(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 1 210 006 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
06.06.2007 Bulletin 2007/23

(21) Application number: **00959696.6**

(22) Date of filing: **31.08.2000**

(51) Int Cl.:
A61B 5/00 *(2006.01)*

(86) International application number:
PCT/US2000/023966

(87) International publication number:
WO 2001/015599 (08.03.2001 Gazette 2001/10)

(54) **SYSTEM AND METHOD FOR GENERATING AND TRANSFERRING DATA**

SYSTEM UND VERFAHREN ZUR GENERIERUNG UND ÜBERTRAGUNG VON DATEN

SYSTEME ET PROCEDE DE GENERATION ET DE TRANSFERT DE DONNEES

(84) Designated Contracting States:
DE FR

(30) Priority: **31.08.1999 US 387013**

(43) Date of publication of application:
05.06.2002 Bulletin 2002/23

(73) Proprietor: **DATA CRITICAL CORPORATION**
Bothell, WA 98011 (US)

(72) Inventors:
- **ALBERT, David, E.**
  Oklahoma City, OK 73120 (US)
- **RIEGER, Carl, J.**
  Oklahoma City, OK 73120 (US)
- **REITER, Mac L.**
  Stillwater, OK 74075 (US)
- **SLAVEN, Rik**
  Bellevue, WA 98008 (US)

(74) Representative: **Goode, Ian Roy**
London Patent Operation
General Electric International, Inc.
15 John Adam Street
London WC2N 6LU (GB)

(56) References cited:

EP-A- 0 779 057	**US-A- 5 626 144**
US-A- 5 735 285	**US-A- 5 810 747**
US-A- 5 997 476	**US-A- 6 022 315**
US-A- 6 090 056	**US-A- 6 095 985**

EP 1 210 006 B1

Description

BACKGROUND OF THE INVENTION

[0001] This invention relates generally to systems and methods for generating and transferring medical and other data. A specific implementation includes a remote heart monitoring method.

[0002] Many industries use real-time information, but the medical field is one in which such information is particularly critical. In hospitals, for example, patients may have their biological functions or conditions continuously monitored so that any significant change can be immediately detected and addressed by appropriate personnel. Such monitoring includes generating the information and transferring it to where it is to be used.

[0003] United States Patent No. 5,481,255 to Albert et al. discloses transmitting medical data, for example, through a paging network to a pager receiver connected to a computer, such as a palmtop computer held by a physician. In one example, a patient is connected to an electrocardiogram (ECG) machine to generate ECG information that is processed and transmitted through the paging network to the receiver and attached computer.

[0004] United States Patent No. 5,735,285 to Albert et al. discloses another communication system for conveying ECG data or other biomedical waveform data between a patient and an attending physician's location. The patient employs a HEART CARD™ device from Instromedix, Inc., or the like, which converts the patient's ECG signal into a frequency modulated audio signal that may then be analyzed by audio inputting via a telephone system to a selected hand-held computer device with integrated microphone and audio system. The computer device functions to digitize, record and demodulate the frequency modulated signal for presentation and viewing on the hand-held computer. The audio signal can also be input directly into a personal computer (PC) via a standard personal computer microphone. The audio signal can be input directly into a PC through a phone line connection via a standard "voice-capable" modem, for example. The stored ECG audio signal can be retransmitted via telephone, either wireline or wireless. At a receiving end, a programmed hand-held computer can be used to receive the audio FM biomedical signal for digitization, recording and demodulation for viewing. Such computer can be one with integrated microphone, audio analog to digital converter, digital to analog converter, speaker, and central processing unit with memory for performing various computational, data storage and signal processing tasks.

[0005] Document US 5,810,747 discloses a system for generating and transferring medical data over the Internet comprising a computer receiving biological signals over an input unit.

[0006] Despite previously disclosed or implemented systems, there is still the need for a novel and improved system and method for generating and transferring medical (or other) data, especially a system and method which enable a patient to communicate with a medical care provider in real time virtually from anywhere. Preferably such a system and method should be able to communicate globally, such as via the Internet. Such a system and method should be inexpensive, simple, and easy-to-use. This includes use of a computer without having to purchase or manually install a specialized computer application program or any special hardware. Such system and method should be relatively inexpensive both for the patient or other input user and the medical care provider or other recipient to use (e.g., minimal equipment requirements, no long distance telephone tolls). Such system and method should quickly (ideally, in real time) provide accurate data from the input user to the recipient.

SUMMARY OF THE INVENTION

[0007] The present invention meets the aforementioned needs by providing a novel and improved system and method for generating and transferring medical and other data. For example, the system and method permit the monitoring of biological functions and conditions of a plurality of patients and the reporting of the resulting data to respective medical care providers. In a particular implementation, the present invention provides for remote heart monitoring.

[0008] Various aspects and embodiments of the present invention are defined in the appended claims.

[0009] One definition of the system for generating and transferring medical data of the present invention comprises: means for sensing a biological function or condition; and means, communicating with the means for sensing, for transferring a response to the sensed biological function or condition over the Internet.

[0010] In accordance with another definition of the present invention, the system comprises a sensor used by a human user to sense a function or condition of the user's body. It also comprises a personal computer located with the sensor. The personal computer has a microphone connector and analog to digital conversion means which are communicated with the sensor such that a digital processing circuit of the personal computer receives from the analog to digital conversion means a digital electric signal derived from an analog electric signal received through the microphone connector in response to the user's body function or condition sensed by the sensor. The personal computer is programmed to transfer data representative of the received digital electric signal over a computer communication network contemporaneously with the sensor sensing the user's body function or condition. This system can further comprise a recipient computer located at a medical facility and connected to communicate with the computer communication network, wherein the

recipient computer is programmed to receive the data transferred over the computer communication network and to provide a visible representation of the sensed body function or condition and to store the data in a searchable database. The system can comprise a recipient computer located at a data communication service provider facility and connected to communicate with the computer communication network, wherein the recipient computer is programmed to receive the data transferred over the computer communication network and to transmit in response thereto another signal to an end user.

[0011] Still another definition of the present invention for a system for generating and transferring medical data comprises: a sensor to provide an acoustical signal in simultaneous response to a biological function or condition; a microphone, located with the sensor, to receive the acoustical signal as the acoustical signal is provided from the sensor; and a computer connected to the microphone and adapted to transfer over the Internet, in real-time response to the microphone receiving the acoustical signal, electric signals representative of the received acoustical signal. In one embodiment the computer includes a personal computer programmed with a web browser and a medical data acquisition and transmission program. The medical data acquisition and transmission program can be downloaded from an Internet site accessed using the web browser.

[0012] The system of the present invention can also be defined without limitation as to type of data, such as to comprise: a plurality of initial user sites, each of the initial user sites having a data generating source and a computer connected to receive signals from the data generating source and to access the Internet; and an Internet site addressable from each of the plurality of initial user sites such that each initial user site can access a data acquisition and transmission program through the Internet site to enable the respective computer to process received signals from the respective data generating source and transmit responsive signals over the Internet. Each initial user site preferably can transmit simultaneously over the Internet to one or more recipient sites. Also, the receiving computer at a recipient site can receive and display in real time the signals from one or a multiple number of initial user sites.

[0013] A method of generating and transferring medical data in accordance with an aspect of the present invention comprises: transducing directly from a patient a human body function or condition into a first signal; converting at the site of the patient the first signal into a second signal adapted for transmission over the Internet; and transmitting the second signal over the Internet to a recipient. In one implementation the method further comprises performing the transducing, converting, and transmitting contemporaneously; and contemporaneously therewith generating at the recipient a display in response to the transmitted second signal, wherein the display is a real-time representation of the transduced human body function or condition.

[0014] A method of generating and transferring data in accordance with the present invention comprises: accessing an Internet site from a computer where a data generating source is located; downloading to the computer from the accessed Internet site a data acquisition and transmission program; and operating the computer with the downloaded data acquisition and transmission program such that the computer receives signals from the data generating source, processes the received signals, and transmits data signals onto the Internet in response thereto. In particular implementations, the downloaded program may or may not operate unless the computer is connected to the specific web site on the Internet from which the program was obtained.

[0015] A method of the present invention for generating medical data representative of a biological function or condition of an individual comprises:

(a) accessing an Internet site with a personal computer located with the individual, wherein the personal computer is programmed with a conventional operating program;
(b) downloading from the accessed Internet site to the personal computer a medical data acquisition program automatically operable with the conventional operating program;
(c) generating an acoustical signal in response to the biological function or condition of the individual;
(d) receiving the acoustical signal in a microphone near the individual and converting with the microphone the received acoustical signal into an analog electric signal;
(e) communicating the analog electric signal from the microphone to an analog to digital conversion circuit of the personal computer and converting thereby the analog electric signal to a digital electric signal;
(f) processing the digital electric signal in the personal computer under control of the medical data acquisition program; and
(g) displaying to the individual, through the personal computer, a representation of the individual's biological function or condition in response to which the acoustical signal was generated.

As a step (h), responsive data can be transmitted over the Internet to other sites for storage and review. From such other site or sites, for example, data can then be transmitted over a paging or other wireless network to a physician, for example. In a preferred embodiment, at least steps (c) through (h) are performed together in real time.

[0016] Still another definition of a method of monitoring biological functions and conditions of a plurality of patients in accordance with the present invention comprises: distributing to each patient at least one sensor to detect at least one

biological function or condition of the patient; and maintaining a medical data acquisition and transmission program at an Internet site accessible by the patients such that the patients can use, from computers at the locations of the patients, the medical data acquisition and transmission program to control their respective computers to receive and process signals from the patients' respective sensors and to transmit medical data onto the Internet in response thereto. This method can further comprise distributing to a plurality of physicians

[0017] Therefore, from the foregoing, it is a general object of the present invention to provide a novel and improved system and method for generating and transferring medical and other data Other and further objects, features and advantages of the present invention will be readily apparent to those skilled in the art when the following description of the preferred embodiments is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018]

FIG. 1 is a block diagram illustrating one configuration of a system of the present invention.
FIG. 2 is a block diagram illustrating one variation of the FTG. 1 configuration wherein a patient site accesses a log-on site to acquire a software program necessary to provide medical data to a recipient site.
FIG. 3 is a block diagram illustrating one implementation of the system depicted in FIG. 2.
FIG. 4 is a block diagram illustrating another implementation of the system depicted in FIG. 2.
FIG. 5 illustrates one implementation for a patient site.
FIG. 6 is a general flow diagram for the use and operation of the implementations of FIGS. 2-5.
FIG. 7 is a more detailed block diagram of the patient site depicted in FIG. 5.
FIG. 8 illustrates a particular implementation of a system of the present invention including at least one patient site of the type shown in FIG. 7.
FIG. 9 is a flow diagram for a medical data acquisition and transmission program for the FIG. 8 system.
FIG. 10 is a flow diagram for displaying data at a recipient Internet site remote from the patient's site of the FIG. 8 system.

DETAILED DESCRIPTION OF THE INVENTION

[0019] A general representation of the system of the present invention as applied to one example of medical data acquisition and transmission is illustrated in FIG. 1. The present invention also can be used for generating and transferring other data that can be processed in the manner described below.

[0020] Although only one patient site need be present, typically there will be a plurality of patient sites 2a-2n (the term "patient" as used herein is not limited to someone who is ill or who is necessarily under a medical provider's care; it simply refers to an individual using the present invention in the medical context). These sites are wherever the patient (or, more broadly, initial user or simply some data generating source) is and has access to a computer and communication network, which is specifically illustrated as including the Internet 4 in the drawings and in the remainder of the following description of the present invention (the term "Internet" as used herein encompasses the global computer network commonly known by that name, any functional part of it suitable for use in the present invention (e.g., the World Wide Web), and any other suitable global computer network capable of providing the communication link of the present invention). Also connected to the Internet 4 is at least one remote site 6 with which the patient sites 2a-2n can communicate.

[0021] The remote Internet site(s) 6 can be embodied in various manners; however, two particular types of the preferred embodiment are represented in FIG. 2. One or more log-on sites 6a are addressed by a patient at his or her site entering the log-on site's address. Such a log-on site 6a is the only site a patient need explicitly address from the respective patient site 2.

[0022] Although the foregoing is the only explicit address a patient need enter, other remote sites can be accessed, such as illustrated by recipient sites 6b in FIG. 2. In preferred embodiments, such sites are automatically addressed when data is to be sent from a patient site 2 to an end user. This can occur through a log-on site 6a, but in the illustrated embodiment of FIG. 2 there are physically and functionally distinct recipient Internet sites.

[0023] A further overview of the foregoing will be described with reference to FIGS. 3-6. A more detailed explanation will then be given with reference to FIGS. 7-10.

[0024] Referring to FIG. 3, a patient site 2 includes means for sensing a biological function or condition (more broadly, a data generating source; for example, a device that responds to some ambient parameter by generating a signal adaptable for use by a computer as described herein). It also includes means, communicating with the means for sensing, for transferring a response to the sensed biological function or condition over the Internet. The means for sensing includes at least one sensor 8 suitable for sensing a function or condition of the body of the respective patient. The means for transferring includes a computer 10 connected to receive signals from the one or more sensors 8 and to access the

Internet 4.

[0025] In a particular implementation, the sensor 8 includes a hand-held device that is inexpensive and simple to use. As illustrated in FIG. 5, the sensor 8 includes a heart monitor that provides an audible output signal in response to the sensed biological function or condition, which in this example is the patient's heart activity. The patient can hold this device against his or her chest by hand, this can be done by another person, or the device can be attached to the chest or held by an elastic strap (for example). The sensor 8 of this implementation in FIG. 5 includes a hand-held transducer that converts energy from an adjacent beating heart of a patient 12 into an acoustical signal. The acoustical signal can be detected by a microphone 14 connected to the conventional microphone jack of a personal computer 16 implementing the computer 10. The term "personal computer" as used in this specification and claims encompasses all types of computers conventionally referred to by that name regardless of size designation (e.g., desktop, laptop, notebook, hand-held, palmtop, etc.). Non-limiting examples of sensors are those capable of providing outputs responsive to (1) changes of electrical potential occurring during the heartbeat to provide ECG data, or (2) brain waves to provide EEG data, or (3) variations in the size of an organ or limb and in the amount of blood present or passing through it to provide plethysmograph data, or (4) galvanic skin resistance.

[0026] It is to be noted that any particular sensor 8 merely needs to provide some signal suitable for inputting to the computer 10 (thus "data" of the present invention includes anything represented by such signal). The microphone 14 is used in the embodiment of FIG. 5 to convert the acoustical signal of this embodiment into a corresponding analog electric signal used by the personal computer 16. The sensor 8 can be a type that provides an electric signal output directly to the personal computer 16, for example.

[0027] The personal computer 16 in the FIG. 5 implementation is connected to the microphone 14 and includes circuits and programs to access the Internet 4 and to transfer a representation of the sensor's output signal over the Internet 4 to a selected Internet site in response to input to the personal computer 16 from the microphone 14. That is, the personal computer 16 transfers over the Internet 4 data representative of the electric signal received either from the microphone or other device, such as the sensor 8 itself, into the personal computer 16. The personal computer 16 need be programmed initially only with conventional programs sufficient to enable the patient or other user at the patient site to access the Internet 4 through the personal computer 16. This programming can include a conventional web browser. At some point in the use of the personal computer 16, it is also programmed with a medical data (or other specific type of data as used for other embodiments) acquisition and transmission program (in broader terms, simply a data acquisition and transmission program). In the preferred embodiment, this program is downloaded into the personal computer 16 from the log-on site 6a using the web browser. The program can be in any suitable form, such as in the form of an application "plug-in," a "Java applet," or in a preferred particular implementation an "ActiveX control." The personal computer 16 connects to the Internet 4 by any suitable communication link (e.g., telephone line, television cable).

[0028] The circuitry and programming of the personal computer 16 also ultimately are sufficient to provide it with the capability to read the input port receiving the signal from the microphone 14 or other input device, to process the input signal, to access the recipient Internet site to which data is to be transferred, and to send the data to the recipient site over the Internet.

[0029] Referring to FIG. 3 again, the log-on site 6a in this embodiment includes a computer 18 having the medical data acquisition and transmission program stored for downloading to a patient site 2. That is, the log-on site 6 of this implementation is an Internet site addressable from each of the plurality of patient sites such that each patient site can access the medical data acquisition and transmission program through the Internet site to enable the respective computer 10 (e.g., personal computer 16) at the respective patient site to process received signals from a patient sensor connected to it and to transmit responsive signals over the Internet 4.

[0030] The log-on site 6 can be implemented in any suitable manner. For example, it can be provided by an existing Internet service provider which the computer at a patient site accesses when the patient initially logs onto the Internet.

[0031] Still referring to FIG. 3, the recipient site 6b includes a computer 20 programmed to receive data and display it through a display 22 (e.g., a monitor or printer). In the implementation of FIG. 3, the computer 20 receives the transferred response over the Internet and directly communicates the information via the display 22. For example, the computer 20 can be located at a medical facility and connected to communicate with the Internet. Such computer is programmed to receive the response transferred over the Internet and to provide a visible representation of the sensed biological function or condition.

[0032] Another implementation of a recipient site 6b is illustrated in FIG. 4. In this implementation, the recipient site 6b functions as an intermediary between a patient site and an end user of the data. As shown in FIG. 4, the recipient site 6b still includes the computer 20, but which is additionally programmed to transfer the data to something other than the display 22. In FIG. 4, the computer 20 is programmed to communicate the data to a wireless network 24 (but other communication techniques can be used in other embodiments) that sends the data through its system to a wireless data receiver and computer combination 26. The wireless system is one that can provide data transmission (e.g., digital phone systems, such as GSM or CDMA which provide data transmission; two-way paging systems; one-way paging systems). This type of transfer can occur in accordance with the known technology described in the patents cited in the

background explanation set forth above, which are incorporated herein by reference. In this implementation, the recipient computer 20 can be located at a data communication service provider facility. This computer 20 is connected to communicate with the Internet 4, and it is programmed to receive at least part of the responsive signals transmitted over the Internet and to transmit in response thereto end user signals from the recipient computer 20 to an end user. The end user signals are a representation of the human body function or condition (or other data) as initially sensed by the sensor 8.

[0033] As mentioned above, the log-on site 6a and the recipient site 6b can be combined such that a single site provides both the medical data acquisition and transmission program to be downloaded to the patient site and also the programming necessary to receive data and directly display it or transfer it by other means to an end user. This is illustrated in FIG. 8 which is described below.

[0034] The method of the present invention for the foregoing embodiments is now described with reference to FIG. 6. Using the computer 10 (e.g., personal computer 16) where the patient is located, the Internet site having the medical data acquisition and transmission program is accessed from the respective patient's site. Access occurs by the computer 10 at the patient site 2 running its web browser or otherwise gaining access to the Internet 4. Once Internet access has been gained, the computer 10 accesses the known log-on site where the medical data acquisition and transmission program is located. This can be the site through which Internet access is gained, or the site containing the medical data acquisition and transmission program can be different from the Internet portal.

[0035] When the appropriate site has been accessed, identification information (e.g., name, password) is entered (e.g., manually by the patient or automatically by preprogramming the patient's computer 10) to enable the computer 10 at the patient site 2 to select and download the medical data acquisition and transmission program. When the appropriate information has been given to the log-on site 6a in the illustrated embodiments of FIGS. 3 and 4, the medical data acquisition and transmission program downloads to, or otherwise becomes accessible for use by, the computer 10 at the patient site 2.

[0036] The medical data acquisition and transmission program preferably downloads in active mode so that it is ready to perform the functions at the patient site 2 without the patient having to perform functions other than manipulating the sensor to sense the biological function or condition to be transmitted to the computer 10 at the patient site. In the implementation of FIG. 5, the patient 12 holds the heart monitor 8 against his or her chest. The heart monitor 8 responds to the beating heart by generating corresponding acoustical signals transmitted to the microphone 14. The microphone 14 converts the signals to analog electric signals communicated to the personal computer 16 programmed with the downloaded medical data acquisition and transmission program. In using the medical data acquisition and transmission program, the personal computer 16 receives the signals, processes the received signals, and transmits medical data signals onto the Internet 4 in response.

[0037] Transmitting over the Internet 4 to a recipient site 6b includes accessing a second Internet site from the computer 10 and transmitting patient data over the Internet to the accessed second Internet site 6b, both under control of the medical data acquisition and transmission program used by the computer 10.

[0038] In addition to the foregoing which is illustrated in FIG. 6, the method of the present invention can further comprise receiving, over the Internet, the transmitted medical data signals at a recipient site. As explained above, this can be any suitable site, such as a medical facility or a data communication service provider facility. The former would typically be an end user of the data whereas the latter would typically be an intermediary that transfers the data onto an end user. As explained above with reference to FIG. 4, such could include transmitting, from the data communication server provider facility to an end user, end user signals responsive to the received medical data signals. In FIG. 4, this includes transmitting the medical data signals through the wireless data network 24.

[0039] When the data is received by the end user, a display is generated in response to a received signal. The display is a representation of the transduced human body function or condition. In the example of a heart monitor implementing the sensor 8, the display can be a waveform corresponding to the detected heart beat energy. This display can be by way of a computer monitor or printer or other display device.

[0040] The method of the present invention can also be defined as comprising distributing by any suitable means (e.g., through direct mail, at physicians' offices) at least one sensor to each patient. For example, this includes distributing to a plurality of patients respective portable heart monitoring devices that the respective patients can hold against their chests without assistance. Each sensor can be marked with indicia defining the address of the Internet site of the log-on site 6a. In the preferred embodiments of FIGS. 3 and 4, this can include the World Wide Web address for the respective log-on site 6a that one or more of the patients is to access to download the medical data acquisition and transmission program.

[0041] The method further comprises maintaining the medical data acquisition and transmission program at the Internet site which is accessible by the patients such that the patients can use the medical data acquisition and transmission program to control their respective computers to receive and process signals from the patients' respective sensors and to transmit medical data onto the Internet in response. Use of the program is made by accessing the Internet site using a web browser program or other suitable programming loaded on the computer at the location of the patient.

[0042] Maintaining the medical data acquisition and transmission program includes storing the program in a computer

at the Internet site 6a for FIGS. 3 and 4. The method can further comprise storing in the computer at the Internet site a database of potential recipients of the medical data, wherein the database is also accessible by each patient such that each patient can identify from the potential recipients at least one selected recipient to receive the medical data for that patient.

[0043] One way of getting the information to the end users is to distribute to a plurality of physicians receivers for receiving at least portions of the medical data transmitted over the Internet. The receivers (which may also have transmitting capability can include pagers connected to palmtop computers such as referred to in the patents mentioned in the background portion of this specification and incorporated herein by reference.

[0044] A more detailed explanation of the foregoing will next be given with reference to FIGS. 7-10.

[0045] FIG. 7 shows a particular implementation of the components generally identified in FIG. 5. This includes the sensor 8, the microphone 14 and the personal computer 16. The sensor 8 provides an acoustical signal in simultaneous response to a biological function or condition, namely a beating heart in the illustrated implementation. The microphone 14 is located with the sensor 8 to receive the acoustical signal as the signal is provided from the sensor 8. The personal computer 16 is connected to the microphone and is adapted to transfer over the Internet, in real-time response to the microphone receiving the acoustical signal, electric signals representative of the received acoustical signal.

[0046] The sensor 8 of the FIG. 7 implementation includes a device 30 that converts the beating heart energy into electric signals. This can be by conventional electrocardiogram (ECG) electrodes as represented in FIG. 7 or other suitable device (e.g., a suitable piezoelectric device). The electric signals generated by the device 30 are amplified through an amplifier 32. The output of the amplifier 32 is processed through a voltage to frequency converter 34 to provide an alternating current signal that drives a speaker 36 to emit the acoustical signal directed toward the microphone 14.

[0047] The microphone 14 converts the acoustical signal into an analog electric signal conducted through a connected microphone jack 38 of the personal computer 16. The microphone jack 38 connects to the input of an analog to digital converter circuit 40 in the personal computer 16. The circuit 40 interfaces the microphone to the computer by converting the analog signal from the microphone 14 to a digital signal that can be used by a microprocessor circuit 42. The microprocessor circuit 42 is connected to a mouse 44, a keyboard 46, and a display 48. The microprocessor circuit 42 also communicates with an Internet connection 50, such as a coupling to a telephone line (e.g., a modem).

[0048] The analog to digital converter circuit 40 preferably provides at least eight-bit resolution and samples at 8,000 or more samples per second; this can be implemented with conventional technology, one example of which is a SOUND BLASTER brand sound card from Creative Labs. The microprocessor circuit 42 is also conventional, but preferably has a microprocessor nominally rated at least 20 megahertz. It also includes suitable memory for program storage and processing functions (e.g., typically both read only memory and random access memory). The programming of such memory is also conventional prior to loading the medical data acquisition and transmission program referred to above. This conventional programming can include, for example, a Windows operating system and a compatible web browser, such as Microsoft's INTERNET EXPLORER program or Netscape's NAVIGATOR program. Thus, the foregoing, and the other components illustrated in FIG. 7, can be implemented with conventional devices.

[0049] With the equipment of FIG. 7, shown in simplified blocks in FIG. 8, a patient at the site of the sensor 8, microphone 14 and personal computer 16 uses the web browser stored in the personal computer 16 to log onto the Internet. Typically this occurs through whatever Internet service provider the patient uses. Non-limiting examples include America On-Line and AT&T Worldnet. Once logged onto the Internet through his or her Internet service provider, the patient types in the address of the Internet site containing the medical data acquisition and transmission program if it is not available at the Internet service provider site. For example, for the illustration in FIG. 8, the patient enters the address "http//www.datacritical.com". The home page of datacritical.com appears on the display 48 of the personal computer 16. Through this home page, the patient downloads or otherwise accesses the medical data acquisition and transmission program. In one example, the datacritical.com home page contains a link that the patient selects to go to a page from which the program is downloaded. This linked page includes patient identification entry spaces, such as calling for the patient's unique password, to enable only a registered patient to download the program if such registry is desired to permit someone to download the program.

[0050] An example of a medical data acquisition and transmission program that is downloaded to the patient's computer 16 is appended at the end of this specification. The program can be of any suitable language and type to enable the implementation of the present invention; however, non-limiting examples include a program in ActiveX, or Java, or a plug-in module. Preferably the program is downloaded in an active mode so that the patient does not need to do anything to actuate the program to receive the medical data and to process it for display locally or transmission onto the Internet.

[0051] With the downloaded program running, an acoustical signal is generated at the patient site in response to the biological function or condition of the individual being monitored. This includes the heart monitor for the illustration of FIGS. 7 and 8. The acoustical signal is received in the microphone 14 near the individual. The microphone 14 converts the acoustical signal into an analog electric signal. The analog electric signal is communicated from the microphone 14 to the analog to digital conversion circuit 40 where it is converted to a digital electric signal. The digital electric signal is

processed in the microprocessor circuit 42 of the personal computer 16 under control of the medical data acquisition and transmission program that has been downloaded and is actively running. The program can then display to the individual, through the display 48 of the personal computer 16, a representation of the individual's biological function or condition in response to which the acoustical signal was generated. The medical data acquisition and transmission program can also transmit the data onto the Internet. In the illustration of FIG. 8, the data is transmitted to datacritical.com where it can be stored in a searchable database and transmitted to an end user. Preferably the transmission to the end user occurs contemporaneously with the foregoing steps so that there is real-time transmission of the data from the time the biological function or condition of the individual was sensed to the time it is received by the end user. Thus, once the program has been downloaded into the personal computer 16, preferably the subsequent steps from sensing the condition through display to the individual at the patient's site or transmission to an end user are performed together in real time. The data downloaded to the end user can also be stored in the end user's computer for later retrieval. Thus, the medical information derived from the sensing activity can be useful both for clinical purposes by an end user medical care provider as well as for providing self-knowledge benefits for the patient, such as in monitoring his or her own biological functions or conditions.

[0052] A flow diagram of a preferred embodiment of the medical data acquisition and transmission program downloaded to the personal computer 16 is illustrated in FIG. 9, and the source code is listed at the end of this specification before the claims. The program is an ActiveX control in combination with an HTML page which collects, filters and displays a user's ECG waveform and heart rate. When the control is embedded in a web page (e.g., at datacritical.com in the FIG. 8 illustration), it is downloaded to the user's personal computer 16 and configured to record ECG data from the microphone 14 connected to the personal computer 16 and to display the ECG waveform in the browser window of the personal computer 16. Configuration of the control is achieved via HTML "Param" tags. The control is embedded in an HTML page via the "Object" tag-this tells the browser to load the control. After the "Object" tag, and the name of the control, come the "Param" tags which provide values for named parameters of the control. For example, in the appended program listing there is a parameter called "Microphone_Not_Server," and the Param statement looks like:

<PARAM NAME = "Microphone_Not_Server" VALUE = "1">

[0053] This statement initializes the control to listen for data at the microphone.

[0054] The browser-loaded control can also stream, across the Internet, demodulated ECG data to a redisplay server at a recipient site if one is available. The redisplay server also contains a control program (see FIG. 10 and program listing) that processes the forwarded ECG data.

[0055] When the program of FIG. 9 operates in the personal computer 16 at a patient's site, the data source in this implementation referring to FIGS. 7-10 is an 8 kilohertz pulse code modulated (PCM) ECG signal received from the analog to digital converter circuit 40 in response to the acoustical signal sensed by the microphone 14. When the control is contained in a redisplay server, the data source is demodulated ECG data that has been sent from the personal computer 16 over the Internet to the redisplay server (or from the personal computer to datacritical.com and then to the redisplay server via the Internet or a paging network, for example).

[0056] In the flow diagram of FIG. 9, once the control has been downloaded to the user's personal computer, it is configured via an HTML "Param" tag to collect data from the microphone as input through the analog to digital converter circuit 40 as explained above. The control may also be configured to connect to a redisplay server via another "Param" tag in the manner described above. The control then receives a "draw yourself' event from Windows. The first time this event is received, the control starts listening for data on the microphone.

[0057] Upon receiving data from the microphone, the control demodulates the data from 8 kilohertz pulse code modulated to 200 hertz ECG samples. If a redisplay server is specified, the control streams the new, demodulated samples to the server over the Internet.

[0058] Once the ECG data has been demodulated, it is filtered based upon the filtering level set in a "Param" tag on the web page. Filtering levels are as listed in FIG. 9 (low pass filter, high pass filter, derivative filter, square filter).

[0059] The most recent two seconds worth of filtered data is then scanned for QRS complexes to determine the user's heart rate in beats per minute. The software fires a Windows "my view has changed" event, which causes Windows to call the control's "OnDraw" routine such that "OnDraw" updates the screen 48 of the personal computer 16 with the latest ECG waveform and heart rate.

[0060] Referring to FIG. 10, in a redisplay server, the program enables the server to listen at a known port for incoming demodulated ECG samples from a remote site. When samples are received, the server passes them to the control's filtering routine as described above. The same chain of events then occurs: the samples are filtered, QRS detection is executed, and the waveform is updated to the remote server's window.

[0061] Thus, the present invention is well adapted to carry out the objects and attain the ends and advantages mentioned above as well as those inherent therein. While preferred embodiments of the invention have been described for the purpose of this disclosure, changes in the construction and arrangement of parts and the performance of steps can be made by those skilled in the art, which changes are encompassed within the scope of this invention as defined by the appended claims.

```
            ++mCurrentAverage;

            if( mCurrentAverage >= mAverageCount )
                mCurrentAverage = 0;
        }
        else
            ++mCumuSamps;

            //----------------------------------------
            // Average BPM Calculation
            //----------------------------------------
        int     total = 0;

        for( i = 0; i < mAverageCount; i++ )
            total += *(mpBPMs + i);

        mAvgBPM = total / mAverageCount;

        if( !mAvgBPM )
            mAvgBPM = mCurBPM;

            //----------------------------------------
            // update the "current samp" index
            // NOTE: works like a ring buffer
            //----------------------------------------
        ++mCurrentSamp;

        if( mCurrentSamp >= mNumSamples )
            mCurrentSamp = 0;

        mPreviousPoint = newPoint;

        return mCurBPM;

    }   // end of Update
```

```
// WebEcg.h : Declaration of the CWebEcg

#ifndef __WEBECG_H_
#define __WEBECG_H_

#include "resource.h"       // main symbols
#include "network.h"
#include "BPMDetector.h"
#include "ringbuf.h"


const unsigned long SAMPLE_FREQ         = 8000;
const unsigned long DECIMATION          = 40;       // 40 to 1 decimation
const unsigned long ECG_FREQ            = (SAMPLE_FREQ / DECIMATION);
const unsigned long FRAMES_PER_SEC      = 20;
const unsigned long WORK_BUFFER_SIZE    = (SAMPLE_FREQ / FRAMES_PER_SEC);
//#define DISPLAY_SECONDS       5


/////////////////////////////////////////////////////////////////////////////
// CWebEcg
class ATL_NO_VTABLE CWebEcg :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CWebEcg, &CLSID_WebEcg>,
    public CComControl<CWebEcg>,
    public CStockPropImpl<CWebEcg, IWebEcg, &IID_IWebEcg, &LIBID_WEBECGCONTROLLi
b>,
    public IProvideClassInfo2Impl<&CLSID_WebEcg, NULL, &LIBID_WEBECGCONTROLLib>,
    public IPersistPropertyBagImpl<CWebEcg>,
    public IPersistStreamInitImpl<CWebEcg>,
    public IPersistStorageImpl<CWebEcg>,
    public IQuickActivateImpl<CWebEcg>,
    public IOleControlImpl<CWebEcg>,
    public IOleObjectImpl<CWebEcg>,
    public IOleInPlaceActiveObjectImpl<CWebEcg>,
    public IViewObjectExImpl<CWebEcg>,
    public IOleInPlaceObjectWindowlessImpl<CWebEcg>,
    public IDataObjectImpl<CWebEcg>,
    public ISupportErrorInfo,
    public IConnectionPointContainerImpl<CWebEcg>,
    public IPropertyNotifySinkCP<CWebEcg>,
    public ISpecifyPropertyPagesImpl<CWebEcg>
{
public:
    CWebEcg()
        : mBPM( 200,    // samps per sec
                4,      // maxQRSPerSec
                2,      // secs to buffer
                3,      // secs to average
                40.0 )// detect thresh
    {
//      MessageBox("WebECG Starting", "Informational message...", MB_OK);
        m_bMicrophoneNotServer = false;

        mbDrawECG  = false;
        m_bFrozen  = false;

        m_bWindowOnly = TRUE;

        m_nVolumeThreshold = 8;
        m_nBPMFontSize = 36;
        m_dDetectPercent = 50.0;
```

```
m_nBPM = m_nAvgBPM = 0;
mAge   = 30;
mMaxHR = 220 - mAge;

m_nForeWidth = 2;
m_clrForeColor = RGB(0,0,255);
m_clrBackColor = RGB(0,0,0);
m_clrGridColor = RGB(0,192,0);
m_dwCardioGramLength = ECG_FREQ*2;
m_pCardioGram = new (signed char[m_dwCardioGramLength]);
m_pCardioGramPoints = new(POINT[m_dwCardioGramLength]);
m_pVerticalBar = new (unsigned char[m_dwCardioGramLength]);
for (unsigned long i = 0; i<m_dwCardioGramLength; i++)
{
    m_pCardioGram[i] = 0;
    m_pVerticalBar[i] = 0;
}

m_dwStartDrawing = 0;
m_dwStopDrawing  = 0;

m_bRecording = false;

mBPMSampleCount = 0;

mSampsPerBPMPt  = ECG_FREQ / 4;
mCumSamps       = 0;

for( i = 0; i < 4; i++ )
{
    m_pWaveHdr[i] = new(WAVEHDR);
    m_pWavBuf[i]  = NULL;
}

m_dMaxmV =  2.0;
m_dMinmV = -2.0;
m_dStepmV = 0.5;

m_hbmpWorkBitmap = NULL;
m_hdcWorkDC = NULL;

WSAStartup(0x0101, &m_wsaWSAData); // 0x0101 means Winsock 1.1
m_bConnectedToServer = false;

mMaxPoints      = 0;
mPointWidth     = 4;
mMinPointWidth  = 4;
mSpaceWidth     = 2;
mCurPoint       = 0;
mPointsRead     = 0;
mPointsPerCyc   = 0;
mDrawingWid     = 0;

mXscale    = 0.0;
mYscale    = 0.0;
mXoffset   = 0.0;
mYoffset   = 0.0;
mHeight    = 0;
mWidth     = 0;
mPrevWidth = -1;
mYmax      = 0;
mMaxHrY    = 0;
mMinHrY    = 0;
```

```
    }   // end constructor

    ~CWebEcg()
    {
        int i;

        waveInReset(m_hWaveIn);
        waveInClose(m_hWaveIn);

//      Sleep((1.0/SAMPLE_FREQ)*(2*WORK_BUFFER_SIZE)*1000); // Give system time
to shut down whatever.
//      Sleep((1000/SAMPLE_FREQ)*(2*WORK_BUFFER_SIZE)); // Give system time to s
hut down whatever.
//      Sleep((1000*2*WORK_BUFFER_SIZE)/SAMPLE_FREQ);   // Give system time to s
hut down whatever.
        Sleep((2000*WORK_BUFFER_SIZE)/SAMPLE_FREQ); // Give system time to shut
down whatever.

        for (i=0; i<4; i++)
        {
            waveInUnprepareHeader(m_hWaveIn, m_pWaveHdr[i], sizeof(WAVEHDR));
            delete m_pWaveHdr[i];
            if( m_pWavBuf[i] )
                delete m_pWavBuf[i];
        }

        delete m_pVerticalBar;
        delete m_pCardioGramPoints;
        delete m_pCardioGram;

        if (m_hbmpWorkBitmap)
            DeleteObject(m_hbmpWorkBitmap);
        if (m_hdcWorkDC)
            DeleteObject(m_hdcWorkDC);


        shutdown(m_sckLocalSocket, 1);
        closesocket(m_sckLocalSocket);
        WSACleanup();
//      MessageBox("WebECG Stopping", "Informational message...", MB_OK);
    }

DECLARE_REGISTRY_RESOURCEID(IDR_WEBECG)

BEGIN_COM_MAP(CWebEcg)
    COM_INTERFACE_ENTRY(IWebEcg)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY_IMPL(IViewObjectEx)
    COM_INTERFACE_ENTRY_IMPL_IID(IID_IViewObject2, IViewObjectEx)
    COM_INTERFACE_ENTRY_IMPL_IID(IID_IViewObject, IViewObjectEx)
    COM_INTERFACE_ENTRY_IMPL(IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY_IMPL_IID(IID_IOleInPlaceObject, IOleInPlaceObjectWindowl
ess)
    COM_INTERFACE_ENTRY_IMPL_IID(IID_IOleWindow, IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY_IMPL(IOleInPlaceActiveObject)
    COM_INTERFACE_ENTRY_IMPL(IOleControl)
    COM_INTERFACE_ENTRY_IMPL(IOleObject)
    COM_INTERFACE_ENTRY_IMPL(IQuickActivate)
    COM_INTERFACE_ENTRY_IMPL(IPersistPropertyBag)
    COM_INTERFACE_ENTRY_IMPL(IPersistStorage)
    COM_INTERFACE_ENTRY_IMPL(IPersistStreamInit)
    COM_INTERFACE_ENTRY_IMPL(ISpecifyPropertyPages)
```

```
    COM_INTERFACE_ENTRY_IMPL(IDataObject)
    COM_INTERFACE_ENTRY(IProvideClassInfo)
    COM_INTERFACE_ENTRY(IProvideClassInfo2)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
END_COM_MAP()

BEGIN_PROPERTY_MAP(CWebEcg)
    // Example entries
    // PROP_ENTRY("Property Description", dispid, clsid)
    PROP_ENTRY("Fore_Color",        DISPID_FORECOLOR,       CLSID_StockColorPage)
    PROP_ENTRY("Back_Color",        DISPID_BACKCOLOR,       CLSID_StockColorPage)
    PROP_ENTRY("Grid_Color",        dispidGridColor,        CLSID_WebEcgPPG)
    PROP_ENTRY("Grid_Style",        dispidGridStyle,        CLSID_WebEcgPPG)
    PROP_ENTRY("Max_mV",            dispidMaxmV,            CLSID_WebEcgPPG)
    PROP_ENTRY("Min_mV",            dispidMinmV,            CLSID_WebEcgPPG)
    PROP_ENTRY("Step_mV",           dispidStepmV,           CLSID_WebEcgPPG)
    PROP_ENTRY("Fore_Width",        dispidForeWidth,        CLSID_WebEcgPPG)
    PROP_ENTRY("Display_Seconds",   dispidDisplaySeconds,   CLSID_WebEcgPPG)
    PROP_ENTRY("Volume_Threshold",  dispidVolumeThreshold,  CLSID_WebEcgPPG)
    PROP_ENTRY("Filtering",         dispidFiltering,        CLSID_WebEcgPPG)
    PROP_ENTRY("BPM_Font_Size",     dispidBPMFontSize,      CLSID_WebEcgPPG)
    PROP_ENTRY("Detect_Percent",    dispidDetectPercent,    CLSID_WebEcgPPG)
    PROP_ENTRY("Server_Address",    dispidServerAddress,    CLSID_WebEcgPPG)
    PROP_ENTRY("Microphone_Not_Server", dispidMicrophoneNotServer, CLSID_WebEcgP
PG)
    PROP_ENTRY("AddNewPoint",       dispidAddNewPoint,      CLSID_WebEcgPPG)
    PROP_ENTRY("Age",               dispidAge,              CLSID_WebEcgPPG)
    PROP_ENTRY("Average_Count",     dispidAverageCount,     CLSID_WebEcgPPG)
    PROP_ENTRY("ViewECG",           dispidViewECG,          CLSID_WebEcgPPG)
END_PROPERTY_MAP()

BEGIN_CONNECTION_POINT_MAP(CWebEcg)
    CONNECTION_POINT_ENTRY(IID_IPropertyNotifySink)
END_CONNECTION_POINT_MAP()

BEGIN_MSG_MAP(CWebEcg)
    MESSAGE_HANDLER(WM_PAINT,       OnPaint)
    MESSAGE_HANDLER(WM_SETFOCUS,    OnSetFocus)
    MESSAGE_HANDLER(WM_KILLFOCUS,   OnKillFocus)
    MESSAGE_HANDLER(MM_WIM_OPEN,    OnWimOpen)
    MESSAGE_HANDLER(MM_WIM_DATA,    OnWimData)
    MESSAGE_HANDLER(MM_WIM_CLOSE,   OnWimClose)
    MESSAGE_HANDLER(MM_WOM_OPEN,    OnWomOpen)
    MESSAGE_HANDLER(MM_WOM_DONE,    OnWomDone)
    MESSAGE_HANDLER(MM_WOM_CLOSE,   OnWomClose)
    MESSAGE_HANDLER(WM_LBUTTONDOWN, OnLButtonDown)
END_MSG_MAP()

// ISupportsErrorInfo
    STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

// IViewObjectEx
    STDMETHOD(GetViewStatus)(DWORD* pdwStatus)
    {
        ATLTRACE(_T("IViewObjectExImpl::GetViewStatus\n"));
        *pdwStatus = VIEWSTATUS_SOLIDBKGND | VIEWSTATUS_OPAQUE;
        return S_OK;
    }
```

```
// IwebEcg
public:
    STDMETHOD(get_ViewECG)(/*[out, retval]*/ BOOL *pVal);
    STDMETHOD(put_ViewECG)(/*[in]*/ BOOL newVal);
    STDMETHOD(get_Average_Count)(/*[out, retval]*/ long *pVal);
    STDMETHOD(put_Average_Count)(/*[in]*/ long newVal);
    STDMETHOD(get_Age)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_Age)(/*[in]*/ short newVal);
    STDMETHOD(get_AddNewPoint)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_AddNewPoint)(/*[in]*/ short newVal);
    STDMETHOD(get_MicrophoneNotServer)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_MicrophoneNotServer)(/*[in]*/ short newVal);
    STDMETHOD(get_ServerAddress)(/*[out, retval]*/ BSTR *pVal);
    STDMETHOD(put_ServerAddress)(/*[in]*/ BSTR newVal);
    STDMETHOD(get_DetectPercent)(/*[out, retval]*/ double *pVal);
    STDMETHOD(put_DetectPercent)(/*[in]*/ double newVal);
    STDMETHOD(get_BPMFontSize)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_BPMFontSize)(/*[in]*/ short newVal);
    STDMETHOD(get_Filtering)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_Filtering)(/*[in]*/ short newVal);
    STDMETHOD(get_VolumeThreshold)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_VolumeThreshold)(/*[in]*/ short newVal);
    STDMETHOD(get_DisplaySeconds)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_DisplaySeconds)(/*[in]*/ short newVal);
    STDMETHOD(get_Forewidth)(/*[out, retval]*/ long *pVal);
    STDMETHOD(put_Forewidth)(/*[in]*/ long newVal);
    STDMETHOD(get_StepmV)(/*[out, retval]*/ double *pVal);
    STDMETHOD(put_StepmV)(/*[in]*/ double newVal);
    STDMETHOD(get_MaxmV)(/*[out, retval]*/ double *pVal);
    STDMETHOD(put_MaxmV)(/*[in]*/ double newVal);
    STDMETHOD(get_MinmV)(/*[out, retval]*/ double *pVal);
    STDMETHOD(put_MinmV)(/*[in]*/ double newVal);
    STDMETHOD(get_GridStyle)(/*[out, retval]*/ long *pVal);
    STDMETHOD(put_GridStyle)(/*[in]*/ long newVal);
    STDMETHOD(get_GridColor)(/*[out, retval]*/ OLE_COLOR *pVal);
    STDMETHOD(put_GridColor)(/*[in]*/ OLE_COLOR newVal);
    HRESULT FireViewChange();
    HRESULT OnDraw(ATL_DRAWINFO& di);
    HRESULT drawECG( ATL_DRAWINFO& di );
    HRESULT drawBPMPoints( ATL_DRAWINFO& di );
    HRESULT OnWimOpen(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    HRESULT OnWimData(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    HRESULT OnWimClose(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    HRESULT OnWomOpen(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    HRESULT OnWomDone(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    HRESULT OnWomClose(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled);
    LRESULT OnLButtonDown(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandle
d);

    CComPtr<IPictureDisp> m_pMouseIcon;
    OLE_COLOR m_clrBackColor;
    OLE_COLOR m_clrBorderColor;
    OLE_COLOR m_clrForeColor;
    OLE_COLOR m_clrGridColor;
    BOOL m_bBorderVisible;
    long m_nBorderStyle;
    long m_nBorderwidth;
    long m_nForewidth;
    long m_nGridStyle;
    long m_nMousePointer;
    short m_nVolumeThreshold;
    short m_nFiltering;
    short m_nBPMFontSize;
```

```
        CComBSTR m_bstrFileName;
        CComBSTR m_bstrServerAddress;
        bool m_bConnectedToServer;
        double m_dMinmV;
        double m_dMaxmV;
        double m_dStepmV;
        double m_dDetectPercent;

        // Shouldn't be public, but I don't care about nice now as
        // much as getting something worki
        BOOL            m_bMicrophoneNotServer;
        signed char *   m_pCardioGram;
        unsigned char * m_pVerticalBar;
        POINT *         m_pCardioGramPoints;
        unsigned long   m_dwCardioGramLength;
        bool            m_bRecording;
        unsigned char * m_pWavBuf[4];
        unsigned char * m_pFullWav;
        unsigned long   m_dwFullWavLength;
        HWAVEIN         m_hWaveIn;
        PWAVEHDR        m_pWaveHdr[4];
        unsigned short  m_nBPM;
        unsigned short  m_nAvgBPM;
        unsigned long   m_dwStartDrawing;
        unsigned long   m_dwStopDrawing;
        HBITMAP         m_hbmpWorkBitmap;
        HDC             m_hdcWorkDC;
        bool            m_bFrozen;
        RECT            m_rctFreezeButton;
        SOCKET          m_sckLocalSocket;
        sockaddr_in     m_sckadrRemoteSocketAddress;
        WSADATA         m_wsaWSAData;

private:
    void NewRawData(signed short * RawData, unsigned long NumData);
    void ConnectToServer();

    void drawPoints( HDC &dc,    RECT &rc,  int dataStart,
                     int xStart, int  xEnd, int numBars  );
    void upArrow( POINT *pPoints, int x, int y, int wid );
    void downArrow( POINT *pPoints, int x, int y, int wid );

    CBPMDetector    mBPM;
    bool            mbDrawECG;

    short           mAge;
    short           mMaxHR;

    short           mBPMSampleCount;
    ringbuf < int > mBPMData;

    short           mSampsPerBPMPt;
    short           mCumSamps;

        // Members for drawing point graph
    int             mMaxPoints;     // # points which can be drawn
    int             mPointWidth;    // point width
    int             mMinPointWidth; // minimum point width
    int             mSpaceWidth;    // width of space between points
    int             mCurPoint;      // current point #
    int             mPointsRead;    // total points read in
    int             mPointsPerCyc;  // total points to read before wrap
```

```
    int             mDrawingWid;    // width of actual drawing area

        // General drawing vars
    double          mXscale;
    double          mYscale;
    double          mYscale_mV;
    double          mYscale_Cnts;
    double          mXoffset;
    double          mYoffset;
    int             mHeight;
    int             mWidth;
    int             mPrevWidth;
    int             mYmax;
    long            mMaxHrY;
    long            mMinHrY;

};  // end of class CWebEcg

#endif //__WEBECG_H_
```

```
// webEcg.cpp : Implementation of CWebEcg
#include "stdafx.h"
#include "webEcgControl.h"
#include "webEcg.h"

#include "stdio.h"
#include "DemodECG.h"
#include "QRS.h"


//-----------------------------------------------------------------------------
//  Constants
//-----------------------------------------------------------------------------
#define FREEZE_HEIGHT 30

const COLORREF cnYello = RGB( 255, 255,   0 );
const COLORREF cnRed   = RGB( 255,   0,   0 );
const COLORREF cnGreen = RGB(   0, 255,   0 );
const COLORREF cnPurp  = RGB( 200,   0, 200 );
const COLORREF cnBlue  = RGB(   0,   0, 255 );
const COLORREF cnLtBlu = RGB(  50, 100, 255 );
const COLORREF cnBlack = RGB(   0,   0,   0 );


//-----------------------------------------------------------------------------
//  Function Prototypes
//-----------------------------------------------------------------------------
inline void DrawLine(HDC hdc, long x1, long y1, long x2, long y2);
void DrawString( HDC      dc,
                 HFONT    font,
                 UINT     align,
                 COLORREF col,
                 int      left,
                 int      top,
                 char     *str );


//-----------------------------------------------------------------------------
//  InterfaceSupportsErrorInfo
//-----------------------------------------------------------------------------
STDMETHODIMP CWebEcg::InterfaceSupportsErrorInfo( REFIID riid )
{
    static const IID* arr[] =
    {
        &IID_IWebEcg,
    };
    for (int i=0;i<sizeof(arr)/sizeof(arr[0]);i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }
    return S_FALSE;
}

HRESULT CWebEcg::FireViewChange()
{
    if (m_bInPlaceActive)
    {
        // Active
        if (m_hWndCD != NULL)
            return (::InvalidateRect(m_hWndCD, NULL, FALSE)) ? S_OK : E_FAIL; //
Window based
```

```
            if (m_spInPlaceSite != NULL)
                return m_spInPlaceSite->InvalidateRect(NULL, FALSE); // Windowless
        }
        // Inactive
        SendOnViewChange(DVASPECT_CONTENT);
        return S_OK;
    }

    LRESULT CWebEcg::OnLButtonDown(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL & b
    Handled)
    {
        POINTS ClickPos;

        ClickPos = MAKEPOINTS(lParam);
        if ( (ClickPos.y >= m_rctFreezeButton.top) && (ClickPos.y <= m_rctFreezeButt
    on.bottom) &&
             (ClickPos.x >= m_rctFreezeButton.left) && (ClickPos.x <= m_rctFreezeButt
    on.right) )
        {
            m_bFrozen = !m_bFrozen;
            //      Fire_Click();
            FireViewChange();
        }
        return 0;
    }


    HRESULT CWebEcg::OnDraw( ATL_DRAWINFO& di )
    {
        if( mbDrawECG )
            return drawECG( di );
        else
            return drawBPMPoints( di );

    }   // end of OnDraw


    HRESULT CWebEcg::drawECG( ATL_DRAWINFO& di )
    {
        unsigned long   i;
        double          y;
        long            yPos;
        long            xPos;
        long            StartXPos[2];
        long            StopXPos[2];
        long            LeftEdge;
        static long     NextStartPos=0;
        bool            Wrapped;
        char            buf[80];
        HRESULT         Result;
        double          WorkMin;
        double          WorkMax;
        double          OverShoot;
        static SIZE     BPMsize;
        static ATL_DRAWINFO PrevDI;


        /*
        // Handy little trick for very simple debugging output.
        // Writes whatever you want directly to the screen...
        HDC ScreenDC;
        ScreenDC = ::GetDC(0);
        sprintf(buf,"%d - %d                                    ",m_dwStartDrawing, m_dwStopDra
```

```
wing);
    SetBkMode(ScreenDC,OPAQUE);
    TextOut(ScreenDC,0,0,_T(buf),strlen(buf));
    ReleaseDC(ScreenDC);
    */

    if ((!m_bRecording) && (m_hWndCD) && (m_bMicrophoneNotServer))
    {
        m_bRecording = true;

        WAVEFORMATEX    waveformat;

        // Set up wave recording.
        waveformat.wFormatTag       = WAVE_FORMAT_PCM;
        waveformat.nChannels        = 1;
        waveformat.nSamplesPerSec   = SAMPLE_FREQ;
        waveformat.nAvgBytesPerSec  = SAMPLE_FREQ;
        waveformat.nBlockAlign      = 1;
        waveformat.wBitsPerSample   = 8;
        waveformat.cbSize           = 0;

        Result = waveInOpen (&m_hWaveIn, WAVE_MAPPER, &waveformat,
            (DWORD) m_hWndCD, 0, CALLBACK_WINDOW);
    }

    if (m_dMaxmV > m_dMinmV)
    {
        WorkMax = m_dMaxmV;
        WorkMin = m_dMinmV;
    }
    else if (m_dMaxmV == m_dMinmV)
    {
        WorkMax = m_dMaxmV + 0.5;
        WorkMin = m_dMaxmV - 0.5;
    }
    else
    {
        WorkMax = m_dMinmV;
        WorkMin = m_dMaxmV;
    }
    OverShoot = (WorkMax - WorkMin)/20.0;

    RECT& rc = *(RECT*)di.prcBounds;

    if (m_pCardioGramPoints == NULL)
    {
        m_pCardioGramPoints = new(POINT[m_dwCardioGramLength]);
        mPrevWidth = -1; // Force following check to reload x positions
    }

    if (mWidth != mPrevWidth)
    {
        mHeight = rc.bottom - rc.top + 1;
        // Make room for Freeze/Unfreeze button
        mHeight -= FREEZE_HEIGHT;
        mWidth = rc.right - rc.left + 1;
        // Multiply original value by these
        mXscale = (double)mWidth/(double)m_dwCardioGramLength;
        mYscale_mV = -(mHeight)/(WorkMax - WorkMin + (2*OverShoot));  // +/- 2mV
        // Set mYscale_Cnts to counts, not mV
        mYscale_Cnts = mYscale_mV / 40.0;
        // Add these to the result
        mXoffset = rc.left;
```

```
        //mYoffset = rc.top + mHeight/2;
        // Algebraically derived.  Don't ask me to explain it...  In my logbook
1-29-1999
        // (Actually, I have "T + .2*Scale" there, but since scale is negative,
we have to
        //  subtract instead of add)
        mYoffset = (rc.top - (OverShoot*mYscale_mV)) - WorkMax * (-(rc.bottom -
rc.top + 1)/(WorkMax - WorkMin + (2*OverShoot)));

        // Recalculate X positions of all of the Cardiogram points.
        for (i=0; i<m_dwCardioGramLength; i++)
        {
            m_pCardioGramPoints[i].x = (long)(i*mXscale+mXoffset);
        }
    }
    mPrevWidth = mWidth;

    HDC WorkDC;
    HBITMAP WorkBitmap;
    HBITMAP OrigWorkBitmap;

    // Hold onto (and re-use) WorkDC and WorkBitmap if we can, but rebuild them
    // if anything in the DrawInfo changes.
    if (memcmp((void *)&di, (void *)&PrevDI, sizeof PrevDI) != 0)
    {
        DeleteObject(m_hdcWorkDC);
        DeleteObject(m_hbmpWorkBitmap);
        m_hdcWorkDC = CreateCompatibleDC(di.hdcDraw);
        m_hbmpWorkBitmap = CreateCompatibleBitmap(di.hdcDraw, mWidth, mHeight);
        // Something weird has happened - redraw the entire screen.
        m_dwStartDrawing = m_dwStopDrawing + 1;
        if (m_dwStartDrawing == m_dwCardioGramLength)
            m_dwStartDrawing = 0;
    }
    WorkDC = m_hdcWorkDC;
    WorkBitmap = m_hbmpWorkBitmap;
    memcpy((void *)&PrevDI, (void *)&di, sizeof PrevDI);
    OrigWorkBitmap = (HBITMAP)SelectObject(WorkDC, WorkBitmap);

    SetBkColor(WorkDC, m_clrBackColor);
    SetTextColor(WorkDC, m_clrGridColor);

    HPEN    OrigPen;

    HBRUSH      OrigBrush;
    LOGBRUSH    BrushDesc;
    BrushDesc.lbStyle = PS_SOLID;
    BrushDesc.lbColor = m_clrBackColor;
    OrigBrush = (HBRUSH)SelectObject(WorkDC, CreateBrushIndirect(&BrushDesc));
    OrigPen = (HPEN)SelectObject(WorkDC, CreatePen(PS_SOLID | PS_INSIDEFRAME, 0,
BrushDesc.lbColor));

    // StopXPos[0] is the most recent point's position
    if (m_dwStopDrawing > m_dwStartDrawing)
    {
        Wrapped = false;
        StartXPos[0] = NextStartPos;
        StopXPos[0]  = (long)(m_dwStopDrawing * mXscale + mXoffset);
        Rectangle(WorkDC, StartXPos[0]-1, rc.top,
            StopXPos[0]+1,  rc.bottom - FREEZE_HEIGHT);
        // Grab the last vertical line of the previous section so we can merge w
ith it.
        LeftEdge = (StartXPos[0]?StartXPos[0]-1:0);
```

```
            BitBlt(WorkDC,      LeftEdge, rc.top, 1, mHeight,
                di.hdcDraw, LeftEdge, rc.top,
                SRCCOPY);
        }
        else
        {
            Wrapped = true;
            StartXPos[0] = rc.left;
            StopXPos[0]  = (long)(m_dwStopDrawing * mXscale + mXoffset);
            StartXPos[1] = NextStartPos;
            StopXPos[1]  = rc.right;
            Rectangle(WorkDC, StartXPos[0]-5, rc.top,
                StopXPos[0]+5,  rc.bottom - FREEZE_HEIGHT);
            Rectangle(WorkDC, StartXPos[1]-5, rc.top,
                StopXPos[1]+5,  rc.bottom - FREEZE_HEIGHT);
            // Grab the last vertical line of the previous section so we can merge w
ith it.
            LeftEdge = (StartXPos[1]?StartXPos[1]-1:0);
            BitBlt(WorkDC,      LeftEdge, rc.top, 1, mHeight,
                di.hdcDraw, LeftEdge, rc.top,
                SRCCOPY);
        }
        DeleteObject(SelectObject(WorkDC, OrigBrush));

        // Now draw the grid in GridColor
        double LowAxis;
        double HighAxis;
        LowAxis = WorkMin;
        HighAxis = WorkMax;
        while ((LowAxis - m_dStepmV) >= (WorkMin - OverShoot))
            LowAxis -= m_dStepmV;
        while ((HighAxis + m_dStepmV) <= (WorkMax + OverShoot))
            HighAxis += m_dStepmV;

        DeleteObject(SelectObject(WorkDC, CreatePen(m_nGridStyle, 0, m_clrGridColor)
));

        if (Wrapped)
        {
            // Horizontal lines
            for (y=LowAxis; y<=HighAxis; y+=m_dStepmV)
            {
                yPos = (long)(y*mYscale_mV+mYoffset);
                DrawLine(WorkDC, StartXPos[0], yPos, StopXPos[0]+1, yPos);
                DrawLine(WorkDC, StartXPos[1], yPos, StopXPos[1]+1, yPos);
            }
            // Vertical lines
            for (i=m_dwStartDrawing; i < m_dwCardioGramLength; i++)
            {
                if (m_pVerticalBar[i] & 1)
                {
                    xPos = (long)(i * mXscale + mXoffset);
                    DrawLine(WorkDC, xPos, rc.top, xPos, rc.bottom - FREEZE_HEIGHT);
                }
            }
            // Yes, that should be <=, not <, and the one above should be
            // <, not <=...
            for (i=0; i <= m_dwStopDrawing; i++)
            {
                if (m_pVerticalBar[i] & 1)
                {
                    xPos = (long)(i * mXscale + mXoffset);
                    DrawLine(WorkDC, xPos, rc.top, xPos, rc.bottom - FREEZE_HEIGHT);
```

```
                }
            }
            // OK, now draw the Cardiogram in ForeColor
            for (i=(m_dwStartDrawing?m_dwStartDrawing-1:0); i < m_dwCardioGramLength
; i++)
            {
                m_pCardioGramPoints[i].y = (long)(m_pCardioGram[i] * mYscale_Cnts +
mYoffset);
                m_pCardioGramPoints[i].x = (long)(i * mXscale + mXoffset);
            }
            m_pCardioGramPoints[m_dwCardioGramLength-1].x = rc.right;
            DeleteObject(SelectObject(WorkDC, CreatePen(PS_SOLID, m_nForeWidth, m_cl
rForeColor)));
            Polyline(WorkDC, &(m_pCardioGramPoints[m_dwStartDrawing]), m_dwCardioGra
mLength - m_dwStartDrawing);
            // OK, now draw the Cardiogram in ForeColor for the wrapped section
            for (i=0; i <= m_dwStopDrawing; i++)
            {
                m_pCardioGramPoints[i].y = (long)(m_pCardioGram[i] * mYscale_Cnts +
mYoffset);
                m_pCardioGramPoints[i].x = (long)(i * mXscale + mXoffset);
            }
            m_pCardioGramPoints[m_dwStopDrawing].x = StopXPos[0];
            Polyline(WorkDC, m_pCardioGramPoints, m_dwStopDrawing + 1);
        }
        else
        {
            // Horizontal lines
            for (y=LowAxis; y<=HighAxis; y+=m_dStepmV)
            {
                yPos = (long)(y*mYscale_mV+mYoffset);
                DrawLine(WorkDC, StartXPos[0], yPos, StopXPos[0]+1, yPos);
            }
            // Vertical lines
            for (i=m_dwStartDrawing; i <= m_dwStopDrawing; i++)
            {
                if (m_pVerticalBar[i] & 1)
                {
                    xPos = (long)(i * mXscale + mXoffset);
                    DrawLine(WorkDC, xPos, rc.top, xPos, rc.bottom - FREEZE_HEIGHT);
                }
            }
            // OK, now draw the Cardiogram in ForeColor
            for (i=(m_dwStartDrawing?m_dwStartDrawing-1:0); i <= m_dwStopDrawing; i+
+)
            {
                m_pCardioGramPoints[i].y = (long)(m_pCardioGram[i] * mYscale_Cnts +
mYoffset);
                m_pCardioGramPoints[i].x = (long)(i * mXscale + mXoffset);
            }
            m_pCardioGramPoints[(m_dwStartDrawing?m_dwStartDrawing-1:0)].x = StartXP
os[0]-1;
            m_pCardioGramPoints[m_dwStopDrawing].x = StopXPos[0];

            DeleteObject(SelectObject(WorkDC, CreatePen(PS_SOLID, m_nForeWidth, m_cl
rForeColor)));
            Polyline(WorkDC, &(m_pCardioGramPoints[(m_dwStartDrawing?m_dwStartDrawin
g-1:0)]), m_dwStopDrawing - (m_dwStartDrawing?m_dwStartDrawing-1:0) + 1);
            //      SetPixelV(WorkDC, m_pCardioGramPoints[m_dwStopDrawing].x,m_pCard
ioGramPoints[m_dwStopDrawing].y,m_clrForeColor);
        }
```

```
        // Now draw the grid labels, outlined in BackColor
        SetTextAlign(workDC, TA_LEFT);
        SetBkMode(workDC, TRANSPARENT);
        SetTextColor(workDC, m_clrBackColor);
        for (y=LowAxis; y<=HighAxis; y+=m_dStepmV)
        {
            yPos = (long)(y*mYscale_mV+mYoffset);
            sprintf(buf,"%5.2fmV",y);
            TextOut(workDC, rc.left+9, yPos, _T(buf),strlen(buf));
            TextOut(workDC, rc.left+11, yPos, _T(buf),strlen(buf));
            TextOut(workDC, rc.left+10, yPos-1, _T(buf),strlen(buf));
            TextOut(workDC, rc.left+10, yPos+1, _T(buf),strlen(buf));
        }
        SetBkMode(workDC, TRANSPARENT);
        SetTextColor(workDC, m_clrGridColor);
        for (y=LowAxis; y<=HighAxis; y+=m_dStepmV)
        {
            yPos = (long)(y*mYscale_mV+mYoffset);
            sprintf(buf,"%5.2fmV",y);
            TextOut(workDC, rc.left+10, yPos, _T(buf),strlen(buf));
        }

        // Copy the newly created region(s) to the screen.
        if (Wrapped)
        {
            BitBlt(di.hdcDraw,rc.left,rc.top,StopXPos[0] - rc.left + 1,mHeight,workD
C,rc.left,rc.top,SRCCOPY);
            BitBlt(di.hdcDraw,StartXPos[1],rc.top,rc.right - StartXPos[1] + 1,mHeigh
t,workDC,StartXPos[1],rc.top,SRCCOPY);
        }
        else
        {
            LeftEdge = (StartXPos[0]?StartXPos[0]-1:0);
            BitBlt(di.hdcDraw, LeftEdge, rc.top, StopXPos[0] - LeftEdge + 1, mHeight

                workDC,     LeftEdge, rc.top,
                SRCCOPY);
        }

        // Draw BPM in upper right
        HFONT OrigFont;
        OrigFont = (HFONT)SelectObject(workDC, CreateFont(-MulDiv(m_nBPMFontSize,Get
DeviceCaps(workDC,LOGPIXELSY),72),
            0/*mWidth*/,
            0/*escapement*/,
            0/*orientation*/,
            FW_NORMAL/*weight*/,
            FALSE/*Italic*/,
            FALSE/*Underline*/,
            FALSE/*StrikeOut*/,
            DEFAULT_CHARSET,
            OUT_TT_ONLY_PRECIS/*OutputPrecision*/,
            CLIP_DEFAULT_PRECIS,
            DEFAULT_QUALITY,
            DEFAULT_PITCH | FF_ROMAN,
            "Times New Roman"
            ));

        sprintf(buf, "                    %d BPM", m_nAvgBPM);
//      sprintf(buf, "                  %d  %d BPM", m_nBPM, m_nAvgBPM );

        SetBkMode(workDC, OPAQUE);
        SetTextAlign(workDC, TA_RIGHT);
```

```
    SetTextColor(WorkDC, m_clrBackColor);
    SetBkColor(WorkDC, m_clrGridColor);
    TextOut(WorkDC, rc.right, rc.top, _T(buf), strlen(buf));

    // Figure out how big the biggest reading we will ever take is.
    GetTextExtentPoint32(WorkDC, _T("999 BPM"), strlen("999 BPM"), &BPMsize);
//    GetTextExtentPoint32(WorkDC, _T("999  999 BPM"), strlen("999  999 BPM"), &
BPMsize);

    DeleteObject(SelectObject(WorkDC, OrigFont));
    SetBkMode(di.hdcDraw, TRANSPARENT);

    DeleteObject(SelectObject(WorkDC, OrigPen));

    BitBlt(di.hdcDraw, rc.right - BPMsize.cx + 1, rc.top, BPMsize.cx, BPMsize.cy

        WorkDC,     rc.right - BPMsize.cx + 1, rc.top,
        SRCCOPY);


    // Draw the sweeping line
    OrigPen = (HPEN)SelectObject(di.hdcDraw, CreatePen(PS_SOLID, 2, RGB(255,255,
255)));
    DrawLine(di.hdcDraw, StopXPos[0]+3, rc.top, StopXPos[0]+3, rc.bottom - FREEZ
E_HEIGHT);
    DeleteObject(SelectObject(di.hdcDraw, OrigPen));

    // Do not delete the returned Bitmap, since it is tracked as a member of thi
s object.
    SelectObject(WorkDC, OrigWorkBitmap);
    //  DeleteObject(WorkBitmap);   // HACKHACKHACK - Deleted elsewhere
    //  DeleteDC(WorkDC);           // HACKHACKHACK - Deleted elsewhere



    /*
    // Hack for drawing BPM numbers next to detection point.
    // Probably needs to be removed for final version.
    SetTextAlign(di.hdcDraw, TA_RIGHT);
    SetBkMode(di.hdcDraw, TRANSPARENT);
    for (i=m_dwStartDrawing; i <= m_dwStopDrawing; i++)
    {
    if (m_pVerticalBar[i] & 254)
    {
    xPos = (long)(i * mXscale + mXoffset);
    yPos = (long)(m_pCardioGram[i] * mYscale_Cnts + mYoffset);
    sprintf(buf,"%d BPM",(m_pVerticalBar[i]>>1) & 0x7F);
    SetTextColor(di.hdcDraw, m_clrBackColor);
    TextOut(di.hdcDraw, xPos+1, yPos, _T(buf),strlen(buf));
    TextOut(di.hdcDraw, xPos-1, yPos, _T(buf),strlen(buf));
    TextOut(di.hdcDraw, xPos, yPos+1, _T(buf),strlen(buf));
    TextOut(di.hdcDraw, xPos, yPos-1, _T(buf),strlen(buf));
    SetTextColor(di.hdcDraw, m_clrGridColor);
    TextOut(di.hdcDraw, xPos, yPos, _T(buf),strlen(buf));
    }
    }
    */


    if (m_bFrozen)
        strcpy(buf, "Unfreeze Display");
    else
```

```
            strcpy(buf, "Freeze Display");
        BrushDesc.lbStyle = PS_SOLID;
        BrushDesc.lbColor = m_clrBackColor;
        OrigBrush = (HBRUSH)SelectObject(di.hdcDraw, CreateBrushIndirect(&BrushDesc)
);
        OrigPen = (HPEN)SelectObject(di.hdcDraw, CreatePen(PS_SOLID | PS_INSIDEFRAME
, 2, m_clrGridColor));
        m_rctFreezeButton.left = rc.left;
        m_rctFreezeButton.right = rc.right;
        m_rctFreezeButton.top = rc.bottom - FREEZE_HEIGHT;
        m_rctFreezeButton.bottom = rc.bottom;
        Rectangle(di.hdcDraw, m_rctFreezeButton.left, m_rctFreezeButton.top,
            m_rctFreezeButton.right,  m_rctFreezeButton.bottom);
        SetTextColor(di.hdcDraw, m_clrGridColor);
        DrawText(di.hdcDraw, _T(buf), strlen(buf), &m_rctFreezeButton, DT_CENTER | D
T_VCENTER | DT_SINGLELINE);
        DeleteObject(SelectObject(di.hdcDraw, OrigPen));
        DeleteObject(SelectObject(di.hdcDraw, OrigBrush));


        m_dwStartDrawing = m_dwStopDrawing+1;
        if (m_dwStartDrawing >= m_dwCardioGramLength)
        {
            m_dwStartDrawing = 0;
            NextStartPos = rc.left;
        }
        else
        {
            NextStartPos = StopXPos[0] + 1;
        }

        return S_OK;

    }   // end of drawECG


    HRESULT CWebEcg::drawBPMPoints( ATL_DRAWINFO &di )
    {
        char                buf[80];
        HRESULT             Result;
        static SIZE         BPMsize;
        static ATL_DRAWINFO PrevDI;
        static long         maxTextWidth = 0;

            //---------------------------------------
            // Turn on the microphone input if necess
            //---------------------------------------
        if( (!m_bRecording) && (m_hWndCD) && (m_bMicrophoneNotServer) )
        {
            m_bRecording = true;

            WAVEFORMATEX    waveformat;

            // Set up wave recording.
            waveformat.wFormatTag       = WAVE_FORMAT_PCM;
            waveformat.nChannels        = 1;
            waveformat.nSamplesPerSec   = SAMPLE_FREQ;
            waveformat.nAvgBytesPerSec  = SAMPLE_FREQ;
            waveformat.nBlockAlign      = 1;
            waveformat.wBitsPerSample   = 8;
            waveformat.cbSize           = 0;
```

```
        Result = waveInOpen( &m_hWaveIn, WAVE_MAPPER, &waveformat,
                             (DWORD) m_hwndCD, 0, CALLBACK_WINDOW);

    }   // end if to turn on wave recorder

        //---------------------------------------
        // If no data yet, return
        //---------------------------------------
    if( mBPMData.empty() )
        return S_OK;

    RECT &  rc = *(RECT*)di.prcBounds;

        //---------------------------------------
        // Recalculate dimensions, offsets if
        // width changes
        //---------------------------------------
    bool    widthChanged = false;

    if( mWidth != mPrevWidth )
    {
        widthChanged = true;

        mHeight      = rc.bottom - rc.top;
        mWidth       = rc.right - rc.left;
        mXoffset     = rc.left + 66.0;
        mYoffset     = 0.0;

        mDrawingWid  = mWidth - mXoffset;
        mPointWidth  = mDrawingWid / (mBPMSampleCount +
                                      (mBPMSampleCount - 1) * mSpaceWidth);

        mPointWidth   = __max( mMinPointWidth, mPointWidth );
        mMaxPoints    = mDrawingWid / (mPointWidth + mSpaceWidth);
        mPointsPerCyc = mBPMSampleCount * mMaxPoints;

        if( mPointsRead >= mPointsPerCyc )
            mPointsRead   = 0;

            // Scale original values with these
        mXscale = (double) (mPointWidth + mSpaceWidth);
        mYscale = (double) (mHeight - mYoffset) / (double) mMaxHR;
        mYmax   = rc.bottom;

        mMaxHrY = (long) ( ((double)(mMaxHR - mMaxHR * .85) * mYscale) +
                                      mYoffset );

        mMinHrY = (long) ( ((double)(mMaxHR - mMaxHR * .65) * mYscale) +
                                      mYoffset );

    }   // end if mWidths don't match

    mPrevWidth = mWidth;

    HBITMAP workBitmap;
    HBITMAP origWorkBitmap;
    HFONT   origFont;
    HFONT   lSmlFont;
    HFONT   lMedFont;
    HFONT   lBigFont;
    HDC     workDC;
```

```
        HPEN        origPen;
        HBRUSH      origBrush;
        LOGBRUSH    brushDesc;

        brushDesc.lbStyle = PS_SOLID;
        brushDesc.lbColor = m_clrBackColor;

            //----------------------------------------
            // Hold onto (and re-use) workDC and
            // workBitmap if we can, but rebuild
            // them if anything in the DrawInfo
            // changes.
            //----------------------------------------
        bool    dcChanged = false;

        if( memcmp( (void *) &di, (void *) &PrevDI, sizeof PrevDI ) != 0 )
        {
            DeleteObject( m_hdcWorkDC );
            DeleteObject( m_hbmpWorkBitmap );
            m_hdcWorkDC       = CreateCompatibleDC( di.hdcDraw );
            m_hbmpWorkBitmap = CreateCompatibleBitmap( di.hdcDraw, mWidth, mHeight )
;

            dcChanged = true;

        }   // end if prev DC not = this

        workDC          = m_hdcWorkDC;
        workBitmap      = m_hbmpWorkBitmap;

        memcpy( (void *)&PrevDI, (void *)&di, sizeof PrevDI );

        origBrush = (HBRUSH) SelectObject( workDC, CreateBrushIndirect( &brushDesc )
);
        origPen   = (HPEN) SelectObject( workDC, CreatePen(PS_SOLID | PS_INSIDEFRAME
,
                                        0, brushDesc.lbColor) );

        origWorkBitmap = (HBITMAP) SelectObject( workDC, workBitmap );

        SetBkColor( workDC, m_clrBackColor );

            //----------------------------------------
            // Initialize positions, numPoints
            //----------------------------------------
        int     dataStart   = 0;
        int     numPoints   = mBPMData.numPoints();

            //----------------------------------------
            // If there are no new data points, some
            // other event caused OnDraw to be called.
            // Redraw the entire screen.  Also redraw
            // if width changed.
            //----------------------------------------
        if( !numPoints || widthChanged || dcChanged )
        {
            if( mBPMData.wrapped() || mPointsRead >= mMaxPoints )
            {
                dataStart = mBPMData.backBy( mMaxPoints );
                numPoints = mMaxPoints;
            }
            else
```

```
        {
            dataStart = 0;
            numPoints = mBPMData.curPos();
        }
    }
    else
    {
        dataStart   = mBPMData.lastRead();
        numPoints   = __min( mMaxPoints, numPoints );
    }

        //---------------------------------------
        // Determine where to start drawing on the
        // screen, and if wrapping occurs
        //---------------------------------------
    bool    drawingwrapped = false;
    int     screenStart;

        // numPoints will normally be < mPointsRead
    if( numPoints < mPointsRead )
        screenStart = (mPointsRead - numPoints) % mMaxPoints;
    else
    {
            //---------------------------------------
            // in this case, mPointsRead has wrapped
            // around from mPointsPerCyc back to 0,
            // which requires adding mPointsPerCyc
            // back in until numPoints < mPointsRead
            //---------------------------------------
        screenStart = (mPointsPerCyc + (mPointsRead - numPoints)) % mMaxPoints;

    }   // end else numPoints > mPointsRead

        //---------------------------------------
        // Determine if drawing will be wrapped
        //---------------------------------------
    if( screenStart + numPoints >= mMaxPoints )
        drawingWrapped = true;

        //---------------------------------------
        // If not wrapped, just draw straight set
        // of points.
        //---------------------------------------
    if( !drawingWrapped )
    {
        int     xStart = mXoffset + (int)(screenStart * mXscale);
        int     xEnd   = xStart + numPoints * mXscale;

        drawPoints( workDC, rc, dataStart, xStart, xEnd, numPoints );

            //---------------------------------------
            // Blit the new area to the screen
            //---------------------------------------
        int     xWidth = numPoints * mXscale;

        BitBlt( di.hdcDraw, xStart, rc.top, xWidth, mHeight,
                workDC,     xStart, rc.top, SRCCOPY );

    }   // end if not wrapped
    else
    {
            //---------------------------------------
            // Calculate bounding rectangles of both
```

```
            // regions
            //----------------------------------------
        int     xStart[2];
        int     xEnd[2];
        int     dataPoints[2];

        dataPoints[0] = mMaxPoints - screenStart;
        dataPoints[1] = numPoints  - dataPoints[0];

        xStart[0] = mXoffset + (int) (screenStart * mXscale);
        xEnd[0]   = xStart[0] + dataPoints[0] * mXscale;

        xStart[1] = mXoffset;
        xEnd[1]   = xStart[1] + dataPoints[1] * mXscale;

        int     datStart2 = dataStart;

        mBPMData.incr( datStart2, dataPoints[0] );

            //----------------------------------------
            // Draw the points
            //----------------------------------------
        drawPoints( workDC, rc, dataStart, xStart[0], xEnd[0], dataPoints[0] );
        drawPoints( workDC, rc, datStart2, xStart[1], xEnd[1], dataPoints[1] );

            //----------------------------------------
            // Blit the new areas to the screen
            //----------------------------------------
        int     xWidth[2];

        xWidth[0] = dataPoints[0] * mXscale;
        xWidth[1] = dataPoints[1] * mXscale;

        BitBlt( di.hdcDraw, xStart[0], rc.top, xWidth[0], mHeight,
                workDC,     xStart[0], rc.top, SRCCOPY );

        BitBlt( di.hdcDraw, xStart[1], rc.top, xWidth[1], mHeight,
                workDC,     xStart[1], rc.top, SRCCOPY );

    }   // end else wrapped

        //----------------------------------------
        // Setup for drawing text
        //----------------------------------------
    lSmlFont = CreateFont( -MulDiv( 9,
                            GetDeviceCaps( workDC, LOGPIXELSY ), 72 ),
                            0 /*mWidth*/,
                            0 /*escapement*/,
                            0 /*orientation*/,
                            FW_NORMAL/*weight*/,
                            FALSE /*Italic*/,
                            FALSE /*Underline*/,
                            FALSE /*StrikeOut*/,
                            DEFAULT_CHARSET,
                            OUT_TT_ONLY_PRECIS /*OutputPrecision*/,
                            CLIP_DEFAULT_PRECIS,
                            DEFAULT_QUALITY,
                            DEFAULT_PITCH | FF_ROMAN,
                            "Times New Roman"
                            );

    lMedFont = CreateFont( -MulDiv( 10,
                            GetDeviceCaps( workDC, LOGPIXELSY ), 72 ),
```

```
                                0 /*mWidth*/,
                                0 /*escapement*/,
                                0 /*orientation*/,
                                FW_NORMAL/*weight*/,
                                FALSE /*Italic*/,
                                FALSE /*Underline*/,
                                FALSE /*StrikeOut*/,
                                DEFAULT_CHARSET,
                                OUT_TT_ONLY_PRECIS /*OutputPrecision*/,
                                CLIP_DEFAULT_PRECIS,
                                DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN,
                                //"Times New Roman"
                                "Arial"
                                );

    lBigFont = CreateFont( -MulDiv( m_nBPMFontSize,
                                GetDeviceCaps( workDC, LOGPIXELSY ), 72 ),
                                0 /*mWidth*/,
                                0 /*escapement*/,
                                0 /*orientation*/,
                                FW_NORMAL/*weight*/,
                                FALSE /*Italic*/,
                                FALSE /*Underline*/,
                                FALSE /*StrikeOut*/,
                                DEFAULT_CHARSET,
                                OUT_TT_ONLY_PRECIS /*OutputPrecision*/,
                                CLIP_DEFAULT_PRECIS,
                                DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN,
                                //"Times New Roman"
                                "Arial"
                                );

    origFont = (HFONT) SelectObject( workDC, lBigFont );

    SetTextAlign( workDC, TA_RIGHT );

        //----------------------------------------
        // Figure out how big the biggest
        // reading we will ever take is.
        //----------------------------------------
    GetTextExtentPoint32( workDC, _T("999 BPM"), strlen("999 BPM"), &BPMsize );
    maxTextWidth = __max( BPMsize.cy, maxTextWidth );

    SetBkMode( workDC, OPAQUE );

        //----------------------------------------
        // Max HR at top left
        //----------------------------------------
    DrawString( workDC, lSmlFont, TA_LEFT, cnRed, rc.left+2, rc.top, "MAX" );

    sprintf( buf, "%d", mMaxHR );
    DrawString( workDC, lMedFont, TA_RIGHT, cnRed, mXoffset-2, rc.top, buf );

        //----------------------------------------
        // 85 % of max
        //----------------------------------------
    long    hr = (long) ((double) mMaxHR * 0.85);

    DrawString( workDC, lSmlFont, TA_LEFT, cnLtBlu, rc.left+2, mMaxHrY, "85%" );

    sprintf( buf, "%d", hr );
```

```
        DrawString( workDC, lMedFont, TA_RIGHT, cnLtBlu, mXoffset-2, mMaxHrY, buf );

            //----------------------------------------
            // 65 % of max
            //----------------------------------------
        hr = (long) ((double) mMaxHR * 0.65);

        DrawString( workDC, lSmlFont, TA_LEFT, cnLtBlu, rc.left+2, mMinHrY, "65%" );

        sprintf( buf, "%d", hr );
        DrawString( workDC, lMedFont, TA_RIGHT, cnLtBlu, mXoffset-2, mMinHrY, buf );

            //----------------------------------------
            // Current BMP lower left
            //----------------------------------------
        if( m_nAvgBPM < 10 )
            sprintf( buf, " %d    BPM", m_nAvgBPM );
        else if( m_nAvgBPM < 100 )
            sprintf( buf, " %d  BPM", m_nAvgBPM );
        else
            sprintf( buf, " %d BPM", m_nAvgBPM );

        SetBkColor( workDC, m_clrGridColor );

        DrawString( workDC, lBigFont, TA_RIGHT, m_clrBackColor,
                    mXoffset - 2, rc.bottom - BPMsize.cy, buf );

        SetBkMode( di.hdcDraw, TRANSPARENT );

            // Left border line
        DeleteObject( SelectObject( workDC, CreatePen( m_nGridStyle, 1, cnBlue ) ) )
    ;
        DrawLine( workDC, mXoffset-1, rc.top, mXoffset-1, rc.bottom );

        SelectObject( workDC, origFont );
        SelectObject( workDC, origPen );
        SelectObject( workDC, origBrush );

        DeleteObject( lSmlFont );
        DeleteObject( lMedFont );
        DeleteObject( lBigFont );

            //----------------------------------------
            // Blit new area to screen
            //----------------------------------------
        BitBlt( di.hdcDraw, rc.left, rc.top, mXoffset, rc.bottom,
                workDC,     rc.left, rc.top, SRCCOPY );

        return S_OK;

    }   // end of drawBPMPoints

    void CWebEcg::drawPoints( HDC  &dc,
                              RECT &rc,
                              int  dataStart,
                              int  xStart,
                              int  xEnd,
                              int  numPoints )
    {

        HPEN      blackPen    = CreatePen( PS_SOLID | PS_INSIDEFRAME, 0, cnBlack );
        HPEN      redPen      = CreatePen( PS_SOLID | PS_INSIDEFRAME, 0, cnRed );
```

```
HPEN      gridPen      = CreatePen( PS_SOLID | PS_INSIDEFRAME, 0, cnLtBlu );
HPEN      greenPen     = CreatePen( PS_SOLID | PS_INSIDEFRAME, 0, cnGreen );
HPEN      purpPen      = CreatePen( PS_SOLID | PS_INSIDEFRAME, 0, cnPurp );
HPEN      origPen;
HBRUSH    bkBrush      = CreateSolidBrush( m_clrBackColor );
HBRUSH    greenBrush   = CreateSolidBrush( cnGreen );
HBRUSH    redBrush     = CreateSolidBrush( cnRed );
HBRUSH    purpBrush    = CreateSolidBrush( cnPurp );
HBRUSH    origBrush;
POINT     points[3];

origBrush = (HBRUSH) SelectObject( dc, bkBrush );
origPen   = (HPEN)   SelectObject( dc, blackPen );

    //---------------------------------------
    // First clear previous areas
    //---------------------------------------
Rectangle( dc, xStart, rc.top, xEnd, rc.bottom );

    //---------------------------------------
    // Draw gridlines
    //---------------------------------------
SelectObject( dc, gridPen );

int lineStart = xStart - mSpaceWidth;
lineStart     = __max( mXoffset, lineStart );

DrawLine( dc, lineStart, mMaxHrY, xEnd, mMaxHrY );
DrawLine( dc, lineStart, mMinHrY, xEnd, mMinHrY );

    //---------------------------------------
    // Determine where to stop reading in the
    // ring buffer
    //---------------------------------------
int       dataStop = dataStart;

mBPMData.incr( dataStop, numPoints );

    //---------------------------------------
    // Draw new points
    //---------------------------------------
int     x;
int     y;

for( int i = dataStart, xI = 0; i != dataStop; mBPMData.incr( i ), xI++ )
{
    x = xStart + (int) (xI * mXscale);
    y = mYmax - (int) (mBPMData[i] * mYscale + mYoffset);

    if( y < 0 || y >= mYmax )
        continue;

        //---------------------------------------
        // Since the upper left x,y is 0,0
        // a value > min HR Y means the HR
        // is too low
        //---------------------------------------
    if( y > mMinHrY )
    {
        SelectObject( dc, purpBrush );
        SelectObject( dc, purpPen );
        upArrow( points, x, y, mPointWidth );
        Polygon( dc, points, 3 );
```

```
            }
            else if( y < mMaxHrY )  // HR too high
            {
                SelectObject( dc, redBrush );
                SelectObject( dc, redPen );
                downArrow( points, x, y, mPointWidth );
                Polygon( dc, points, 3 );
            }
            else    // HR just right
            {
                SelectObject( dc, greenBrush );
                SelectObject( dc, greenPen );
                Rectangle( dc, x, y, x + mPointWidth, y + mPointWidth );
            }

        }   // end for thru bars

            //---------------------------------------
            // Update ring buffer index to indicate
            // we've read numPoints
            //---------------------------------------
        mBPMData.updateBy( numPoints );

            //---------------------------------------
            // Clean up
            //---------------------------------------
        SelectObject( dc, origBrush );
        SelectObject( dc, origPen );

        DeleteObject( bkBrush );
        DeleteObject( redBrush );
        DeleteObject( greenBrush );
        DeleteObject( purpBrush );

        DeleteObject( gridPen );
        DeleteObject( blackPen );
        DeleteObject( redPen );
        DeleteObject( greenPen );
        DeleteObject( purpPen );

    }   // end of drawPoints

    void CWebEcg::upArrow( POINT *pPoints, int x, int y, int wid )
    {
        pPoints[0].x = x;
        pPoints[0].y = y + wid;
        pPoints[1].x = x + wid / 2;
        pPoints[1].y = y;
        pPoints[2].x = x + wid;
        pPoints[2].y = y + wid;

    }   // end of upArrow

    void CWebEcg::downArrow( POINT *pPoints, int x, int y, int wid )
    {
        pPoints[0].x = x;
        pPoints[0].y = y;
        pPoints[1].x = x + wid;
        pPoints[1].y = y;
        pPoints[2].x = x + wid / 2;
        pPoints[2].y = y + wid;
```

```
    }   // end of downArrow


HRESULT CWebEcg::OnWimOpen(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandl
ed)
{
    int i;

    // Should have LOTS of memory checking error handling all through here...
    m_pFullWav = NULL;

    for (i=0; i<4; i++)
    {
        m_pWavBuf[i] = new(unsigned char[WORK_BUFFER_SIZE]);
        m_pWaveHdr[i]->lpData          = (char *)m_pWavBuf[i];
        m_pWaveHdr[i]->dwBufferLength  = WORK_BUFFER_SIZE;
        m_pWaveHdr[i]->dwBytesRecorded = 0;
        m_pWaveHdr[i]->dwUser          = 0;
        m_pWaveHdr[i]->dwFlags         = 0;
        m_pWaveHdr[i]->dwLoops         = 1;
        m_pWaveHdr[i]->lpNext          = NULL;
        m_pWaveHdr[i]->reserved        = 0;
        waveInPrepareHeader (m_hWaveIn, m_pWaveHdr[i], sizeof (WAVEHDR));
        waveInAddBuffer     (m_hWaveIn, m_pWaveHdr[i], sizeof (WAVEHDR));
    }


    // Start sampling
    m_dwFullWavLength = 0;
    MMRESULT mmResult = waveInStart(m_hWaveIn);

    return S_OK;
}

HRESULT CWebEcg::OnWimData(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandl
ed)
{
    const unsigned int OverlapPts = 4;
    const unsigned int DiscardPts = 4;

    static int SampleCount;

    static signed char  wavedata[WORK_BUFFER_SIZE+OverlapPts*DECIMATION];
    static short        lpDataOut[WORK_BUFFER_SIZE/DECIMATION + OverlapPts];
    int                 iDataOutSize;
    unsigned long       i;
    bool                bSomethingThere;
    bool                initPass = true;

    bSomethingThere = false;

    if (!m_bFrozen)
    {
        for (i=0; i<((PWAVEHDR) lParam)->dwBytesRecorded; i++)
        {
            if ( ((unsigned char)(((PWAVEHDR) lParam)->lpData[i]) <= 128-m_nVolu
meThreshold) ||
                 ((unsigned char)(((PWAVEHDR) lParam)->lpData[i]) >= 128+m_nVolum
eThreshold) )
            {
                bSomethingThere = true;
```

```
                    break;
                }
            }
        }

        if (bSomethingThere)
        {
            if( initPass )
            {
                // Do first time initialization here
                initPass = false;
                // Only do this the first time we get something past the volume thre
shold.
                // Setting m_dwStopDrawing to -1 will work properly as long as the f
ollowing
                // code moves m_dwStopDrawing forward by filling in the new data.  I
f you do
                // not have any new data and let OnDraw try to work with m_dwStopDra
wing==-1,
                // you get a crash.
                // Also, -1 is an odd value to push into an UNSIGNED long.  Fortunat
ely, it
                // has a value of all 1's, and adding one to that will roll it over
to all 0's,
                // so it has the essential -1'ness that we need.
                m_dwStopDrawing = -1;
            }

            for (i=0; i<WORK_BUFFER_SIZE; i++)
            {
                signed char p = (signed char) ((PWAVEHDR) lParam)->lpData[i];
                signed char q = (signed char) (p - 128);
                signed char r;

                wavedata[i+OverlapPts*DECIMATION] = (signed char)(((PWAVEHDR) lParam
)->lpData[i] - 128);

                r = wavedata[i+OverlapPts*DECIMATION];
            }

            DemodECG(lpDataOut, &iDataOutSize,
                (char *)wavedata, WORK_BUFFER_SIZE + OverlapPts*DECIMATION,
                80,
                1520);

            // We are done with this buffer.  Return it to the system as soon as
            // possible.
            waveInAddBuffer (m_hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR));

            // First several ECG points are garbage.
            // Save the last several ECG points worth of samples to prepend to the
            // next buffer.  That way the first several ECG points of this new buffe
r
            // are garbage and discarded (OK, since they are the last several return
ed
            // this time).
            if (OverlapPts)
            {
                // OK, some explanation here...
                // We want to move the LAST OverlapPts worth of samples to the front
                // of wavedata.  The end of wavedata is at:
                // (remember - each ECG point is equivalent to DECIMATION samples)
                //
```

```
                // wavedata+WORK_BUFFER_SIZE+OverlapPts*DECIMATION
                //
                // (actually, that is the address of the first byte after the end of
                // wavedata, but if you wanted to see the last 1 byte of wavedata, y
ou
                // would go to that address-1, and if you wanted to see the last 10
bytes
                // you would go to that address-10, etc.  This makes sense when you
read
                // the following)
                // But we want to start OverlapPts worth of samples before that, so:
                //
                // wavedata+WORK_BUFFER_SIZE+OverlapPts*DECIMATION-OverlapPts*DECIMA
TION
                //
                // which simplifies to:
                //
                // wavedata+WORK_BUFFER_SIZE
                MoveMemory(wavedata, wavedata+WORK_BUFFER_SIZE, OverlapPts*DECIMATIO
N);
            }

            // First several returned points are garbage...
            if (DiscardPts)
            {
                iDataOutSize -= DiscardPts;
                MoveMemory(lpDataOut,&(lpDataOut[DiscardPts]),iDataOutSize * sizeof(
lpDataOut[0]));
            }

            if (m_bConnectedToServer)
            {
                send(m_sckLocalSocket, (const char *)lpDataOut, iDataOutSize*2, 0);
            }

            NewRawData(lpDataOut, iDataOutSize);


        }
        else
        {
            waveInAddBuffer (m_hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR));
        }
        return S_OK;
    }

    HRESULT CWebEcg::OnWimClose(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHand
led)
    {
        // Never gets called in time if waveInClose is called from ~CWebEcg...
        //MessageBox("OnWimClose","NOTE",MB_OK);
        //      waveInUnprepareHeader(m_hWaveIn, m_pWaveHdr1, sizeof(WAVEHDR));
        //      waveInUnprepareHeader(m_hWaveIn, m_pWaveHdr2, sizeof(WAVEHDR));

        //      delete m_pWaveHdr1;
        //      delete m_pWaveHdr2;
        //      delete m_pWavBuf[0];
        //      delete m_pWavBuf[1];

        return S_OK;
    }
```

```
HRESULT CWebEcg::OnWomOpen(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandl
ed)
{

    return S_OK;
}
HRESULT CWebEcg::OnWomDone(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandl
ed)
{

    return S_OK;
}
HRESULT CWebEcg::OnWomClose(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHand
led)
{

    return S_OK;
}

STDMETHODIMP CWebEcg::get_GridColor(OLE_COLOR * pVal)
{
    // TODO: Add your implementation code here
    *pVal = m_clrGridColor;
    return S_OK;
}

STDMETHODIMP CWebEcg::put_GridColor(OLE_COLOR newVal)
{
    // TODO: Add your implementation code here

    if (FireOnRequestEdit(dispidGridColor) == S_FALSE)
    {
        return S_FALSE;
    }

    m_clrGridColor = newVal;

    SetDirty(TRUE);
    FireOnChanged(dispidGridColor);
    FireViewChange();

    return S_OK;
}

STDMETHODIMP CWebEcg::get_GridStyle(long * pVal)
{
    // TODO: Add your implementation code here
    *pVal = m_nGridStyle;
    return S_OK;
}

STDMETHODIMP CWebEcg::put_GridStyle(long newVal)
{
    // TODO: Add your implementation code here

    if (FireOnRequestEdit(dispidGridStyle) == S_FALSE)
    {
        return S_FALSE;
    }

    m_nGridStyle = newVal;

    SetDirty(TRUE);
```

```
        FireOnChanged(dispidGridStyle);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_MinmV(double * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_dMinmV;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_MinmV(double newVal)
    {
        if (FireOnRequestEdit(dispidMinmV) == S_FALSE)
        {
            return S_FALSE;
        }

        m_dMinmV = newVal;

        SetDirty(TRUE);
        FireOnChanged(dispidMinmV);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_MaxmV(double * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_dMaxmV;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_MaxmV(double newVal)
    {
        if (FireOnRequestEdit(dispidMaxmV) == S_FALSE)
        {
            return S_FALSE;
        }

        m_dMaxmV = newVal;

        SetDirty(TRUE);
        FireOnChanged(dispidMaxmV);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_StepmV(double * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_dStepmV;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_StepmV(double newVal)
    {
        if (FireOnRequestEdit(dispidStepmV) == S_FALSE)
        {
```

```
            m_pCardioGram[i] = 0;
            m_pVerticalBar[i] = 0;
        }

        mBPMSampleCount = newVal * 4; // 4 samples per sec
        mPointsPerCyc   = mBPMSampleCount * mMaxPoints;

        if( !mPointsPerCyc )
            mPointsPerCyc = mBPMSampleCount;

        mBPMData.resize( mBPMSampleCount );

        SetDirty(TRUE);
        FireOnChanged(dispidDisplaySeconds);
        FireViewChange();

        return S_OK;

    }   // end of put_DisplaySeconds


    STDMETHODIMP CWebEcg::get_VolumeThreshold(short * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_nVolumeThreshold;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_VolumeThreshold(short newVal)
    {
        // TODO: Add your implementation code here
        if (FireOnRequestEdit(dispidVolumeThreshold) == S_FALSE)
        {
            return S_FALSE;
        }

        m_nVolumeThreshold = newVal;

        SetDirty(TRUE);
        FireOnChanged(dispidVolumeThreshold);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_Filtering(short * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_nFiltering;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_Filtering(short newVal)
    {
        // TODO: Add your implementation code here
        if (FireOnRequestEdit(dispidFiltering) == S_FALSE)
        {
            return S_FALSE;
        }

        m_nFiltering = newVal;

        SetDirty(TRUE);
```

```
        FireOnChanged(dispidFiltering);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_BPMFontSize(short * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_nBPMFontSize;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_BPMFontSize(short newVal)
    {
        // TODO: Add your implementation code here
        if (FireOnRequestEdit(dispidBPMFontSize) == S_FALSE)
        {
            return S_FALSE;
        }

        m_nBPMFontSize = newVal;

        SetDirty(TRUE);
        FireOnChanged(dispidBPMFontSize);
        FireViewChange();

        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_DetectPercent(double * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_dDetectPercent;
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_DetectPercent(double newVal)
    {
        // TODO: Add your implementation code here
        if (FireOnRequestEdit(dispidDetectPercent) == S_FALSE)
        {
            return S_FALSE;
        }

        m_dDetectPercent = newVal;

        mBPM.DetectPercent( newVal );

        SetDirty(TRUE);
        FireOnChanged(dispidDetectPercent);
        FireViewChange();
        return S_OK;
    }

    STDMETHODIMP CWebEcg::get_ServerAddress(BSTR * pVal)
    {
        // TODO: Add your implementation code here
        *pVal = m_bstrServerAddress.Copy();
        return S_OK;
    }

    STDMETHODIMP CWebEcg::put_ServerAddress( BSTR newVal )
```

```
{
    // TODO: Add your implementation code here
    USES_CONVERSION; // Seems necessary for a call to W2A later to work...

    if( FireOnRequestEdit(dispidServerAddress) == S_FALSE )
    {
        return S_FALSE;
    }

    m_bstrServerAddress = newVal;

    if( m_bConnectedToServer )
    {
        shutdown( m_sckLocalSocket, 1 );
        closesocket( m_sckLocalSocket );
        m_bConnectedToServer = false;
    }

    if( m_bstrServerAddress.Length() < 7 )
        return S_OK;

    if( m_bMicrophoneNotServer )
        ConnectToServer();

    SetDirty( TRUE );
    FireOnChanged( dispidServerAddress );
    FireViewChange();

    return S_OK;
}

STDMETHODIMP CWebEcg::get_MicrophoneNotServer(short * pVal)
{
    // TODO: Add your implementation code here
    *pVal = m_bMicrophoneNotServer;
    return S_OK;
}

STDMETHODIMP CWebEcg::put_MicrophoneNotServer(short newVal)
{
    // TODO: Add your implementation code here
    if (FireOnRequestEdit(dispidMicrophoneNotServer) == S_FALSE)
    {
        return S_FALSE;
    }

    if (m_bMicrophoneNotServer && newVal)
    {
        // Both true -- no change -- don't care
    }
    else if ( ! (m_bMicrophoneNotServer || newVal))
    {
        // Neither true -- no change -- don't care
    }
    else
    {
        // State changed - adjust as necessary.
        if (m_bMicrophoneNotServer)
        {
            // was running from microphone, now from net
            // must shut down microphone
            int i;
```

```
            waveInReset(m_hWaveIn);
            waveInClose(m_hWaveIn);

            //      Sleep((1.0/SAMPLE_FREQ)*(2*WORK_BUFFER_SIZE)*1000); // Give
system time to shut down whatever.
            //      Sleep((1000/SAMPLE_FREQ)*(2*WORK_BUFFER_SIZE)); // Give syst
em time to shut down whatever.
            //      Sleep((1000*2*WORK_BUFFER_SIZE)/SAMPLE_FREQ);   // Give syst
em time to shut down whatever.
            Sleep((2000*WORK_BUFFER_SIZE)/SAMPLE_FREQ); // Give system time to s
hut down whatever.

            for (i=0; i<4; i++)
            {
                waveInUnprepareHeader(m_hWaveIn, m_pWaveHdr[i], sizeof(WAVEHDR))
;
                delete m_pWaveHdr[i];
                delete m_pWavBuf[i];
            }
        }
        else
        {
            // was running from net, now from microphone
            if( m_bstrServerAddress.Length() >= 7 )
                ConnectToServer();
            m_bRecording = false;
        }

        m_bMicrophoneNotServer = newVal;

        SetDirty(TRUE);
        FireOnChanged(dispidMicrophoneNotServer);
        FireViewChange();
    }
    return S_OK;
}

void CWebEcg::NewRawData( signed short * RawData, unsigned long NumData )
{
    static int              SampleCount;
    unsigned long           i;
    short                   CurrFilteredPoint;
    static unsigned long    SamplesSinceQRS;
    static bool             InitPass = true;
    static short            Last2SecsFiltered[ECG_FREQ*2];
    static short            PrevBPM;

        // m_dwStopDrawing should always point to the
        // most recently inserted CardioGram Point.
    signed char             WorkReading;


    for( i=0; i < (unsigned long)NumData; i++ )
    {
        // CharFilter is #defined macro in QRS.h that converts input into
        // SIGNED character, and pegs value at 127 or -128 boundaries.
        WorkReading = CharFilter( RawData[i] );

        // The following set of filters is in the QRS files, and is
        // taken from a routine in a book on biomedical signal processing.
        // As we played with the signal, certain parts were changed and/or
        // removed.
```

```
            CurrFilteredPoint = LowPassFilter( WorkReading );

            if( m_nFiltering == 1 )
                WorkReading = CharFilter( CurrFilteredPoint );

            CurrFilteredPoint = HighPassFilter( CurrFilteredPoint );

            if( m_nFiltering == 2 )
                WorkReading = CharFilter( CurrFilteredPoint );

            WorkReading += 55;

            CurrFilteredPoint = Derivative( CurrFilteredPoint );

            if( m_nFiltering == 3 )
                WorkReading = CharFilter( CurrFilteredPoint );

            CurrFilteredPoint = Square( CurrFilteredPoint );

            if( m_nFiltering == 4 )
                WorkReading = CharFilter( CurrFilteredPoint );

            //          CurrFilteredPoint = MovingWindowIntegral(CurrFilteredPoint);
            if (m_nFiltering >= 5) WorkReading = CharFilter(CurrFilteredPoint);

            // Store the reading in the CardioGram
            // m_dwStopDrawing should always point to the most recently inserted Car
dioGram Point.

            if( mbDrawECG )
            {
                m_dwStopDrawing++;

                if (m_dwStopDrawing >= m_dwCardioGramLength)
                    m_dwStopDrawing = 0;

                m_pCardioGram[m_dwStopDrawing] = WorkReading;

                // Flag the axis, if necessary
                if (SampleCount == 0)
                    m_pVerticalBar[m_dwStopDrawing] = 1;
                else
                    m_pVerticalBar[m_dwStopDrawing] = 0;

                SampleCount++;

                if (SampleCount >= ECG_FREQ)
                    SampleCount = 0;

            }   // end if mbDrawECG

                // Detect QRS complexes
                // If this seems overly simplified, set Filtering property to 5 and
                // take a look at the fully filtered data.
                // First, find max value in last 2 seconds of data.
            m_nBPM    = mBPM.Update( CurrFilteredPoint );
            m_nAvgBPM = mBPM.AvgBPM();


        }   // end for thru data


        if( !mbDrawECG )
```

```
        {
            mCumSamps += NumData;

            if( mCumSamps >= mSampsPerBPMPt )
            {
                int     minHr = __min( m_nAvgBPM, mMaxHR );

                mBPMData.push_back( minHr );

                mCumSamps = 0;

                    // mPointsRead tells us where to put
                    // the point on the screen when the
                    // number of screen points is less
                    // than the number of data points
                ++mPointsRead;

                if( mPointsRead >= mPointsPerCyc )
                    mPointsRead = 0;

                FireViewChange();

            }   // end if time to draw again

        }   // end if ! drawECG
        else
            FireViewChange();

    }   // end of NewRawData

    void CWebEcg::ConnectToServer()
    {
        USES_CONVERSION;

        m_bConnectedToServer = true;
        m_sckLocalSocket = socket(PF_INET, SOCK_STREAM, 0);
        m_sckadrRemoteSocketAddress.sin_family = PF_INET;
        m_sckadrRemoteSocketAddress.sin_port = htons(4321);
        m_sckadrRemoteSocketAddress.sin_addr.s_addr = GetAddr(W2A(m_bstrServerAddres
    s));
        m_sckadrRemoteSocketAddress.sin_zero[0] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[1] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[2] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[3] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[4] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[5] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[6] = 0;
        m_sckadrRemoteSocketAddress.sin_zero[7] = 0;
        if (connect(m_sckLocalSocket,
            (sockaddr *)(&m_sckadrRemoteSocketAddress),
            sizeof(m_sckadrRemoteSocketAddress)))
        {
            // Some kind of problem...
            // 05/02 - No window here....
            char buff[200];
            sprintf( buff, "Connect: Could not attach to host: %s",
                    m_bstrServerAddress.m_str );
            MessageBox( buff, "DR DAVE", MB_OK);
            m_bConnectedToServer = false;
        }
    }
```

```
STDMETHODIMP CWebEcg::get_AddNewPoint(short * pVal)
{
    // TODO: Add your implementation code here
    *pVal = 0;
    return S_OK;
}

STDMETHODIMP CWebEcg::put_AddNewPoint(short newVal)
{
    // TODO: Add your implementation code here
    if (FireOnRequestEdit(dispidDetectPercent) == S_FALSE)
    {
        return S_FALSE;
    }

    NewRawData(&newVal, 1);

    SetDirty(TRUE);
    FireOnChanged(dispidDetectPercent);
    return S_OK;
}


STDMETHODIMP CWebEcg::get_Age( short *pVal )
{
    *pVal = mAge;
    return S_OK;
}


STDMETHODIMP CWebEcg::put_Age( short newVal )
{
    mAge = newVal;
    return S_OK;
}
STDMETHODIMP CWebEcg::get_Average_Count(long *pVal)
{
    *pVal = mBPM.AverageCount();
    return S_OK;
}

STDMETHODIMP CWebEcg::put_Average_Count(long newVal)
{
    mBPM.AverageCount( newVal );

    return S_OK;
}

STDMETHODIMP CWebEcg::get_ViewECG(BOOL *pVal)
{
    if( mbDrawECG )
        *pVal = TRUE;
    else
        *pVal = FALSE;

    return S_OK;
}

STDMETHODIMP CWebEcg::put_ViewECG(BOOL newVal)
{
    if( newVal == TRUE )
        mbDrawECG = true;
```

```
        else
            mbDrawECG = false;

        return S_OK;
    }


    inline void DrawLine(HDC hdc, long x1, long y1, long x2, long y2)
    {
        MoveToEx(hdc, x1, y1, NULL);
        LineTo  (hdc, x2, y2);
    }

    void DrawString( HDC      dc,
                     HFONT    font,
                     UINT     align,
                     COLORREF col,
                     int      left,
                     int      top,
                     char     *str )
    {
        SelectObject( dc, font );
        SetTextAlign( dc, align );
        SetTextColor( dc, col );

        TextOut( dc, left, top, _T(str), strlen(str) );

    }   // end of DrawString
```

```
#include <olectl.h>
// webEcgControl.idl : IDL source for webEcgControl.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (webEcgControl.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

typedef enum propertydispids
    {
    dispidGridColor = 2,
    dispidGridStyle = 3,
    dispidMaxmV     = 4,
    dispidMinmV     = 5,
    dispidStepmV    = 6,
    dispidForeWidth = 7,
    dispidDisplaySeconds = 8,
    dispidVolumeThreshold = 9,
    dispidFiltering = 10,
    dispidBPMFontSize = 11,
    dispidDetectPercent = 12,
    dispidServerAddress = 13,
    dispidMicrophoneNotServer = 14,
    dispidNewPointByte1 = 15,
    dispidNewPointByte2 = 16,
    dispidAddNewPoint = 17,
    }PROPERTYDISPIDS;

    [
        object,
        uuid(E254A2C1-B470-11D2-8455-00104B05249C),
        dual,
        helpstring("IWebEcg Interface"),
        pointer_default(unique)
    ]
    interface IWebEcg : IDispatch
    {
        [propput, id(DISPID_BACKCOLOR)]
        HRESULT BackColor([in]OLE_COLOR clr);
        [propget, id(DISPID_BACKCOLOR)]
        HRESULT BackColor([out,retval]OLE_COLOR* pclr);
        [propput, id(DISPID_BORDERCOLOR)]
        HRESULT BorderColor([in]OLE_COLOR clr);
        [propget, id(DISPID_BORDERCOLOR)]
        HRESULT BorderColor([out, retval]OLE_COLOR* pclr);
        [propput, id(DISPID_BORDERSTYLE)]
        HRESULT BorderStyle([in]long style);
        [propget, id(DISPID_BORDERSTYLE)]
        HRESULT BorderStyle([out, retval]long* pstyle);
        [propput, id(DISPID_BORDERWIDTH)]
        HRESULT BorderWidth([in]long width);
        [propget, id(DISPID_BORDERWIDTH)]
        HRESULT BorderWidth([out, retval]long* width);
        [propput, id(DISPID_FORECOLOR)]
        HRESULT ForeColor([in]OLE_COLOR clr);
        [propget, id(DISPID_FORECOLOR)]
        HRESULT ForeColor([out,retval]OLE_COLOR* pclr);
        [propput, id(DISPID_BORDERVISIBLE)]
        HRESULT BorderVisible([in]VARIANT_BOOL vbool);
        [propget, id(DISPID_BORDERVISIBLE)]
        HRESULT BorderVisible([out, retval]VARIANT_BOOL* pbool);
```

```
        [propput, id(DISPID_MOUSEPOINTER)]
        HRESULT MousePointer([in]long pointer);
        [propget, id(DISPID_MOUSEPOINTER)]
        HRESULT MousePointer([out, retval]long* ppointer);
        [propputref, id(DISPID_MOUSEICON)]
        HRESULT MouseIcon([in]IPictureDisp* pMouseIcon);
        [propput, id(DISPID_MOUSEICON)]
        HRESULT MouseIcon([in]IPictureDisp* pMouseIcon);
        [propget, id(DISPID_MOUSEICON)]
        HRESULT MouseIcon([out, retval]IPictureDisp** ppMouseIcon);
        [propget, id(dispidGridColor), helpstring("property GridColor")] HRESULT
GridColor([out, retval] OLE_COLOR *pVal);
        [propput, id(dispidGridColor), helpstring("property GridColor")] HRESULT
GridColor([in] OLE_COLOR newVal);
        [propget, id(dispidGridStyle), helpstring("property GridStyle")] HRESULT
GridStyle([out, retval] long *pVal);
        [propput, id(dispidGridStyle), helpstring("property GridStyle")] HRESULT
GridStyle([in] long newVal);
        [propget, id(dispidMinmV), helpstring("property MinmV")] HRESULT MinmV([o
ut, retval] double *pVal);
        [propput, id(dispidMinmV), helpstring("property MinmV")] HRESULT MinmV([i
n] double newVal);
        [propget, id(dispidMaxmV), helpstring("property MaxmV")] HRESULT MaxmV([o
ut, retval] double *pVal);
        [propput, id(dispidMaxmV), helpstring("property MaxmV")] HRESULT MaxmV([i
n] double newVal);
        [propget, id(dispidStepmV), helpstring("property StepmV")] HRESULT StepmV
([out, retval] double *pVal);
        [propput, id(dispidStepmV), helpstring("property StepmV")] HRESULT StepmV
([in] double newVal);
        [propget, id(dispidForeWidth), helpstring("property ForeWidth")] HRESULT
ForeWidth([out, retval] long *pVal);
        [propput, id(dispidForeWidth), helpstring("property ForeWidth")] HRESULT
ForeWidth([in] long newVal);
        [propget, id(dispidDisplaySeconds), helpstring("property DisplaySeconds")
] HRESULT DisplaySeconds([out, retval] short *pVal);
        [propput, id(dispidDisplaySeconds), helpstring("property DisplaySeconds")
] HRESULT DisplaySeconds([in] short newVal);
        [propget, id(dispidVolumeThreshold), helpstring("property VolumeThreshold
")] HRESULT VolumeThreshold([out, retval] short *pVal);
        [propput, id(dispidVolumeThreshold), helpstring("property VolumeThreshold
")] HRESULT VolumeThreshold([in] short newVal);
        [propget, id(dispidFiltering), helpstring("property Filtering")] HRESULT
Filtering([out, retval] short *pVal);
        [propput, id(dispidFiltering), helpstring("property Filtering")] HRESULT
Filtering([in] short newVal);
        [propget, id(dispidBPMFontSize), helpstring("property BPMFontSize")] HRES
ULT BPMFontSize([out, retval] short *pVal);
        [propput, id(dispidBPMFontSize), helpstring("property BPMFontSize")] HRES
ULT BPMFontSize([in] short newVal);
        [propget, id(dispidDetectPercent), helpstring("property DetectPercent")]
HRESULT DetectPercent([out, retval] double *pVal);
        [propput, id(dispidDetectPercent), helpstring("property DetectPercent")]
HRESULT DetectPercent([in] double newVal);
        [propget, id(dispidServerAddress), helpstring("property ServerAddress")]
HRESULT ServerAddress([out, retval] BSTR *pVal);
        [propput, id(dispidServerAddress), helpstring("property ServerAddress")]
HRESULT ServerAddress([in] BSTR newVal);
        [propget, id(dispidMicrophoneNotServer), helpstring("property MicrophoneN
otServer")] HRESULT MicrophoneNotServer([out, retval] short *pVal);
        [propput, id(dispidMicrophoneNotServer), helpstring("property MicrophoneN
otServer")] HRESULT MicrophoneNotServer([in] short newVal);
        [propget, id(dispidAddNewPoint), helpstring("property AddNewPoint")] HRES
```

```
    ULT AddNewPoint([out, retval] short *pVal);
            [propput, id(dispidAddNewPoint), helpstring("property AddNewPoint")] HRES
    ULT AddNewPoint([in] short newVal);
        };
    [
        uuid(E254A2B4-B470-11D2-8455-00104B05249C),
        version(1.0),
        helpstring("WebEcgControl 1.0 Type Library")
    ]
    library WEBECGCONTROLLib
    {
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

        [
            uuid(E254A2C2-B470-11D2-8455-00104B05249C),
            helpstring("WebEcg Class")
        ]
        coclass WebEcg
        {
            [default] interface IWebEcg;
        };


        [
            uuid(3D002C02-B9E0-11D2-8455-00104B05249C),
            helpstring("WebEcgPPG Class")
        ]
        coclass WebEcgPPG
        {
            interface IUnknown;
        };
    };
```

```
// WebEcgControl.cpp : Implementation of DLL Exports.


// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f WebEcgControlps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include "initguid.h"
#include "WebEcgControl.h"

#include "WebEcgControl_i.c"
#include "WebEcg.h"
#include "WebEcgPPG.h"


CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
    OBJECT_ENTRY(CLSID_WebEcg, CWebEcg)
    OBJECT_ENTRY(CLSID_WebEcgPPG, CWebEcgPPG)
END_OBJECT_MAP()

/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID /*lpReserved*/)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        _Module.Init(ObjectMap, hInstance);
        DisableThreadLibraryCalls(hInstance);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
        _Module.Term();
    return TRUE;    // ok
}

/////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
    return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    return _Module.GetClassObject(rclsid, riid, ppv);
}

/////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
    // registers object, typelib and all interfaces in typelib
    return _Module.RegisterServer(TRUE);
```

```
}

////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system registry

STDAPI DllUnregisterServer(void)
{
    _Module.UnregisterServer();
    return S_OK;
}



// resource.h

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by WebEcgControl.rc
//
#define IDS_PROJNAME                    100
#define IDR_WEBECG                      101
#define IDS_TITLEWebEcgPPG              102
#define IDS_HELPFILEWebEcgPPG           103
#define IDS_DOCSTRINGWebEcgPPG          104
#define IDR_WEBECGPPG                   105
#define IDD_WEBECGPPG                   106
#define IDC_FORE_WIDTH                  201
#define IDC_FILENAME                    202
#define IDC_EDIT2                       203
#define IDC_GRID_STYLE                  204
#define IDC_MAX_MV                      205
#define IDC_MIN_MV                      206
#define IDC_STEP_MV                     207
#define IDC_DISPLAY_SECONDS             208
#define IDC_VOLUME_THRESHOLD            209

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        201
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         210
#define _APS_NEXT_SYMED_VALUE           107
#endif
#endif
```

```
// stdafx.h : include file for standard system include files,
//      or project specific include files that are used frequently,
//      but are changed infrequently

#if !defined(AFX_STDAFX_H__E254A2B8_B470_11D2_8455_00104B05249C__INCLUDED_)
#define AFX_STDAFX_H__E254A2B8_B470_11D2_8455_00104B05249C__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define STRICT


#define _WIN32_WINNT 0x0400
#define _ATL_APARTMENT_THREADED


#include <atlbase.h>
//You may derive a class from CComModule and use it if you want to override
//something, but do not change the name of _Module
extern CComModule _Module;
#include <atlcom.h>
#include <atlctl.h>
#include <ocidl.h>  // Added by ClassView

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately bef
ore the previous line.

#endif // !defined(AFX_STDAFX_H__E254A2B8_B470_11D2_8455_00104B05249C__INCLUDED)
```

```
// stdafx.cpp : source file that includes just the standard includes
//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>
#include <statreg.cpp>
#endif

#include <atlimpl.cpp>
#include <atlctl.cpp>
#include <atlwin.cpp>
```

Claims

1. A system for generating and transferring medical data in real time, comprising:

   a sensor (8) operable to provide an acoustical signal in simultaneous response to a biological function or condition;
   a microphone (14), located with the sensor, to receive the acoustical signal as the acoustical signal is provided from the sensor; and
   a computer (16) connected to the microphone and adapted to transfer over the Internet, in real-time response to the microphone receiving the acoustical signal, electric signals representative of the received acoustical signal to a recipient computer (18,20) located at a data communication service provider facility.

2. A system as defined in claim 1, wherein the sensor (8) includes a hand-held transducer (30) that converts energy from an adjacent beating heart of a patient into the acoustical signal.

3. A system as defined in claim 2 wherein the computer (16) is a personal computer including: a microprocessor circuit (42) that has a microprocessor which operates at a nominal speed of at least twenty megahertz, and a microphone

interface circuit (40) connected to the microphone (14) and the microprocessor circuit.

4. A system as defined in claims 1 to 3, wherein the personal computer (16) is programmed with a web browser and a medical data acquisition and transmission program.

5. A system as defined in claim 4 when dependent from claim 3, wherein the medical data acquisition and transmission program is downloaded into the personal computer (16) from an Internet site accessed using the web browser.

6. A system as defined in claim 4 when dependent from claim 1, wherein the medical data acquisition and transmission program is from an Internet site accessed using the web browser.

7. A method of generating and transferring medical data in real time, the method comprising:

operating a sensor (8) to provide an acoustical signal in simultaneous response to a biological function or condition;
receiving the acoustical signal at a microphone (14) as the acoustical signal is provided from the sensor; and
operating a computer (16) connected to the microphone to transfer over the Internet, in real-time response to the microphone receiving the acoustical signal, electrical signals representative of the received acoustical signal to a recipient computer (18,20) located a data communication service provider facility.

8. The method of claim 7, wherein operating the sensor (8) comprises operating a hand held transducer (30) that converts energy from an adjacent beating heart of a patient into the acoustical signal.

9. The method of claim 7, wherein operating the personal computer (16) comprises operating a web browser and a medical data acquisition and transmission program.

10. The method of claim 9, comprising downloading into the personal computer (16) or accessing the medical data acquisition and transmission program from an Internet site using the web browser.

Patentansprüche

1. System zum Erzeugen und Übertragen medizinischer Daten in Echtzeit, aufweisend:

einen Sensor (8), der geeignet ist, um in simultaner Reaktion auf eine biologische Funktion oder Bedingung ein akustisches Signal auszugeben;
ein zusammen mit dem Sensor angeordnetes Mikrofon (14) zum Empfangen des akustischen Signals, während das akustische Signal von dem Sensor ausgegeben wird; und
ein mit dem Mikrofon verbundener Computer (16), der dazu eingerichtet ist, auf das das akustische Signal empfangende Mikrofon ansprechend elektrische Signale, die das empfangene akustische Signal kennzeichnen, in Echtzeit über das Internet an einen Empfängercomputer (18, 20) zu übertragen, der bei einer Einrichtung eines Datenübertragungs-Serviceproviders angeordnet ist.

2. System nach Anspruch 1, wobei der Sensor (8) einen von Hand gehaltenen Wandler (30) enthält, der Energie, die von dem benachbarten schlagenden Herzen eines Patienten ausgeht, in das akustische Signal umwandelt.

3. System nach Anspruch 2, bei dem der Computer (16) ein PC ist, zu dem gehören: ein Mikroprozessorschaltkreis (42), der einen Mikroprozessor, der bei einer nominalen Geschwindigkeit von wenigstens zwanzig Megahertz arbeitet, und einen Mikrofonschnittstellenschaltkreis (40) aufweist, der an das Mikrofon (14) und den Mikroprozessorschaltkreis (42) angeschlossen ist.

4. System gemäß den Ansprüchen 1 bis 3, wobei der PC (16) mit einem Webbrowser und einem medizinischen Datenakquisitions- und Übertragungsprogramm programmiert ist.

5. System nach Anspruch 4, soweit abhängig von Anspruch 3, wobei das medizinische Datenakquisitions- und Übertragungsprogramm zu dem PC (16) von einer Internet-Site heruntergeladen wird, auf die mittels des Webbrowsers zugegriffen wird.

6. System nach Anspruch 4, soweit abhängig von Anspruch 1, wobei das medizinische Datenakquisitions- und Übertragungsprogramm von einer Internet-Site stammt, auf die mittels des Webbrowsers zugegriffen wird.

7. Verfahren zur Erzeugen und Übertragen medizinischer Daten in Echtzeit, wobei das Verfahren die Schritte aufweist:

Bedienen eines Sensors (8), um in simultaner Reaktion auf eine biologische Funktion oder Bedingung ein akustisches Signal zu erzeugen;
Empfangen des akustischen Signals bei einem Mikrofon (14), während das akustische Signal von dem Sensor ausgegeben wird; und
Bedienen eines Computers (16), der mit dem Mikrofon verbundenen ist, um auf das das akustische Signal empfangende Mikrofon ansprechend elektrische Signale, die das empfangene akustische Signal kennzeichnen, in Echtzeit über das Internet an einen Empfängercomputer (18,20) zu übertragen, der bei einer Einrichtung eines Datenübertragungs-Serviceproviders angeordnet ist.

8. Verfahren nach Anspruch 7, wobei zu der Bedienung des Sensors (8) der Schritt gehört, einen von Hand gehaltenen Wandler (30) zu bedienen, der Energie, die von dem benachbarten schlagenden Herzen eines Patienten ausgeht, in das akustische Signal umwandelt.

9. Verfahren nach Anspruch 7, wobei zu der Bedienung des PC (16) der Schritt gehört, einen Webbrowser und ein medizinisches Datenakquisitions- und Übertragungsprogramm zu bedienen.

10. Verfahren nach Anspruch 9, zu dem der Schritt gehört, das medizinische Datenakquisitions- und Übertragungsprogramm von einer Internet-Site mittels des webbrowsers auf den PC (16) herunterzuladen oder darauf zuzugreifen.

Revendications

1. Système pour générer et transférer des données médicales en temps réel, comprenant :

un capteur (8) utilisable pour fournir un signal acoustique en réponse simultanée à une fonction ou condition biologique ;
un microphone (14), situé avec le capteur, pour recevoir le signal acoustique quand le signal acoustique est fourni par le capteur ; et
un ordinateur (16) connecté au microphone et adapté pour transférer par Internet, en réponse en temps réel à la réception du signal acoustique par le microphone, des signaux électriques représentatifs du signal acoustique reçu vers un ordinateur destinataire (18, 20) situé dans un organisme de fournisseur de services de communication de données.

2. Système selon la revendication 1, dans lequel le capteur (8) comprend un transducteur à main (30) qui convertit l'énergie provenant du coeur battant d'un patient voisin en ledit signal acoustique.

3. Système selon la revendication 2, dans lequel l'ordinateur (16) est un ordinateur personnel comprenant un circuit à microprocesseur (42) qui comporte un microprocesseur qui fonctionne à une vitesse nominale d'au moins vingt mégahertz, et un circuit d'interface de microphone (40) connecté au microphone (14) et au circuit à microprocesseur.

4. Système selon les revendications 1 à 3, dans lequel l'ordinateur personnel (16) est programmé avec un navigateur Web et un programme d'acquisition et de transmission de données médicales.

5. Système selon la revendication 4 lorsqu'elle dépend de la revendication 3, dans lequel le programme d'acquisition et de transmission de données médicales est téléchargé dans l'ordinateur personnel (16) à partir d'un site Internet atteint en employant le navigateur Web.

6. Système selon la revendication 4 lorsqu'elle dépend de la revendication 1, dans lequel le programme d'acquisition et de transmission de données médicales provient d'un site Internet atteint en employant le navigateur Web.

7. Procédé pour générer et transférer des données médicales en temps réel, le procédé comprenant les opérations consistant à :

faire fonctionner un capteur (8) pour fournir un signal acoustique en réponse simultanée à une fonction ou condition biologique ;
recevoir le signal acoustique dans un microphone (14) quand le signal acoustique est fourni par le capteur ; et
faire fonctionner un ordinateur (16) connecté au microphone pour transférer par Internet, en réponse en temps réel à la réception du signal acoustique par le microphone, des signaux électriques représentatifs du signal acoustique reçu vers un ordinateur destinataire (18, 20) situé dans un organisme de fournisseur de services de communication de données.

8. Procédé selon la revendication 7, dans lequel l'utilisation du capteur (8) comprend le fait de faire fonctionner un transducteur à main (30) qui convertit l'énergie provenant du coeur battant d'un patient voisin en ledit signal acoustique.

9. Procédé selon la revendication 7, dans lequel l'utilisation de l'ordinateur personnel (16) comprend le fait d'employer un navigateur Web et un programme d'acquisition et de transmission de données médicales.

10. Procédé selon la revendication 9, comprenant le fait de télécharger dans l'ordinateur personnel (16) ou d'accéder au programme d'acquisition et de transmission de données médicales à partir d'un site Internet en utilisant le navigateur Web.

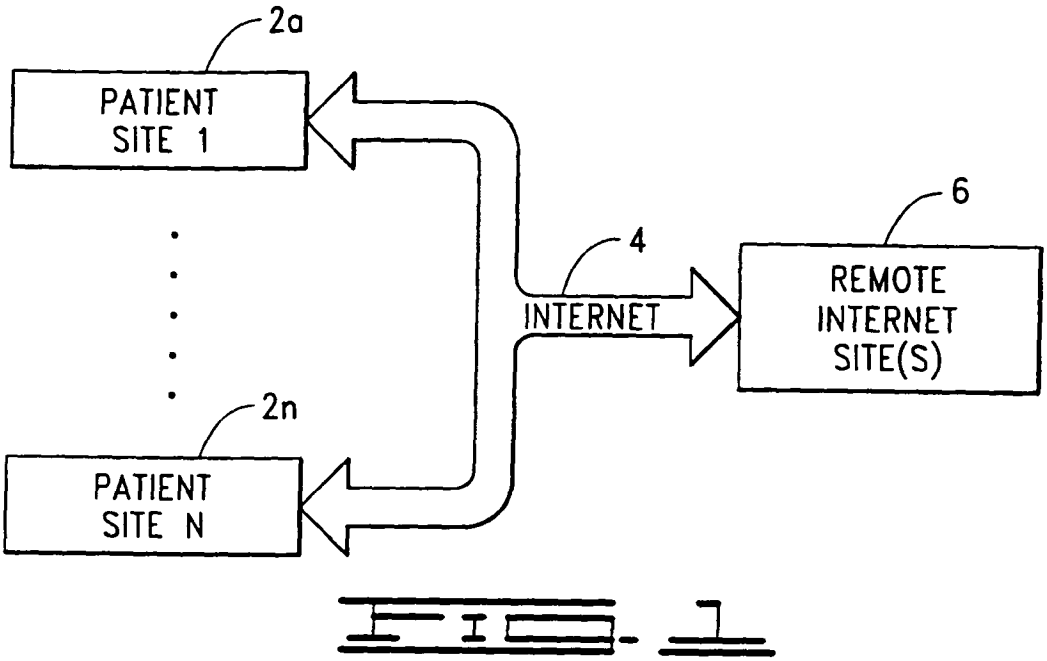

2a
PATIENT SITE 1
2n
PATIENT SITE N
4
INTERNET
6
REMOTE INTERNET SITE(S)

FIG. 1

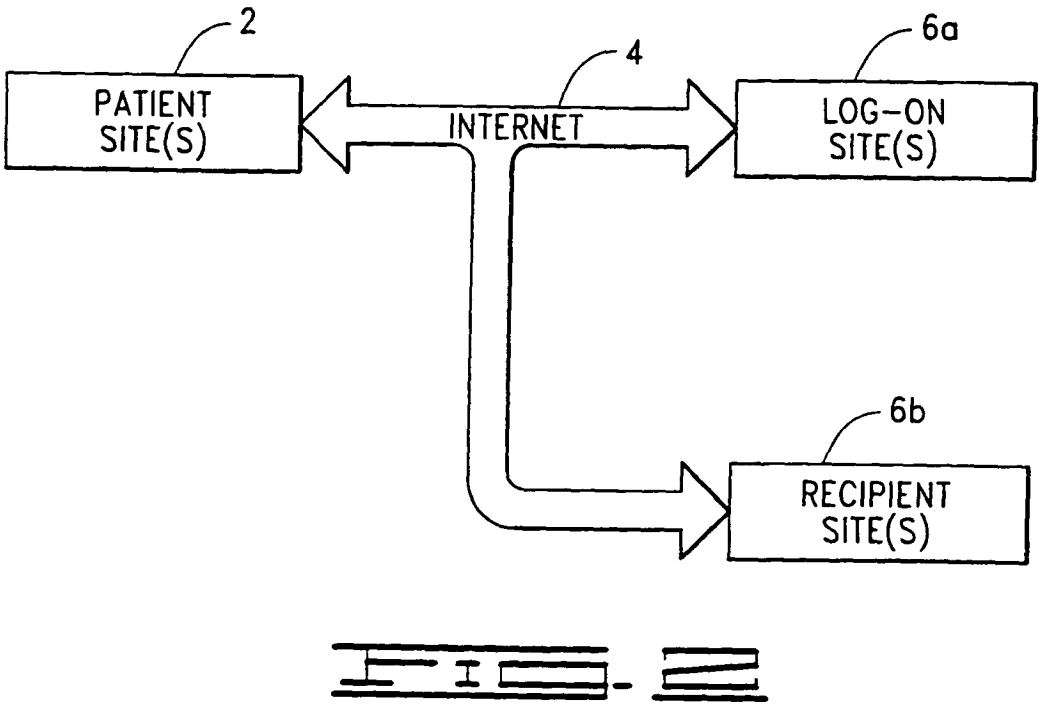

2
PATIENT SITE(S)
4
INTERNET
6a
LOG-ON SITE(S)
6b
RECIPIENT SITE(S)

FIG. 2

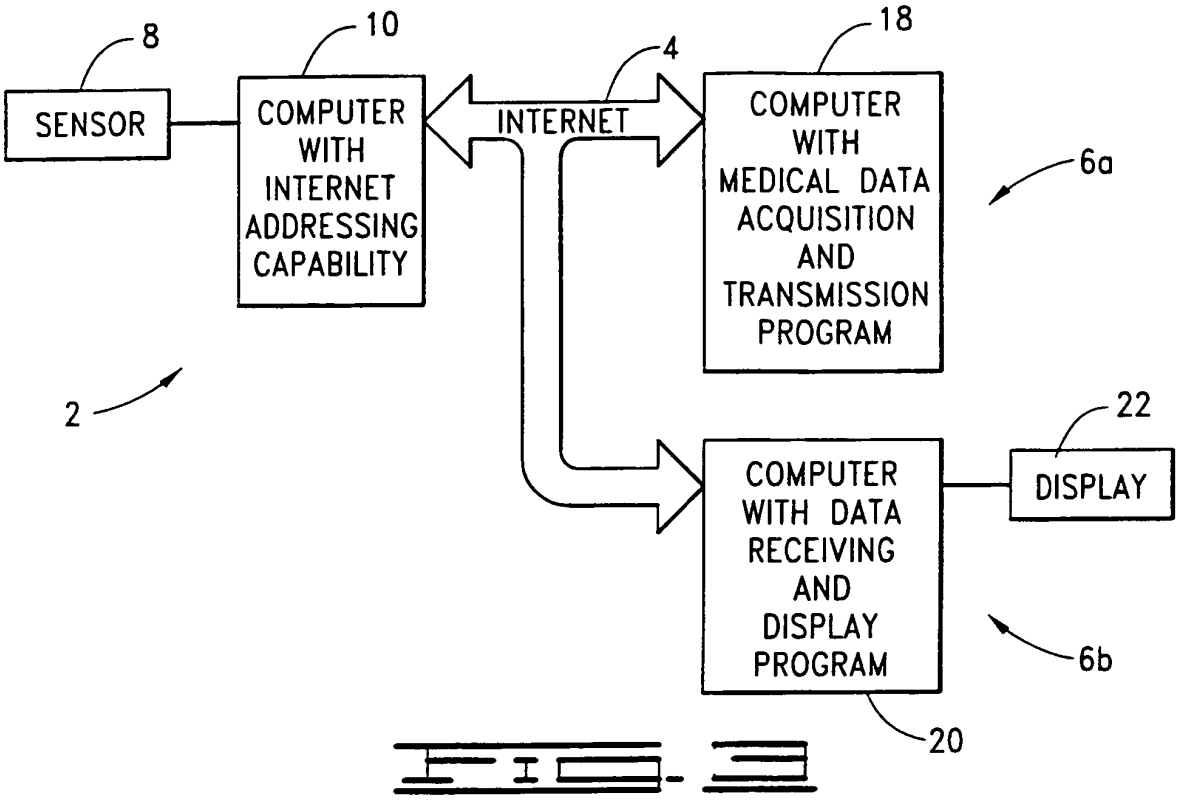

8
10
4
18
SENSOR
COMPUTER WITH INTERNET ADDRESSING CAPABILITY
INTERNET
COMPUTER WITH MEDICAL DATA ACQUISITION AND TRANSMISSION PROGRAM
6a
2
22
COMPUTER WITH DATA RECEIVING AND DISPLAY PROGRAM
DISPLAY
6b
20

FIG. 3

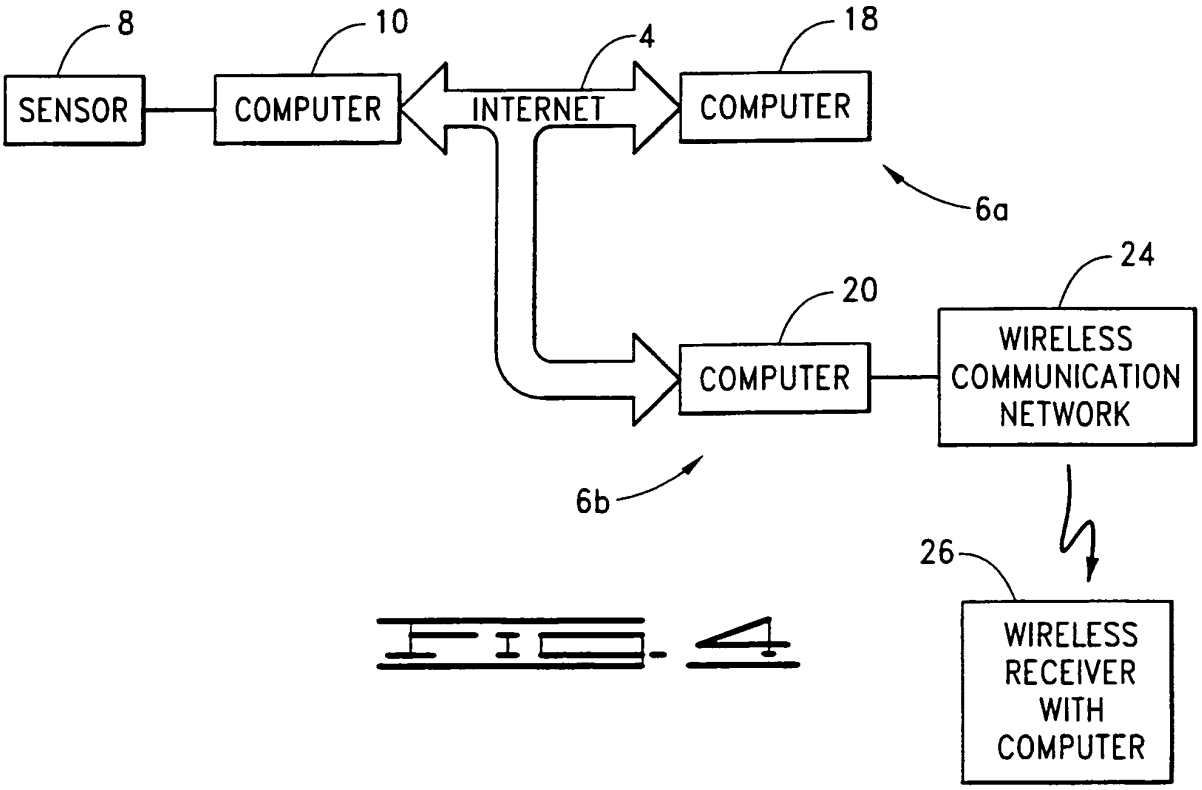

8
10
4
18
SENSOR
COMPUTER
INTERNET
COMPUTER
6a
24
20
COMPUTER
WIRELESS COMMUNICATION NETWORK
6b
26
WIRELESS RECEIVER WITH COMPUTER

FIG. 4

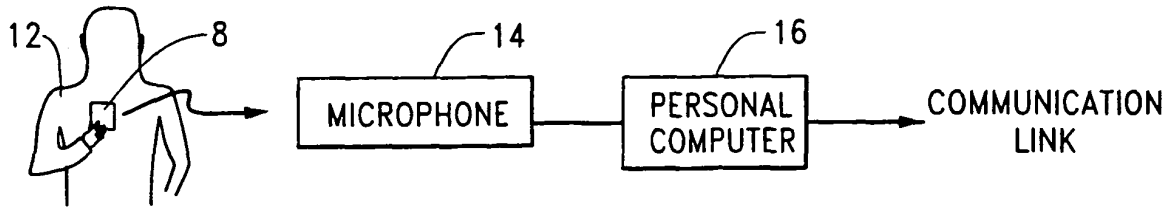

12
8
14
16
MICROPHONE
PERSONAL COMPUTER
COMMUNICATION LINK

FIG. 5

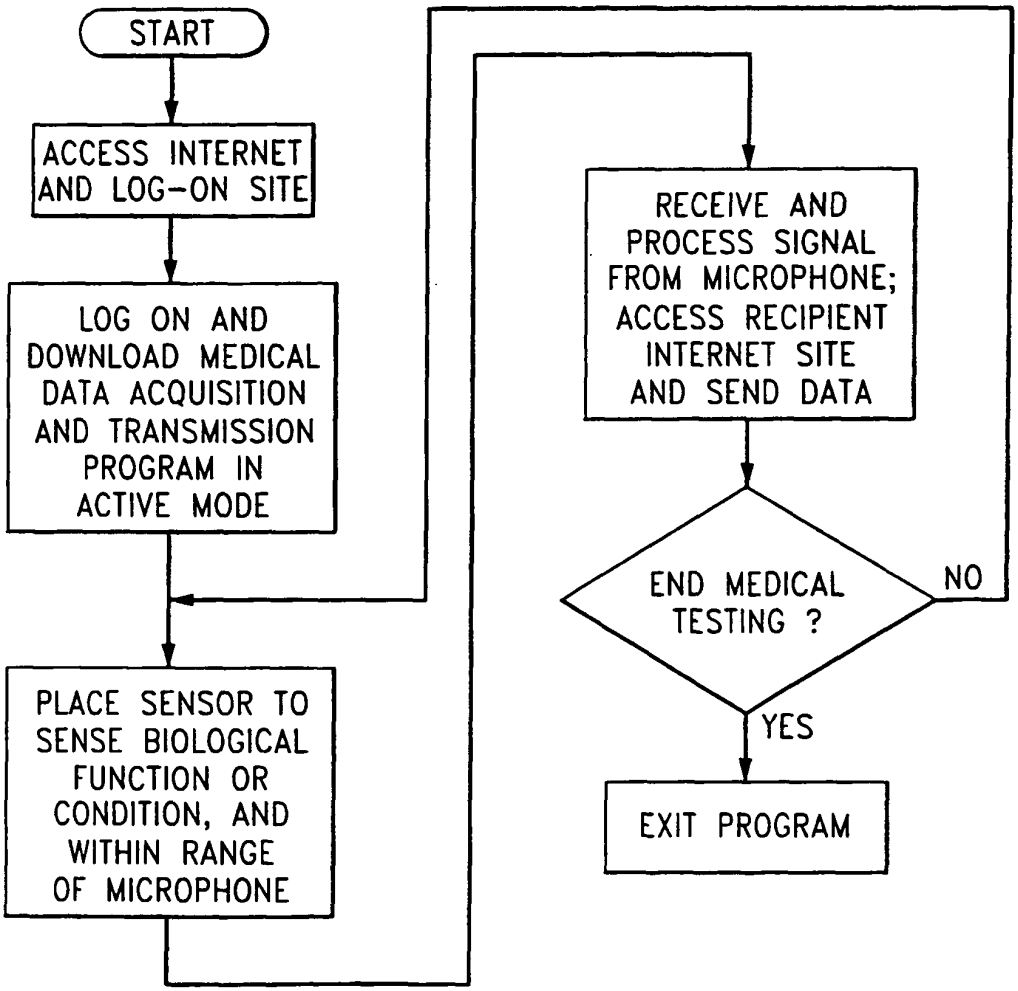

START
ACCESS INTERNET AND LOG-ON SITE
LOG ON AND DOWNLOAD MEDICAL DATA ACQUISITION AND TRANSMISSION PROGRAM IN ACTIVE MODE
PLACE SENSOR TO SENSE BIOLOGICAL FUNCTION OR CONDITION, AND WITHIN RANGE OF MICROPHONE
RECEIVE AND PROCESS SIGNAL FROM MICROPHONE; ACCESS RECIPIENT INTERNET SITE AND SEND DATA
END MEDICAL TESTING ?
NO
YES
EXIT PROGRAM

FIG. 6

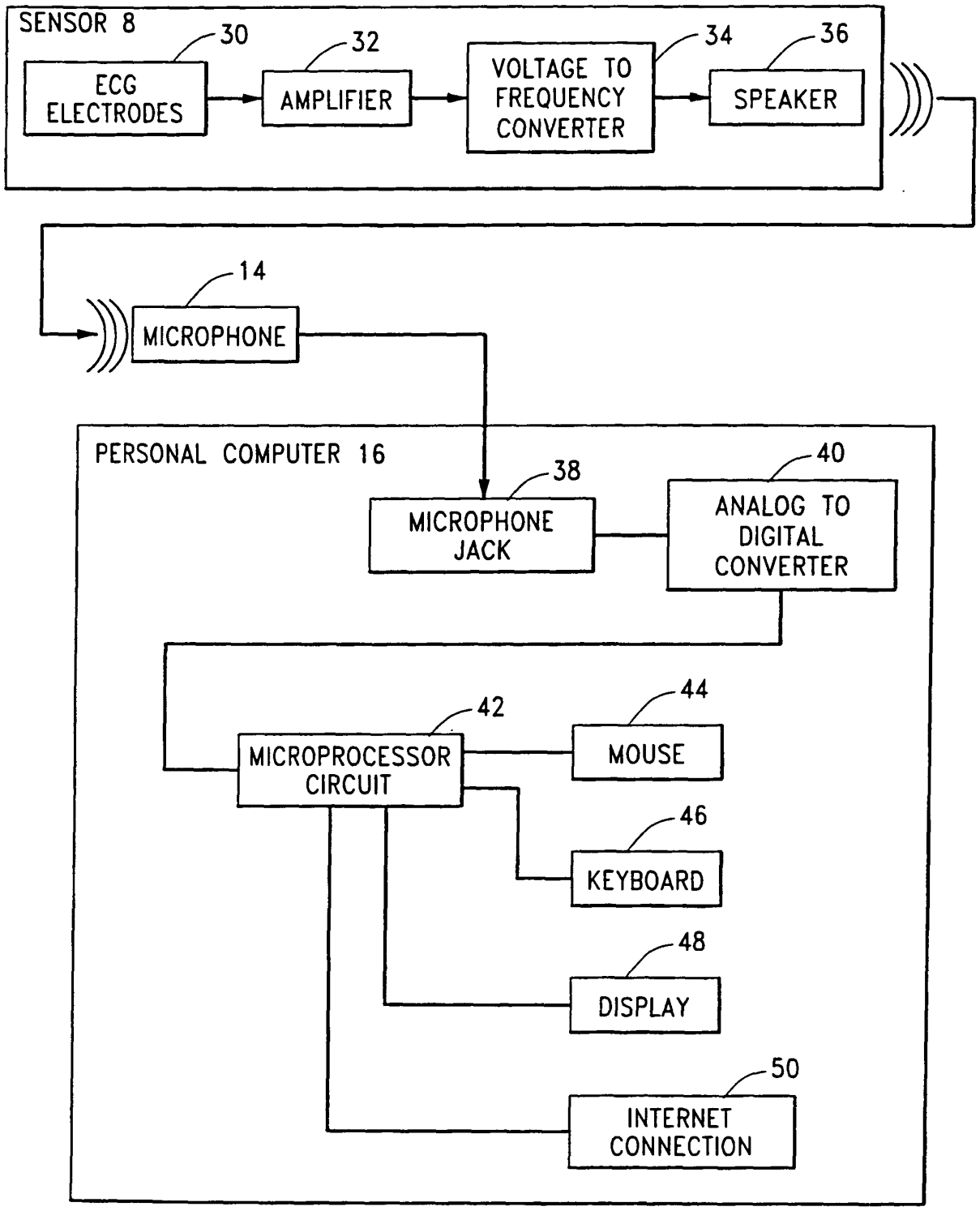
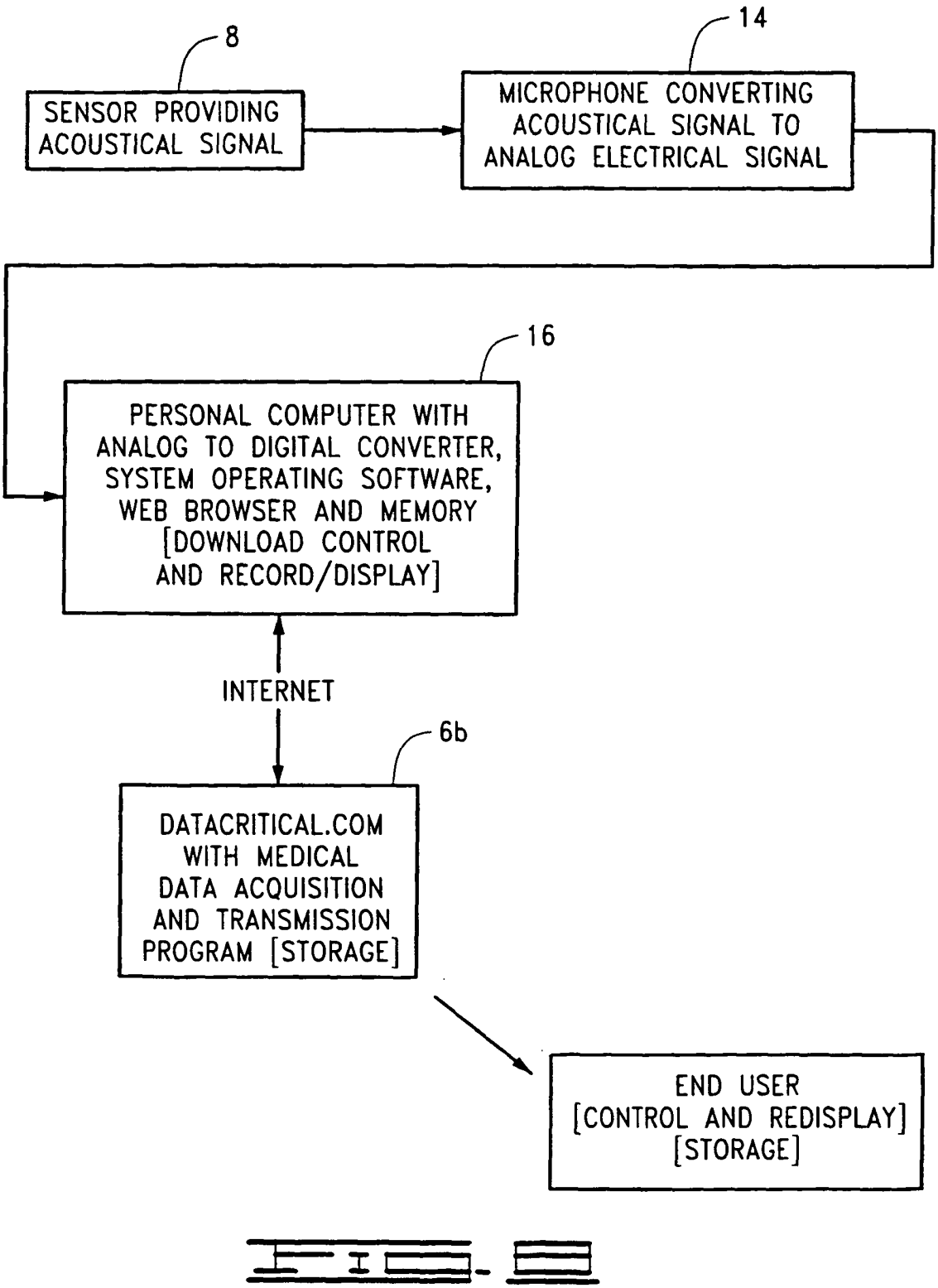
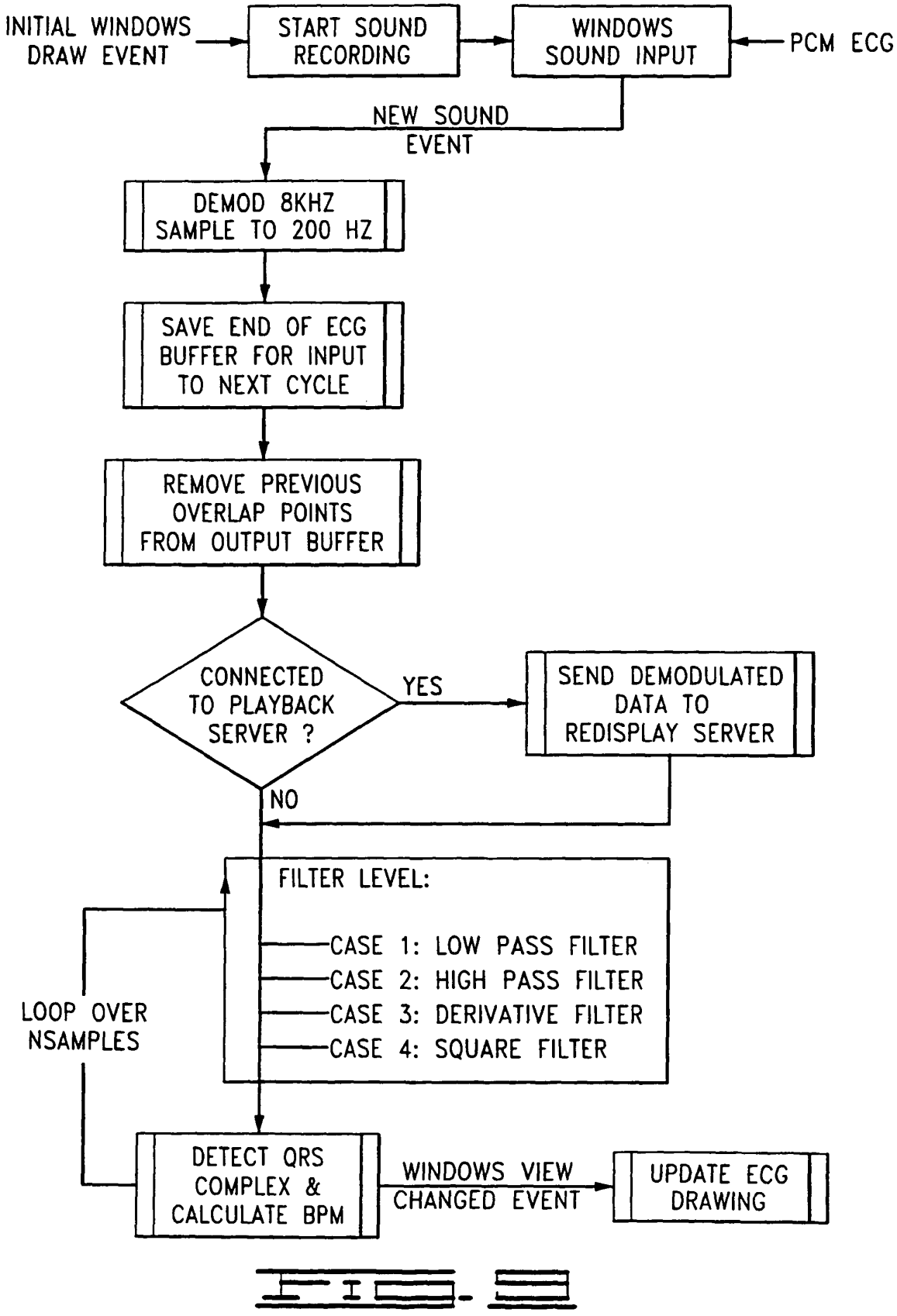
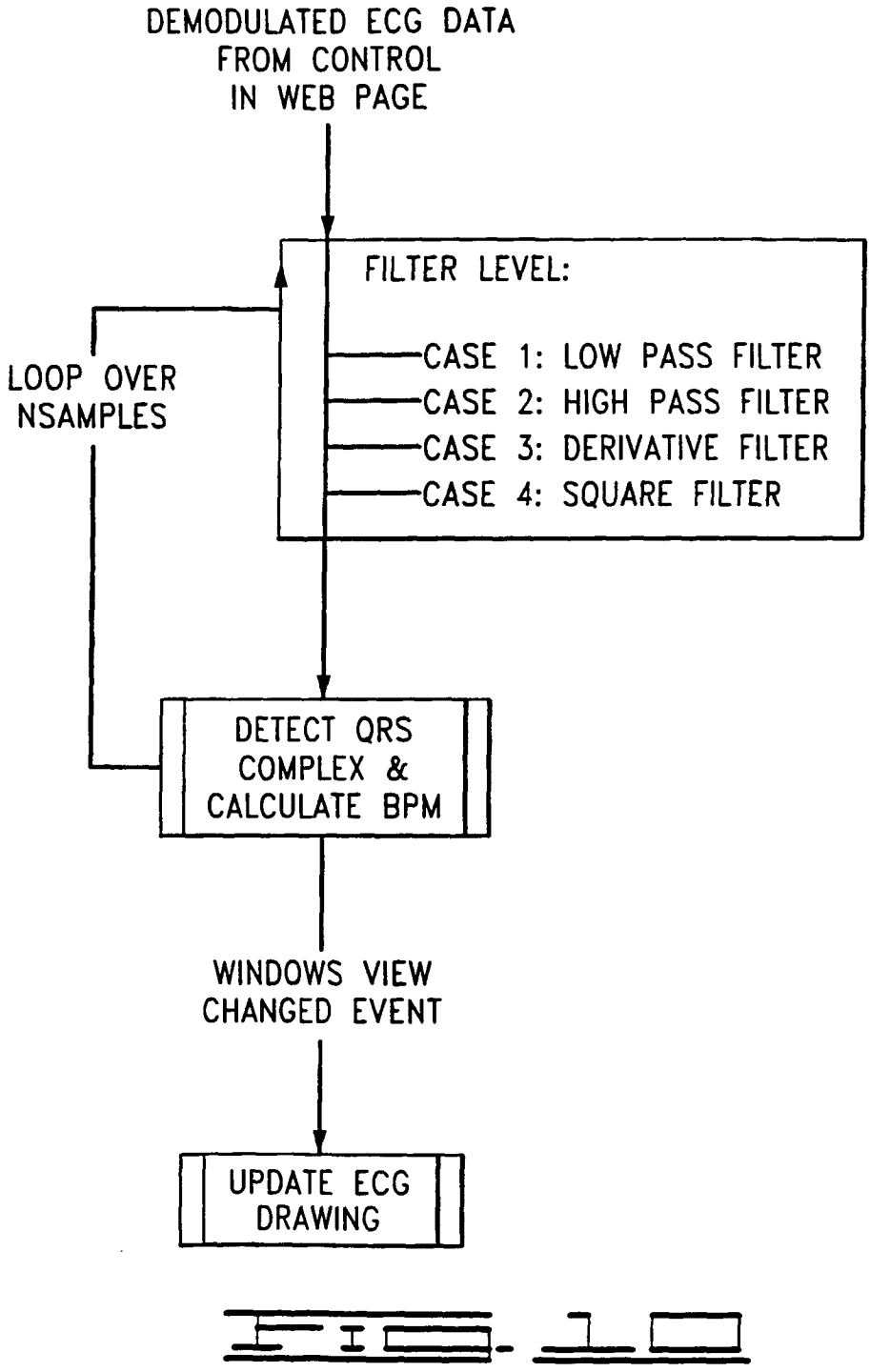

SENSOR 8
30
ECG ELECTRODES
32
AMPLIFIER
VOLTAGE TO FREQUENCY CONVERTER
34
36
SPEAKER
14
MICROPHONE
PERSONAL COMPUTER 16
38
MICROPHONE JACK
40
ANALOG TO DIGITAL CONVERTER
42
MICROPROCESSOR CIRCUIT
44
MOUSE
46
KEYBOARD
48
DISPLAY
50
INTERNET CONNECTION

FIG. 7


8
SENSOR PROVIDING ACOUSTICAL SIGNAL
14
MICROPHONE CONVERTING ACOUSTICAL SIGNAL TO ANALOG ELECTRICAL SIGNAL
16
PERSONAL COMPUTER WITH ANALOG TO DIGITAL CONVERTER, SYSTEM OPERATING SOFTWARE, WEB BROWSER AND MEMORY [DOWNLOAD CONTROL AND RECORD/DISPLAY]
INTERNET
6b
DATACRITICAL.COM WITH MEDICAL DATA ACQUISITION AND TRANSMISSION PROGRAM [STORAGE]
END USER [CONTROL AND REDISPLAY] [STORAGE]

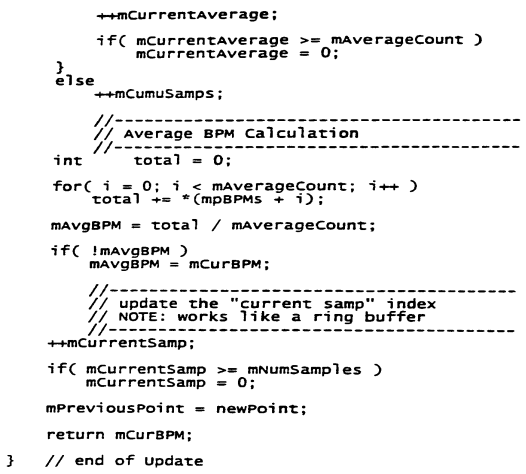

INITIAL WINDOWS DRAW EVENT
START SOUND RECORDING
WINDOWS SOUND INPUT
PCM ECG
NEW SOUND EVENT
DEMOD 8KHZ SAMPLE TO 200 HZ
SAVE END OF ECG BUFFER FOR INPUT TO NEXT CYCLE
REMOVE PREVIOUS OVERLAP POINTS FROM OUTPUT BUFFER
CONNECTED TO PLAYBACK SERVER ?
YES
SEND DEMODULATED DATA TO REDISPLAY SERVER
NO
FILTER LEVEL:
CASE 1: LOW PASS FILTER
CASE 2: HIGH PASS FILTER
CASE 3: DERIVATIVE FILTER
CASE 4: SQUARE FILTER
LOOP OVER NSAMPLES
DETECT QRS COMPLEX & CALCULATE BPM
WINDOWS VIEW CHANGED EVENT
UPDATE ECG DRAWING


DEMODULATED ECG DATA
FROM CONTROL
IN WEB PAGE
FILTER LEVEL:
CASE 1: LOW PASS FILTER
CASE 2: HIGH PASS FILTER
CASE 3: DERIVATIVE FILTER
CASE 4: SQUARE FILTER
LOOP OVER
NSAMPLES
DETECT QRS
COMPLEX &
CALCULATE BPM
WINDOWS VIEW
CHANGED EVENT
UPDATE ECG
DRAWING

专利名称(译)	用于生成和传输数据的系统和方法		
公开(公告)号	EP1210006B1	公开(公告)日	2007-06-06
申请号	EP2000959696	申请日	2000-08-31
[标]申请(专利权)人(译)	数据的关键		
申请(专利权)人(译)	数据的关键CORPORATION		
当前申请(专利权)人(译)	数据的关键CORPORATION		
[标]发明人	ALBERT DAVID E RIEGER CARL J REITER MAC L SLAVEN RIK		
发明人	ALBERT, DAVID, E. RIEGER, CARL, J. REITER, MAC L. SLAVEN, RIK		
IPC分类号	A61B5/00 G06Q10/00 G06Q50/00		
CPC分类号	A61B5/0022 A61B2560/0271 G06F19/3418 G16H40/67 Y10S128/904		
优先权	09/387013 1999-08-31 US		
其他公开文献	EP1210006A1 EP1210006A4		
外部链接	Espacenet		

摘要(译)

计算机站点包括数据生成源。源可以包括可以由患者操纵以感测生物功能或状况（例如心跳）的装置（例如，心脏监测器）。设备响应监控条件输出声音信号。可以由现场的患者或其他初始用户操作的计算机运行程序，该程序处理响应于通过连接到计算机的麦克风接收的可听信号而产生的电信号。通过计算机通信网络（例如因特网）下载或以其他方式访问该程序。计算机通过该网络发送结果数据信号。使用多个前述组件，可以访问计算机通信网络的任何数量的患者（或其他用户）可以向他们的个人医疗护理提供者（或其他人）提供关于他们的个人医疗状况（或其他生成的数据）的实时信息。远程终端用户）。



```
        ++mCurrentAverage;
        if( mCurrentAverage >= mAverageCount )
            mCurrentAverage = 0;
    }
    else
        ++mCumuSamps;

        //-----------------------------------------
        // Average BPM Calculation
        //-----------------------------------------
    int     total = 0;
    for( i = 0; i < mAverageCount; i++ )
        total += *(mpBPMs + i);
    mAvgBPM = total / mAverageCount;
    if( !mAvgBPM )
        mAvgBPM = mCurBPM;

        //-----------------------------------------
        // update the "current samp" index
        // NOTE: works like a ring buffer
        //-----------------------------------------
    ++mCurrentSamp;
    if( mCurrentSamp >= mNumSamples )
        mCurrentSamp = 0;
    mPreviousPoint = newPoint;
    return mCurBPM;
}   // end of Update
```