your license.

Can I use DataCaptor version 3 DDIs with DataCaptor version 4.x?

Version 3 DDIs are compatible with DataCaptor version 4.x. However, version 4 DDIs may only be used with DataCaptor version 4 and above.

Copyright © 2004



## Medical Device Connectivity



**Surgical Information Systems Selects Capsule Technologie to Provide Universal Device Connectivity for its Perioperative Information System**

Advanced DataCaptor Solution Will Enable Plug-and-Play Connectivity for Operating Room Software to Any Bedside Medical Device

**FOR IMMEDIATE RELEASE**

ATLANTA and PARIS, February 22, 2004—Surgical Information Systems (SIS) and Capsule Technologie, the leader in medical device connectivity solutions, today announced that SIS will integrate Capsule Technologie's DataCaptor™ solution to collect and manage bedside medical device data in the SIS perioperative information system.

The integration of DataCaptor will allow the SIS surgery management system to quickly and easily collect, filter and manage data from an extensive range of bedside medical devices. Capsule Technologie's DataCaptor is a medical device connectivity engine that collects data from more than 250 supported medical devices and sends data to a host clinical information system. SIS offers healthcare providers with a comprehensive automated solution for effectively managing the entire perioperative case.

"Integrating the DataCaptor connectivity engine will allow us to offer our customers instant connectivity to an extensive library of bedside monitors, ventilators and infusion pumps, with the ability for Capsule Technologie to develop new interfaces as we need them," said Richard L. Jackson, president and CEO of Surgical Information Systems. "DataCaptor was clearly the most complete and advanced solution for device connectivity." The integration of DataCaptor allows SIS to provide a single integrated solution for point of care documentation for nursing and anesthesia care givers.

"The combination of Surgical Information Systems' number one-ranked perioperative information system and the DataCaptor connectivity engine will result in a more comprehensive electronic patient record," said Capsule Technologie CEO Arnaud Houette. "Ultimately, collecting and managing data from bedside medical devices will help improve patient safety in the OR."

**About Surgical Information Systems**

Surgical Information Systems provides the most comprehensive integrated suite of modules focused specifically on the OR to supply clinicians and administrators with ALL the clinical and financial data relative to every surgical case. The system includes modules for case and staff scheduling, materials management, rules-based charging, and clinical documentation for pre-admission testing (PAT) through post-anesthesia care unit (PACU). Other modules include automated surgeon's operative notes, and SIS PACS, which allows secure access to diagnostic images and videos from anywhere via the Internet. The company's StatCom-O.R. is a workflow communications tool that tracks the surgery department's most important assets - patients, beds, staff and equipment. Together, the system enables healthcare professionals to provide the highest quality care at the lowest possible cost. The Surgical Information Systems product utilizes an open architecture and a Microsoft (Nasdaq: MSFT) Windows- or Unix-based operating system that is tied into an Oracle (Nasdaq: ORCL) database. Visit the company's Web site at <http://www.ORsoftware.com>.

#### About Capsule Technologie

Capsule Technologie is the world's leader in Medical Device Connectivity software and solutions. The company's flagship product, called DataCaptor™, provides an interface between standalone and/or networked medical devices and clinical/hospital information systems. DataCaptor digitally collects, decodes and distributes all the medical data that is made available at the digital communication port of medical devices from manufacturers around the world. With built-in support for more than 250 diverse medical devices, DataCaptor includes the largest library of medical device interfaces available. Capsule Technologie is ISO9001 certified and DataCaptor is FDA 510(k) cleared.

For more information on Capsule Technologie, please visit the company's Web site at <http://www.capsuletech.com>.

DataCaptor™, DataPortal™, DMMServer™, and Automatic Device Identification™ are trademarks of Capsule Technologie.

#### For More Information

Lorin David Kalisky ( [lorink@capsuletech.com](mailto:lorink@capsuletech.com) )  
Marketing & Communications Manager  
Capsule Technologie  
In the US, call: 1-800-260-9537 x39  
In Europe, call +33 (1) 53.34.14.12

Lisa Maggart / ( [maggart@ORsoftware.com](mailto:maggart@ORsoftware.com) )  
Manager, Public Relations  
Surgical Information Systems  
+1-770-643-5547

Copyright © 2003



## Medical Device Connectivity

Search: 

SOLUTIONS

DATACAPTOR

NEWS

COMPANY

RESOURCES

USERS' AREA

**DataCaptor is designed for you**

With more than 1,000 different bedside devices on the market—monitors, ventilators, anesthesia machines, infusion pumps, blood gas analyzers, dialysis machines and others—trying to develop and maintain a complete connectivity solution is an overwhelming task.

**Some common challenges faced by CIS providers and system integrators:**

- Your customers need up-to-date device drivers forcing you to *reallocate* valuable development resources.
- Access to documented protocols, effective support and test devices from device manufacturers is time-consuming and expensive.
- Drivers regularly become outdated or obsolete requiring that you redevelop them often in order to maintain connectivity for your installed base.

**The results can be painful for you and your customers:**

- Long delivery time for your new drivers.
- Insufficient ROI of your *driver development* activities.
- Customer dissatisfaction when drivers cannot be supplied or become outdated.

**DataCaptor is a solution designed for you.**

Take advantage of the largest library of device interfaces available on the market and refocus valuable development resources on your core business.

**Additional Resources:**

- [➔ Learn more about the benefits of DataCaptor](#)
- [➔ Explore the CIS Provider Partnership Agreement](#)
- [➔ Request a Demo](#)
- [➔ Read how to purchase DataCaptor connections](#)

Copyright © 2003



# SURGICAL INFORMATION SYSTEMS

[Sea](#)

**O.R. Software for the Whole Operation**

**Customer**

[Home](#)

[Products](#)

[Services](#)

[News](#)

[Newsletter](#)

[Resources](#)

[Alliances](#)

[Contacts](#)

[Careers](#)

[SIS India](#)

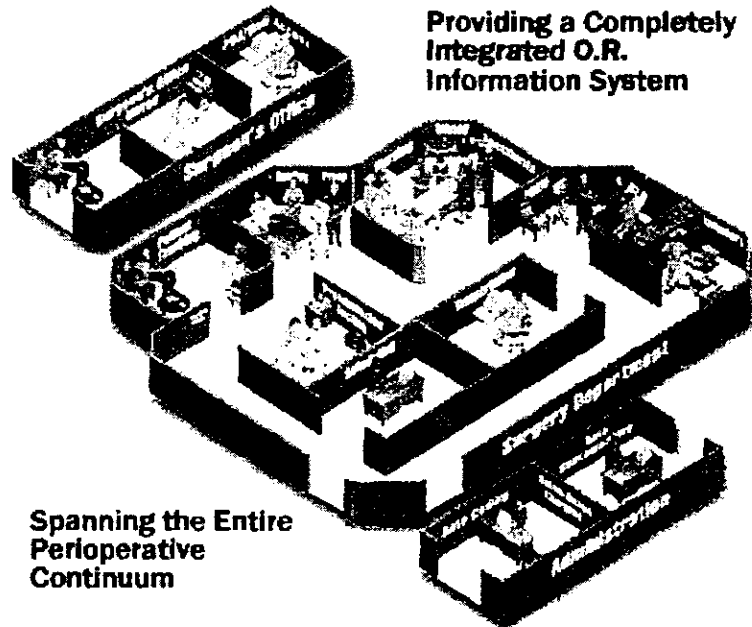
## About SIS Hospital Software

Surgical Information Systems offers hospital surgery software that captures clinical, medical and financial data on every patient event from surgery scheduling through transcription of the surgical report.

[Learn more](#)

## SIS Products

<a href="#">Scheduling</a>	<a href="#">Reporting</a>	<a href="#">StatCom</a>
<a href="#">Nursing</a>	<a href="#">Rules-Based</a>	<a href="#">O.R.</a>
<a href="#">Anesthesia</a>	<a href="#">Charging</a>	<a href="#">Surgeon's</a>
	<a href="#">SISWeb</a>	<a href="#">Notes</a>
<a href="#">Inventory</a>		<a href="#">Staff</a>
		<a href="#">Scheduling</a>



**Providing a Completely Integrated O.R. Information System**

**Spanning the Entire Perioperative Continuum**

## What's New

[Learn more about products from SIS](#)

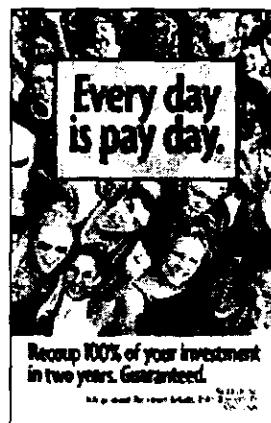
[Contact us to request more information](#)

**Surgical Information Systems Doubles Customer Base, Reaches 200th Hospital Milestone**

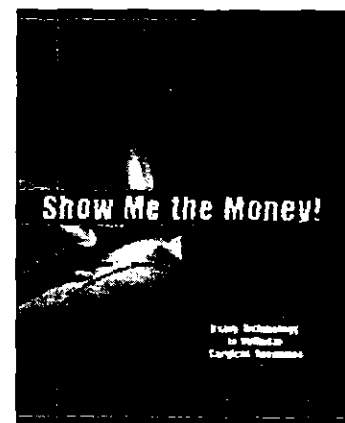
**Surgical Information Systems Launches New Alert System**

**Surgical Information Systems Selects Capsule Technology to Provide Universal Device Connectivity for its Perioperative Information System**

**Show Me the Money!**  
Using Technology to Optimize Surgical Revenues



**Every Day is Payday!**  
Learn about our  
Guaranteed ROI program



**Show me the Money!**  
Using Technology to  
Optimize Surgical Revenues

## Testimonials

*"The system improved patient care, and even paid for itself within 9 months!"*

**Dr. Samuel Mahaffey**

Former Medical Director of  
Perioperative Services, 1997 -  
2003  
Duke University Medical Center  
Durham, NC

[Read more testimonials](#)

## Stay on "The Cutting Edge"

SIS publishes a free surgery journal and medical newsletter called The Cutting Edge, which offers the latest healthcare news and strategies of interest to perioperative professionals and others within the operating room environment.  
[Subscribe today](#)

[Home](#) | [Products](#) | [Services](#) | [News](#) | [Newsletter](#) | [Resources](#) | [Contact Us](#) | [Site Map](#) | [Careers](#)

Copyright © 1999-2004 Surgical Information Systems  
3650 Mansell Road, Suite 300, Alpharetta, GA 30022 USA  
1-800-866-0656



## Company News

### Press Releases

**For Immediate Release**

February 22, 2004

Contact: Lisa Maggart  
770-643-5547

### **Surgical Information Systems Selects Capsule Technologie to Provide Universal Device Connectivity for its Perioperative Information System**

*Advanced DataCaptor Solution Will Enable Plug-and-Play Connectivity for Operating Room Software to Any Bedside Medical Device*

ATLANTA and PARIS, February 22, 2004—Surgical Information Systems (SIS) and Capsule Technologie, the leader in medical device connectivity solutions, today announced that SIS will integrate Capsule Technologie's DataCaptor™ solution to collect and manage bedside medical device data in the SIS perioperative information system.

The integration of DataCaptor will allow the SIS surgery management system to quickly and easily collect, filter and manage data from an extensive range of bedside medical devices. Capsule Technologie's DataCaptor is a medical device connectivity engine that collects data from more than 240 supported medical devices and sends data to a host clinical information system. SIS offers healthcare providers with a comprehensive automated solution for effectively managing the entire perioperative case.

"Integrating the DataCaptor connectivity engine will allow us to offer our customers instant connectivity to an extensive library of bedside monitors, ventilators and infusion pumps, with the ability for Capsule Technologie to develop new interfaces as we need them," said Richard L. Jackson, president and CEO of Surgical Information Systems. "DataCaptor was clearly the most complete and advanced solution for device connectivity." The integration of Data Captor allows SIS to provide a single integrated solution for point of care documentation for nursing and anesthesia care givers.

"The combination of Surgical Information Systems' number one-ranked perioperative information system and the DataCaptor connectivity engine will result in a more comprehensive electronic patient record," said Capsule Technologie CEO Arnaud Houette. "Ultimately, collecting and managing data from bedside medical

devices will help improve patient safety in the OR."

#### **About Surgical Information Systems**

Surgical Information Systems provides the most comprehensive integrated suite of modules focused specifically on the OR to supply clinicians and administrators with ALL the clinical and financial data relative to every surgical case. The system includes modules for case and staff scheduling, materials management, rules-based charging, and clinical documentation for pre-admission testing (PAT) through post-anesthesia care unit (PACU). Other modules include automated surgeon's operative notes, and SIS PACS, which allows secure access to diagnostic images and videos from anywhere via the Internet. The company's StatCom-O.R. is a workflow communications tool that tracks the surgery department's most important assets - patients, beds, staff and equipment. Together, the system enables *healthcare professionals* to provide the highest quality care at the lowest possible cost. The Surgical Information Systems product utilizes an open architecture and a Microsoft (Nasdaq: MSFT) Windows- or Unix-based operating system that is tied into an Oracle (Nasdaq: ORCL) database. Visit the company's Web site at <http://www.ORsoftware.com>.

#### **About Capsule Technologie**

Capsule Technologie is the world's leader in Medical Device Connectivity software and solutions. The company's flagship product, called DataCaptor™, provides an interface between standalone and/or networked medical devices and clinical/hospital information systems. DataCaptor digitally collects, decodes and distributes all the medical data that is made available at the digital communication port of medical devices from manufacturers around the world. With built-in support for more than 240 diverse medical devices, DataCaptor includes the largest library of medical device interfaces available. Capsule Technologie is ISO9001 certified and DataCaptor is FDA 510(k) cleared. For more information on Capsule Technologie, please visit the company's Web site at <http://www.capsuletech.com>. DataCaptor™, DataPortal™, DMMServer™, and Automatic Device Identification™ are trademarks of Capsule Technologie.

###

Contact:

Lisa Maggart / [maggart@orsoftware.com](mailto:maggart@orsoftware.com)  
Manager of Public Relations  
Surgical Information Systems  
770-643-5547

Richard L. Jackson, President and CEO  
Surgical Information Systems  
770-643-5605

Lorin David Kalisky / [lorink@capsuletech.com](mailto:lorink@capsuletech.com)  
Manager, Marketing & Communications

Capsule Technologie  
EU: +33 1 5334 1412 / USA: 1-800-260-9537

[Home](#) | [Products](#) | [Services](#) | [News](#) | [Newsletter](#) | [Resources](#) | [Contact Us](#) | [Site Map](#) | [Careers](#)

Copyright © 1999-2004 Surgical Information Systems  
3650 Mansell Road, Suite 300, Alpharetta, GA 30022 USA  
1-800-866-0656