BrainBay - Developer Manual

Version 2.4, 2019-04-27

Author: Christoph Veigl, webpage: http://brainbay.lo-res.at

Contents:

1. Introduction

This Manual provides information about the development environment and the internal structures of the BrainBay application. It is intended to support software developers in understanding and extending BrainBay. After a brief description of programming principles and important classes and functions, an example for adding a new processing element is given. As an appendix, the transmission protocols of Open-EEG firmware and the EDF- data format are shown.

2. System Requirements and Programming Environment

2.1 Hardware

The application was developed on a laptop computer running Windows-7. It is confirmed that compilation and execution also works on older Windows variants like Window 98, Windows 2000, XP, Vista and Windows 8/8.1.
Wine can be used to run BrainBay on Linux-based systems as on Macs with OS-X. Alternatively, VirtualBox or VMWare virtual machine images with WinXp etc. can be used on all platforms. Hardware requirements are low: computers with Pentium II CPU onwards and about 512 MB Ram should be fine to run BrainBay.

2.2 Development Tools

The source code is written in C++, using the *Win32* API for system calls and graphical user interface. BrainBay is a plain *Win32* - Application, it does not need the *MFC*-framework. *Microsoft Visual Studio 2010* was used as development environment. With this IDE, the user-dialogs can be drawn and compilation and debugging are quite comfortable. The free express version of the IDE can be used for building the application from sources (GUI design is not supported by the express version).

The project-file *brainbay.sln* is located in the source directory and can be read by Visual Studio. It contains information about the relevant source-files (.cpp), header-files (.h) and resource-components like dialogs and menus (.rc-file).

When a freeware development environment is desired, the Minimalis GNU for Windows (*MinGW*) -framework can be used (see http://www.mingw.org). MinGW is a collection of freely available and freely distributable Windows specific header files and import libraries combined with GNU toolsets that allow one to produce native Windows programs that do not rely on any 3rd-party C runtime DLLs. Please note that the MinGW build scripts are not up-to-date. The supported toolchain is Visual Studio 2010, the free express version is fine as long as you don’t need the GUI designer.

3. Libraries and Compilation Options

3.1 Static Libraries for linking the executabe file:

As BrainBay is a growing project which includes functions of other OpenSource developments, there are a number of libraries that have to be included for a successful linking process. In *VisualStudio*, use the *Project -> Settings -> Linker* dialog to view and modify these static libraries:

3.1.1 Standard libraries provided by the Visual Studio IDE

Following libraries provide functions for message handling, communication and user interfacing in a standard Win32 - application:

kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib comctl32.lib

additional standard libraries needed for multimedia and OpenGl -support:

winmm.lib, *vfw.lib, opengl32.lib glu32.lib glaux.lib*

3.1.2 Non-standard Open-Source third-party libraries:

The non standard libraries provide additional function like low-level multimedia support, mathematical calculations or image processing. The .lib - files have to be copied into the /lib directory of the programming environment, e.g. *'visual_studio/vc98/lib*' :

*SDL.lib, SDL_net.lib, SDL_sound.lib, modplug.*lib :
Simple Direct Media Layer (http://www.libsdl.org)

fidlib.lib : Jim Peter's Filter Library (http://uazu.net/fidlib)

matheval.lib : Gnu math-evaluator library
http://www.gnu.org/software/libmatheval/manual/libmatheval.html

The Open Computer Vision Library - OpenCV (version 2.4.2. pre-compiled .lib files are available in the folder “ComputerVision”) and the videoinput library
http://www.opencv.org https://github.com/ofTheo/videoInput

skinstyle.lib : SkinStyle- Win32 Skinning Library by John Roark
http://www.codeproject.com/dialog/skinstyle.asp

3.1.3 Non-standard commercial third party libraries

These libraries are not needed for compiling/linking brainbay in standard configuration. They provide access to the matlab engine which is used by the matlab-element to transfer data to and from matlab:

libeng.lib libmx.lib : Matlab engine (http://www.mathworks.com)

These libraries are part of the matlab distribution and can be copied from there to the /lib directory of the programming IDE, if the matlab-element will is used by defining the MATLAB_RELEASE constant in ob_matlab.h.

3.1.4 Dlls / Runtime library files

The Dlls reside in the same directory as the executable file. They are loaded at runtime and have to match the calls declared in the static libraries / function prototypes. BrainBay needs the following .DLLs:

SDL.dll, SDL_net.dll, SDL_sound.dll: for the Simple Direct Media Layer

3.2 Compiling/Linking-information for Visual Studio 2010 (C++, Win32-Debug):

Preprocessor Options:

WIN32,_DEBUG,_WINDOWS,_MBCS

VS2010 Compiler Options:

/I"ComputerVision\opencv\include" /I"ComputerVision" /I"neurobit_api" /ZI /nologo /W1 /WX- /Od /Oy- /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_CRT_SECURE_NO_WARNINGS" /D "_VC80_UPGRADE=0x0710" /D "_MBCS" /Gm- /EHsc /RTC1 /MTd /GS /fp:precise /Zc:wchar_t /Zc:forScope /Fp".\Debug/brainBay.pch" /Fa".\Debug/" /Fo".\Debug/" /Fd".\Debug/" /Gd /analyze- /errorReport:queue

VS2010 Linker Options:

/OUT:".\Debug/brainBay.exe" /INCREMENTAL:NO /NOLOGO /LIBPATH:"libmsvc" /LIBPATH:"ComputerVision\opencv\lib" /LIBPATH:"ComputerVision\videoInput\lib" "odbc32.lib" "odbccp32.lib" "comctl32.lib" "winmm.lib" "opengl32.lib" "glu32.lib" "SDL.lib" "SDL_net.lib" "matheval.lib" "SDL_sound.lib" "modplug.lib" "vfw32.lib" "glaux.lib" "skinstyle.lib" "ole32.lib" "strmiids.lib" "uuid.lib" "kernel32.lib" "user32.lib" "gdi32.lib" "winspool.lib" "comdlg32.lib" "advapi32.lib" "shell32.lib" "oleaut32.lib" /NODEFAULTLIB:"libcd.lib" /NODEFAULTLIB:"libc.lib" /MANIFEST /ManifestFile:".\Debug\BrainBay.exe.intermediate.manifest" /ALLOWISOLATION /MANIFESTUAC:"level='asInvoker' uiAccess='false'" /DEBUG /PDB:".\Debug/brainBay.pdb" /SUBSYSTEM:WINDOWS /PGD:"C:\data\works\EEG\BrainBayVS2010\Debug\BrainBay.pgd" /TLBID:1 /DYNAMICBASE:NO /MACHINE:X86 /ERRORREPORT:QUEUE

3.3 Compiliation & Linking -Information for MinGW:

Compiling/Linking-information for MinGW: (contributed by Jeremy Wilkerson).
Here are the steps to build the application (please note that the MinGW build is not officially supported – it works but maybe some elements must be excluded from the build):

-use the make to compile and link just the main source files
-use the make-all.bat file to compile and link everthing, including matheval library

The resource file can be compiled seperately using windres -i brainBay.rc -o brainBayRes.o

Following compile switches are used: g++ -c -DWIN32 -D_DEBUG -D_WINDOWS -D_MBCS -DMINGW *.cpp

the -DMINGW define tells the preprocessor to bypass the skinstyle library for the skinned dialog interface because this is only supported by the MSVC++ compiler by now. If mingw\bin is on your path, use the command 'make depends', then 'make'. The 'make depends' generates a file that tells make which header files each cpp file depends on, and it doesn't need to be executed again unless that information changes.

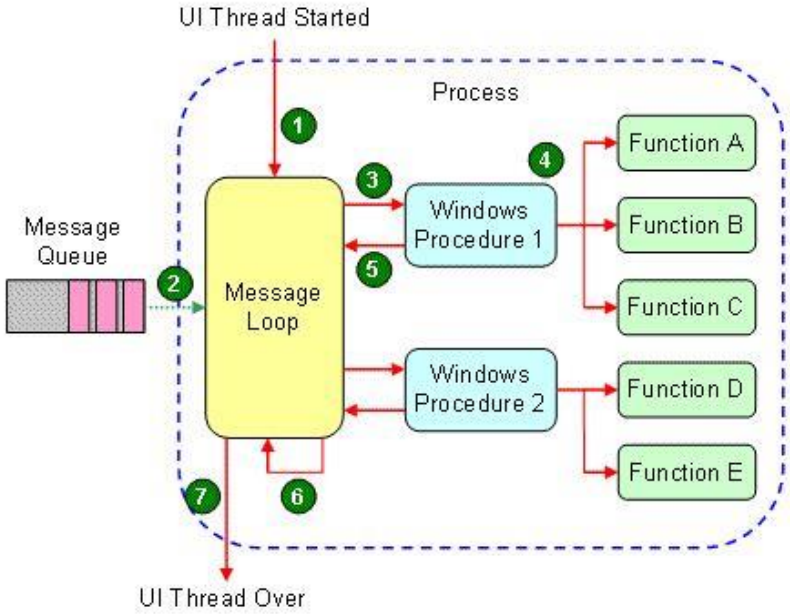
4. BrainBay Development

4.1 Basic Principles of Windows Programming

BrainBay is an event-driven Win32-Application. User interaction is provided by dialog windows, the processing of user interaction resides in the corresponding window handlers. The coordination of events like keyboard- and mouse inputs in a Windows-environment is done by processing *messages*. A message consists of the *message identifier* (16 - bit integer) and two message parameters (*wParam, lParam*, both 32-bit integers). If an event occurs, such as the user typing on the keyboard, moving the mouse, clicking a button, then messages are sent by the system to the windows affected. The *message-loop* holds pending messages that have to be processed. Beside initalisation tasks, the main-function of the application processes the message loop, by querying the message queue and calling the *translate()* and *displatch()* functions:

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
```

Messaging in a Windows Application:



Processing Messages in a simple Window-Handler:

```
LRESULT CALLBACK WndProc(
   HWND hWnd,                          //  the handle to the Window
   UINT message,                       //  the message identifier
   WPARAM wParam,                      //  first 32 bit parameter
   LPARAM lParam)                      //  second 32 bit paramte
{
   int wmId, wmEvent;

   switch (message)
   {
        case WM_COMMAND:
           wmId    = LOWORD(wParam);
           switch (wmId)
           {
              case IDM_ABOUT:
                        DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX,
                                  hWnd, (DLGPROC)About);
                        break;

              case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;

              default:  return DefWindowProc(hWnd, message,wParam, lParam);
           }
           break;
         default:  return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
```

In this example, only the WM_COMMAND message is processed: a dialog box is displayed when a button referenced by the constant IDM_ABOUT is pressed. This constant is passed to the handler in the lower 16-bit of the wParam - Parameter. When IDM_EXIT is received, the window will be closed.

When a window or dialog has more controls, processes mouse functions or needs painting routines, a variety af messages are sent to the window handler function. Some examples:

WM_INITDIALOG: when a dialog is created
WM_CLOSE: when a dialog or window is closed
WM_HSCROLL, WM_VSCROLL: when a scrollbar is changed
WM_MOUSEMOVE: when the mouse moves over the window
WM_LBUTTONDOWN, WM_LBUTTONUP: left mouse button pressed or released
WM_RBUTTONDOWN, WM_LBUTTONDOWN: right mouse button pressed or released
WM_DBLCLICK: left double click into the window or control
WM_PAINT: when the window is to be painted
WM_ACTIVATE: when the window is activated

A thread that contains a message loop is also known as a user-interface thread. A user-interface thread is associated with one or more windows which are created in that thread. The thread is often said to own these windows. The window procedure for a window is called only by the thread that owns the window.

This happens when DispatchMessage() is called inside the thread. Any thread may send or post a message to any window but the window procedure of the target window will only be executed by the owning thread. The end result is that all messages to a target window are synchronized. That is, the window is guaranteed to receive and process messages in the order in which the messages are sent/posted. For creating new windows, the *window class* has to be registered using the *WNDCLASSEX* - structure and the RegisterClassEx(..) function. The structure holds a pointer to the window-handler function, a handle to the instance (the executable file), resource-identifiers for icons, cursor type and background color and the class name.

The CreateWindow(..) - function builds a new window, referring to a given window-class, various window styles, a window title and position.
For an explaination of parameters and details of Windows programming, please refer to http://www.winprog.org or http://msdn.microsoft.com.

4.2 BrainBay - element base class

The main functional parts of BrainBay are the design elements. Each element type has it's own class that is derived from the BASE_CL - class. The BASE_CL - class provides the basic structure for elements:

- input- and output - ports
- methods for starting and stopping the session
- positioning in archives, getting archive length
- loading and saving properties
- receiving values and processing data.

The BASE_CL - class, defined in base.h :

```
class BASE_CL
{
  public:
        int type;                   //  object type
        int inports, outports;      //  number of in- and outports
        int xPos, yPos;             //  position of element in design window
        int width, height;          //  with and height of element in design window
        char tag[30];               //  element tag
        HWND displayWnd;            //  output window (if any)

        OUTPORTStruct out_ports[MAX_PORTS];
        //   name, description, range, dimension and value of an input-connection

        INPORTStruct  in_ports[MAX_PORTS];
        //   name, description, range and dimension of an output-connection

        LINKStruct  out[MAX_CONNECTS];
        //   source (objectnumber, portnumber) and destination of connection,
        //   range, dimension and description
        HWND hDlg;                  //  user dialog window
```

```
        BASE_CL (void) {};                          //  sets all ports and connection to default values
        virtual ~BASE_CL (void) {}                  //  deconstruction
        virtual void work (void) {}                     //  called when a new sample arrives
        virtual void update_inports (void) {}           //  when inports are connected or removed
        virtual void session_start (void) {}            // called when session is started
        virtual void session_stop (void) {}                     //   - " -              stopped
        virtual void session_reset (void) {}                    //  -  "  -               reset
        virtual void session_pos (long pos) {}                  //  new session position
        virtual long session_length (void) { }                  //  ask for archive length
        virtual void make_dialog (void) {}                      //  display the user dialog
        virtual void load (HANDLE hFile) {}                     //  load from configuration file
        virtual void save (HANDLE hFile) {}                     //  save to configuration file
        virtual void incoming_data(int port, float value) {}    //  reveice input on port

        void pass_values (int port, float value)   //  pass ouput to connected elements
};
```

As methods are declared as 'virtual' - functions, an element will implement only the methods that it needs. For example, if an element only does calculations of a value, it will not implement the method for archive-positioning etc.

4.3 Modules with common Functions and Global Variables

The following source- and header files provide globally accessible functions and structures for data processing, file operations, com-port control and dialog operations. This framework could also be used by new elements.

4.3.1 Base.h:

Definitions of the BASE_CL class (see above) and the maximum array sizes (elements, input- and output ports).

4.3.2 Brainbay.h

Definitions of global constans like element-numbers, element-names, file-types, default com-settings, file-modes, protocol-types. Declarations of global varibles and global structures: GlobalStruct, PacketStruct, CaptfileStruct, TTYStruct, TimingStruct, FliterStruct, EDFheaderStruct, EDFChannelStruct. For more information on the structure members see the header-file brainbay.h

4.3.3 Brainbay.cpp:

This modules holds the main function WinMain(...). It registers the window-classes, creates the main- and design-window, initialises the application and loads the initial design if option GLOBAL.startup is TRUE. The message queue is queried and dispatched in the message loop, where also the HotKeys are checked. The MainWndHandler(...) processes menu selections as insertion of new elements or loading design configurations.

4.3.4 Globals.cpp:

This Module stores global variables and functions for initialisation, de-initialisation and element-handling:

// arrays and variables for element handling:

```
BASE_CL * objects[MAX_OBJECTS];    // array of elements in the design
BASE_CL * actobject;               // active element
struct LINKStruct * actconnect;    // active connection
int PACKETSPERSECOND=DEF_PACKETSPERSECOND;  // sampling rate
```

// Window and dialog handles

```
HINSTANCE  hInst;                    //  instance of main class
HACCEL  ghAccel;                     //  keyboard accelerator
HWND   ghWndMain;                    //  handle of main window
HWND   ghWndToolbox;                 //  handle of toolbox window (element user dialogs)
HWND   ghWndStatusbox;               //  handle of Status Window
HWND   ghWndSettings;                //  handle of Settings Window
HWND   ghWndDesign;                  //  handle of Design Window
HWND   ghWndAnimation;               //  handle of Animation Window
HGLRC  GLRC_Animation;               //  handle to OpenGL-context of Animation Window
```

// global data, com-settings, timing

```
struct GLOBALStruct  GLOBAL;         //  global accessible parameters
struct TTYStruct   TTY;              //  Com Port settings
struct PACKETStruct  PACKET;         //  current biosignal packet
struct MIDIPORTStruct  MIDIPORTS[MAX_MIDIPORTS];  // accessible midi output devices
struct TIMINGStruct  TIMING;
```

// captions

```
char objnames[50][20]      = { OBJNAMES };          //  names of all elements
char dimensions[10][10]     = {"uV","mV","Hz","%" };  //  dimensions for connections
```

Functions defined in Globals.cpp:

void register_classes(hinstance)
registers all window classes

void GlobalInitialize(void)
sets up the SDL-framework, sets global variables and structures (path to the application, Com-port settings, window positions, status variables; Loads default settings, initializes timing, opens midi ports, creates drawing structures

void GlobalCleanup(void)
deconstructs all elements, quits SDL, closes midi ports, deletes drawing structures.

void create_object (int type)
creates a new element of given type. A pointer to the element is stored in the global *actobject* variable. The type is defined in brainbay.h as constant *OB_<element_name>.*

void free_object(int actobj)
deletes the element with index *actobj* from the design

void swap_objects(int a, int b)
exchanges the elements indices *a* and *b* for sorting purposes

int sort_objects(void)
sorts all elements in the design according to their connections (signal sources before signal targets)

void set_dimensions(struct LINKStruct * act,float max, min, char * dim, * desc)
sets new range, dimension and description for the given connection

void update_dimensions(void)
propagates the ranges and dimensions through all elements

void update_samplingrate(int newrate)
selects a new samplingrate for signal processing and re-initialises the concerned filter and magnitude elements

void get_session_length()

calculates duration of longest archive file in the current configuration and stores the value in *GLOBAL.session_end*;

void set_session_pos(long pos)
sets the current position of playback to sample number pos, calls the session_pos() method of all elements

float size_value (float min,max,x,to_min,to_max, int clip)
resizes value x from ranges (min, max) to ranges (to_min, to_max)

void reduce_filepath (char* to, char * from)
copies the filename without leading path-information to a new string

int count_inports(BASE_CL * obj)
returns the number of the greatest connected input port of the given object

reset_oscilloscopes()
clears all oscilloscope windows

4.3.5 Files.cpp:

This module holds file operation functions like archive file creation or reading, loading and saving configuration files, parsing EDF files etc.:

HANDLE create_captfile(LPCTSTR lpFName)
creates an archive-file for capturing live data from a biosignal amplifier in P2 or P3 protocol. Set the action in the CAPTFILE-structuce to 'writing' and returns a handle if successful or otherwise returns INVALID_HANLDE_VALUE.

void open_captfile(LPCTSTR lpFName)
opens an archive file for reading. If successful, the CAPTFILE-structure is filled with header data and filehandle, action is set to 'reading'. Displays an error message if the file cannot be opened.

void close_captfile(void)
closes an opened archive file, clears the CAPTFILE structure and updates the total session length.

void update_devicetype(void)
sets archive file reading and writing options according to the type of amplifier protocol (P2, P3) or archive format (P2, P3 or RAW).

void read_captfile(int amount)
reads *amount* bytes from an opened archive of given type file and transfers the bytes to the input buffer *TTY.readBuf*.

void read_captfile(int amount)
writes one byte to an openen archive file

int open_file_dlg(HWND hDlg, char * szFileName, int type, int flag_save)
displays a dialog for choosing a path and filename. *hDlg* is a handle to the calling window, szFilename gives the initial path and will store the new path, *type* selects the wildcards for displaying files and *flag_save* selects a 'open-' or 'create' - type of dialog.

BOOL load_from_file(LPCTSTR pszFileName, void * buffer, int size)
loads a buffer of given size from a file called *pszFileName.*

BOOL save_to_file(LPCTSTR pszFileName, void * buffer, int size)
saves a buffer of given size to a file called *pszFileName.*

BOOL load_configfile(LPCTSTR pszFileName)
loads a configuration file with the given name. If the file exists, a eventually runnig session is stopped and the current design is closed, deleting all elements. The settings for the new desing and all elements are imported from the configuration file. The design will be started when the *GLOBAL.autorun* option is TRUE.

BOOL save_configfile(LPCTSTR pszFileName)
creates a configuration file of given name and stores all global and elemtn data into this file. Returns TRUE is successful and FALSE otherwise.

BOOL save_settings(void)
saves applicatiion settings to the file *brainbay.cfg.* The settings include samplingrate, comport, baudrate, devicetype, refresh-intervals for dialogs and draw-windows, autorun and auto-load options, midi-ports and the current config-file.

BOOL load_settings(void)
loads the application setting from the file brainbay.cfg.

int load_next_config_buffer(HANDLE hFile)
reads strings from file hFile and sores them into *GLOBAL.configbuffer* until the line 'end object' is read. Using this function, a block of parameters can be read from a configuration or settings file.

void load_property(char * desc,int type, void * ad)
loads a variable value from the string buffer *GLOBAL.configbuffer*. This buffer is read from a configuration file when it is loaded. *Desc* is a string identifier for the variable, *type* selects integer (P_INT), float (P_FLOAT) or string (P_STRING) - data, *ad* is a pointer to the variable that will be filled when description/value is found in the string buffer.

void save_property(HANDLE hFile, char * desc,int type, void * ad)
stores a variable value in readable form to a file. *Desc* is a string identifier for the variable that will be written before the value, type selects integer (P_INT), float (P_FLOAT) or string (P_STRING) - data, ad is a pointer to the variable that will be written to the file.

void save_object_basics(HANDLE hFile, BASE_CL * actobj)
stores element position, number of input and output port, element tag and all element connections in readable form to the file hFile.
void load_object_basics(BASE_CL * actobj)
loads element position, number of input and output port, element tag and all element connections from the string buffer *GLOBAL.configbuffer*.

void parse_edf_header(EDFHEADERStruct * to, CHANNELStruct * tochn, char * from)
extracts EDF- header and channel information out of the string **from*

void generate_edf_header(char * to, EDFHEADERStruct * header, CHANNELStruct * channels)
parses EDF - header and channel information into string **to*

HANDLE open_edf_file(EDFHEADERStruct * to, CHANNELStruct * tochn, char * filename)
displays a file-open dialog and extracts EDF-header and channel information from the file, if valid. (returns: file handle if valid EDF-File, INVALID_HANDLE_VALUE if no valid file)

HANDLE create_edf_file(EDFHEADERStruct * from, CHANNELStruct * fromchn, char * filename)
displays a file-save dialog and puts header- and channels information into the chosen file (returns: file handle if valid EDF file, INVALID_HANDLE_VALUE if no valid file)

4.3.6 TTY.cpp

This module provides functions for COM-port handling and reader- and writer-threads for communication.

BOOL SetupCommPort(int port)
initialises and opens the COM - Port *port* (for example COM3 or COM7). 8 data bits, no parity and one stop bit is used, the baudrate is given by TTY.BAUDRATE. If the COM-Port can be sucessfully opened, the handle to the device will be stored in TTY.COMDEV and a reader thread will be created that processes the incoming data (ReaderProc). If an error occurs, a message will be display and the function returns FALSE.

BOOL BreakDownCommPort()
if a Com-Port is openend, the reader thread will be shut down and the COM-Port will be closed.

4.3.7 Timer.cpp

This module provides initialisation and de-initialisation of the timer-interrupt routine and timing-specific structures. If no COM-port reader thread is active, the timer interrupt controls the processing of packets through the elements.

void init_system_time(void)
queries the perfomance-counter (system counter) and selects *TTY.packettime* according to the current sampling rate (*PACKETSPERSECOND*). Resets packet counter and session time to zero.

void process_packets(void)
processes one packet through the elements: calls the woker methods of all elements, updates packet counter and the status bar.

void CALLBACK TimerProc(UINT uID,UINT uMsg,DWORD dwUser,DWORD dw1,DWORD dw2)
the timer interrupt function, it is called as often as possible (1 ms period). If the packet-time (1/*PACKETSPERSECOND*) has passed, process_packets () will be called to do the signal processing in the elements. If an archive file is opened, one packet will be read from there.

void start_timer(void)
enabled the timer interrupt routine

void stop_timer(void)
disables the timer interrupt routine

4.3.8 Dialogs.cpp

This module holds some dialog specific functions and dialog handlers for the design-window, the status bar, the options window and the color- and tonescale editors.

COLORREF select_color(HWND hwnd)
displays a color selection dialog provided by the system. The chosen color is returned.

void update_toolbox_position(HWND hDlg)
stores the current position of the toolbox window is into the GLOBAL structure

void display_toolbox(HWND hDlg)
the dialog hDlg is moved to the stored postion for the toolbox-window and shown as top layer window, the focus is given to the dialog.

void close_toolbox(void)
an eventually opened toolbox window is closed

int get_scrollpos(WPARAM wParam, LPARAM lParam)
can be used to update the position of horizontal or vertical silder bars that receive a message in their dialog window. The updated position will be returned.

void color_button(HWND hWnd, COLORREF newcolor)
sets the color of a window-element (like a button) to *newcolor*

LRESULT CALLBACK **COLORDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler and functions for the color editor

LRESULT CALLBACK **SCALEDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler and functions for the tonescale editor

LRESULT CALLBACK **SETTINGSDlgHandle**r(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): handler for the settings (options) - dialog

LRESULT CALLBACK **CONNECTDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler for the connection - info

LRESULT CALLBACK **OUTPORTDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler for the output - port settings

LRESULT CALLBACK **INPORTDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler for the input - port settings

LRESULT CALLBACK **TAGDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler for the element - tagging

LRESULT CALLBACK **StatusDlgHandler**(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam): dialog handler and functions for the status bar

LRESULT CALLBACK **DesignWndHandler**(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam): dialog handler for the design window
The Design Window Handler processes all mouse events that are necessary to arrange the elements in the design window, connect and delete links between elements, display user dialogs for the elements and scroll the design window. When new connections are established, the **sort_objects**() - function is called to ensure a valid sequence of the elements for processing.

void report(char * Message)
reports a message in a dialog box

void report_error(char * Message)
reports an error message in a dialog box

void critical_error(char * Message)
reports an error message in a dialog box and exits application

void add_to_listbox(HWND hDlg, int idc, char * str)
adds string str to a listbox that is identified by resource-number idc and dialog handle hDlg

4.3.9 Draw.cpp

This module holds function for creating global drawing objects like brushes, pens or fonts, and functions to draw the display window with elements and connections.

void init_draw(void)
creates some brushes, pens and fonts in the *DRAW*- structure

void draw_object(HDC hdc, WORD t)
draws element number *t* into the *hdc* device context. If this element is the active element, it will be drawn with an orange border.

void draw_connections(HDC hdc, WORD t)
draws the outging connectiong from element *t* into the device context *hdc*

void draw_captions(HDC hdc, WORD t)
draws the captions for element *t* into device context *hdc*

void draw_objects(HWND hWnd)
draws all elements, including connections and captions. hWnd is the hande to the design window

SDL_Surface *LoadBMP(char *filename)
Loads a bitmap file (.BMP) and rearranges the bytes for a SDL_surface

4.3.10 Midi.cpp

This module provides vaiables and functions for configuring the Midi-Out Device and generating midi-notes.

*** char midi_instnames[256]**
Ths array holds names for the 127 standard Midi-Instruments and 127 Midi-Controllers.

void init_midi (void)
copies the names of available midi-ouput devices on the system into the MIDIPORTS[..].portname structure and returns a counter of available ports in GLOBAL.midiports

void midi_Instrument(HMIDIOUT * midiout, int chn, int inst)
sets the current instrument for midi-device *midiout* and channel *chn* to *inst*

void midi_ControlChange(HMIDIOUT * midiout, int chn, int cont, int val)
outputs the control-change message *val* to controller *cont* and channel *chn* of midi-device *midiout*

void midi_NoteOn(HMIDIOUT * midiout, int chn, int note, int vol)
plays midi - tone *note* with volume *vol* on channe *chn* of midi-device *midiout*

void midi_NoteOff(HMIDIOUT * midiout, int chn, int note)
mutes the midi-tone *note* on channel *chn* of midi-device *midiout*

int midi_open_port(HMIDIOUT * midiout, int portnum)
opens the midi - output device number *portnum* and stores the hande in *HMIDIOUT*. Returns TRUE if successful, FALSE otherwise.

void mute_all_midi(void)
mutes all playing notes of all elements in the current design

4.4 Element Modules

The following table gives a summary of the modules containing signal processing elements. The row 'specific features' reports special funcionalities or library calls of the element, that could be useful for new element developments.

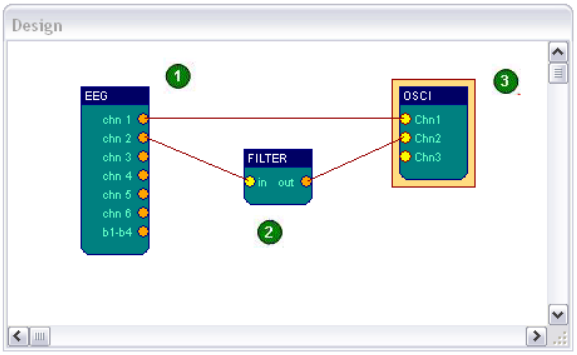
ELEMENT/MODULE	PURPOSE	SPECIFIC FEATURES
ob_and.cpp	outputs logical AND of two inputs	
ob_average.cpp	outputs averages ofprevious n samples	
ob_avi.cpp	extracts and displays a frame of an .AVI file, selected by the input value	OpenGl window
ob_ballgame.cpp	provides a 'catch-the-ball' - game, controlled by the input value	GDI drawing window
ob_cam.cpp	display and record live camera, perform face detection	livecam window, calls to OpenCV-library runs in own thread
ob_compare.cpp	compares two inputs, outputs result	
ob_com_writer.cpp	writes command sequence to COM-Port	writes to the COM-port of currently connected amplifier hardware
ob_constant.cpp	outputs constant value	
ob_correlation.cpp	outputs cross correlation of two inputs	
ob_counter.cpp	counts TRUE/FALSE transitions and / or displays value	GDI drawing window
ob_debounce.cpp	removes TRUE/FALSE transitions from input signal	
ob_deviation.cpp	calculates standard deviation	
ob_doku.cpp	provides textbox for documentation	
ob_edf_reader.cpp	reads edf file and provides signals at ouptu ports	dynamic number of output ports
ob_edf_writer.cpp	writes input values to edf file	dynamic number of input ports
ob_eeg.cpp	connects to eeg-amplifier, provides live data or archive data at ouputs ports	creates a reader thread for receiving data from the amplifier, paces signal procesing if connected
ob_erpdetect.cpp	records and averages input signal, ouputs difference of pattern and last n samples	displays recorded pattern in user dialog
ob_evaluator.cpp	evaluates mathematical expression containing input values	calls matheval-library
ob_fft.cpp	performs fourier transformation, display bar-graph, spectrogram or 3d-display	OpenGL display window
ob_file_writer.cpp	writes input values to text file	dynamic number of input ports
ob_filter.cpp	filters inputs signal with selectable filter type	display filter response in user dialog
ob_integrate.cpp	sums input signal	

ob_magnitude.cpp	outputs magnitude in selectabel passband	
ob_matlab.cpp	calls matlab engine with an array of n samples, provides ANS varibale	calls the matlab-engine libraries
ob_mci.cpp	interface to multimedia player, provides triggered playing, volume, speed and step - control	calls the vfw - API, displays own window for movies
ob_midi.cpp	generated midi-tones with selectable tonescale	calls the mmsystem midi API
ob_mixer4.cpp	mixes up to 4 signals to one output signal	
ob_mouse.cpp	provides mouse-cursor and clicking control	sets the mouse cursor and clicking events using mouseevent()
ob_not.cpp	outputs logical NOT of input value	
ob_or.cpp	outputs logical OR of two input values	
ob_osci.cpp	displays connected signals in oscilloscope view	GDI drawing window
ob_particle.cpp	displays particle system animation with selectable parameters	OpenGL drawing window
ob_sample_hold.cpp	stores an input value and outputs it continuously	
ob_signal.cpp	singnal generator with selectable type, center, amplitude, frequency	selectable input ports
ob_skindialog.cpp	skinned user dialog with buttons and scrollbar	creates skinned dialog window from external .ini - file
ob_tcp_receive.cpp	receives EDF-data from tcp-connection	connects to neuroserver network service, dynamic number of output ports
ob_tcp_sender.cpp	sends EDF-data to tcp-connection	connects to neuroserver network service, dynamic number of input ports
ob_threshold.cpp	performs static or dynamic thresholding on the input value	GDI drawing window
ob_translate.cpp	translate and/or gain input signal	
ob_wav.cpp	play a sound file with selectable speed	uses SDL_SOUND and modplug.dll to provide audio output

Note: the above table is not up-to-date, some of the newer elements are missing.

4.5 Signal Processing

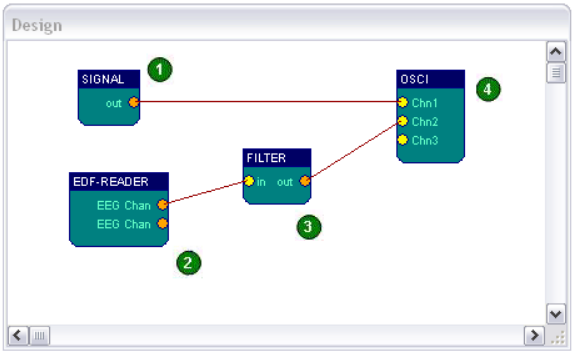
The internal processing of the signals in BrainBay is quite simple:
Each output port of an element can be connected to one or more input ports of other elements. Every input port can handle one connection. Floating point - values are passed from output to input ports at sampling frequency (for example 256 Hz), the *work()*-methods of all existing elements are called with that frequency. The calling-order is given by the element's connections: elements that generate data are called first, elements that receive data are called last.



The sorting is done when a new element is inserted into a design (see implementation in the *DesignWndHandler, dialogs.cpp*)

In the left figure, the EEG element will be processed first, the filter will be second and the oscilloscope will be last.

Processing values through the elements is accomplished by calling the elements' *work()* - methods. This task is initiated from either the timer interrupt (see implementation in *timer.cpp*) or from the reader-thread of the EEG-element that processes inputs from the COM-port. In the design shown in the above figure, the EEG-element will pace the signal processing. When a packet from the EEG-hardware has arrived, the current channel values are copied from the structure *PACKET.buffer[ ]* to the output ports of the EEG-element. Thus, all elements are provided with 'fresh' samples as soon as they arrive. (see implementation in *ob_eeg.cpp* and *tty.cpp*). The method *pass_values (int port, float value)* transfers ouput values to connected elements. *pass_values* will call the method *incoming_data (port, value)* of all connected elements.



This figure shows the element order of another design, where no EEG-element is present. The signal processing of this design will be paced by the timer interrupt.

5. Adding a new Element - an Example

If you want to build a new element, you will need the following:

- the element name and -number defined in *brainbay.h*
- a dialog containing the GUI for the element-properties
- a class derived from *BASE_CL*, with variables for the element-properties and implementations of the needed methods
  Any useful element will at least need the constructor and the *load*()- and *save()* mehtods to store element-specific data. Elements that process data will also need *incoming_data()* and *work()*.
- a dialog handler to control the GUI behaviour
- a menu entry and the call to *create_object()* in *brainbay.cpp* when the menu-point was selected (view the *MainWndHandler* in *brainbay.cpp*)

In the following, these steps will be shown for a new element called '**demo**'.
To access the files, open the project file *brainbay.dsw* with *MS-VisualStudio*.
For this demonstration, *VisualStudio C++ V6.0* was used.

5.1 Entries in brainbay.h

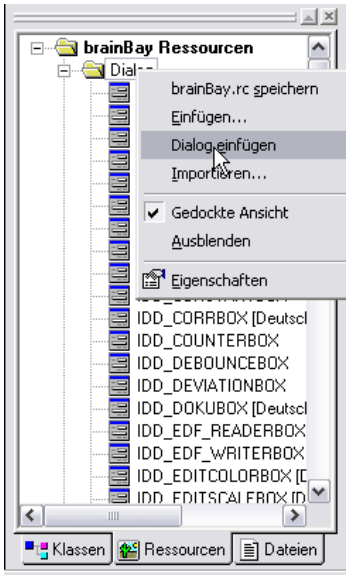
At the beginning of the file brainbay.h, you find the definitions of a unique constant per element. Add a line for the new object, OB_DEMO:

```
(....)
#define OB_FILE_WRITER  37
#define OB_DEVIATION    38
#define OB_MCIPLAYER    39
#define OB_DEMO         40
(...)
```

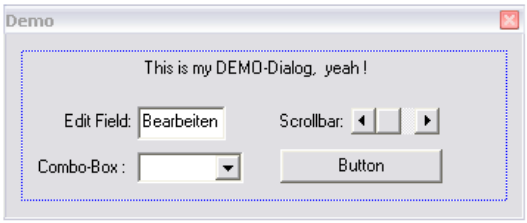
Beneath the constants, the default-tags for the elements in the design window are given in an array, add the name for the new element:

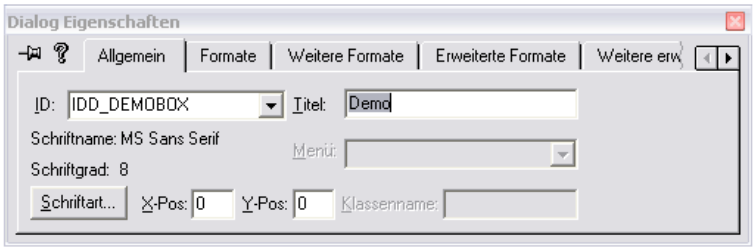
```
#define OBJNAMES "EEG","MIDI", (...),
    "FILEWRITE", "DEVIATION", "MEDIAPLAYER" , "DEMO"
```

5.2 Building a new dialog



Use the Resource Editor to add a new user dialog to the list of existing dialogs. This can be done by right-clicking the dialog-entry in the list of resources. When the new dialog was created, a new entry will emerge on the list, showing the default identifier for the dialog (*IDD_DIALOG1*). Rename the dialog identifier by double clicking the dialog window. This Dialog will be called *IDD_DEMOBOX*. This identifier will be needed later when the dialog window will be created.
Adjust the dialog settings according to the other elements (thin frame, absolute position, modal window, static border, stay in front). Delete the default '*OK*' and '*Cancel*' Buttons, name the dialog and add your interface itmes. For this example, a Text-Edit field, a Scrollbar, a Combobox a Button and some Static-text were inserted. All items have an identifier (*IDC_<idname>*). For simplicity, the identifiers were renamed to *IDC_EDIT, IDC_SCROLLBAR, IDC_COMBO* and *IDC_BUTTON* by double clicking the items.





5.3 Setting up a new class for the element

Create a new header (*.h*) and a new source (*.cpp*) - file to contain the element class and the dialog handler for the user dialog. The class is inherited from BASE_CL. The easiest way to generate source and header files is to copy an existing BrainBay element that has similar functionalities as the new element.

For this example, the files will be called *ob_demo.h* and *ob_demo.c* and the element will be called *DEMOOBJ*. Add the files to the *VC++* - project by using the menu *Project-> Add to Project -> Files*

5.3.1 Header file for the new element

ob_demo.h contains the declaration of the new class:

```
class DEMOOBJ : public BASE_CL
{
        protected:    // declare protected variables here

        public:       // public variables that can be accessed from Dialog Handler
                      int selection;              // selection from our combo-box
                      char string[50];            // text input into the text filed
                      int pos;                    // position of the scrollbar
                      float in1,in2;              // buffer for input values of port 1 and 2

// STANDARD METHODS (nearly every element has these) :

        DEMOOBJ(int num);                         // construction method
        void make_dialog(void);                   // to display the user dialog
        void load(HANDLE hFile);                  // to load element's properties
        void incoming_data(int port, float value);    // to process incoming data
        void save(HANDLE hFile);                  // to save elements properties
        void work(void);                          // for calculations and passing output values
        ~DEMOOBJ();                               // destruction of the element

// NON_STANDARD METHODS (some elements need these) :

        void update_inports (void);                //  when inports are connected or removed
        void session_start (void);                 //  called when session is started by user
        void session_stop (void);                  //  called when session is stopped by user
        void session_reset (void);                 //  called when session is reset by user
        void session_pos (long pos);               //  called when session pos was changes by user

  // Dialog handler to process user interaction with IDD_DEMOBOX :

        friend LRESULT CALLBACK DemoDlgHandler(HWND hDlg, UINT message, WPARAM
                                                    wParam, LPARAM lParam);
    private:        // declare private variables here
};
```

5.3.2 Source file for the new element

ob_demo.cpp contains the implementations of class and dialog handler. The function of the demo element will be to receive two inputs and compute one output in a way selectable by the combo box. The scrollbar-value will be used to scale the output from 0 - 100%. The Button will display a message on screen, containing the value of the text box. All standards methods will be implemented, and using the non-standard methods will be shown with dummy-actions.

We'll start with including the header files :

```
    #include "brainBay.h"                 // include the global definitions and structures
    #include "ob_demo.h"                  // include the element-specific declarations
```

Next comes the constructor for the object, which will be called from the main-menu (insert element) or when a design configuration is loaded :

```
DEMOOBJ::DEMOOBJ(int num) : BASE_CL()          // element constructor
{
    outports = 1;                              //  the element will have one output port
    inports = 2;                               //  the element will have two input ports
    strcpy(in_ports[0].in_name,"val1");        // input port1 will be call 'val1'
    strcpy(in_ports[1].in_name,"val2");        // input port2 will be call 'val2'
    strcpy(out_ports[0].out_name,"out");       // output port will be call 'out'
    strcpy (string, "hello world");            //  do additional initialisations here
}
```

If the element has features like an own drawing window or buffers to allocate, these initialisations should also take place in the constructor.

Next, we implement the *make_dialog()* method to display the user dialog :

```
void DEMOOBJ::make_dialog(void)        // will be called when element is right-clicked,
{
    display_toolbox(hDlg=CreateDialog(hInst, (LPCTSTR)IDD_DEMOBOX, NULL,
                                      (DLGPROC)DemoDlgHandler));
}
```

Now come the *load*- and *save* methods. To maintain a readable ASCII-format for the configuration (*.cfg*) file, the load_property and save_property - methods are used. Thus, configuration files can be imported with a text-editor.

```
void DEMOOBJ::load(HANDLE hFile)
{
  load_object_basics(this);                // load basic values like element position,
                                           // in- and outport properties, element tag
  //  now load our 3 element  properties :
  load_property("pos",P_INT,&pos);                      // load the scrollbar position
  load_property("selection",P_INT,&selection);          // load the combobox selection
  load_property("string",P_STRING,string);                        // load the edit text

 //  (...) do additional initialisations here
}

void DEMOOBJ::save(HANDLE hFile)     // hFile will be the opened configfile
{
   save_object_basics(hFile,this);           // save the basic values
   save_property(hFile,"pos",P_INT,&pos);
   save_property(hFile,"selection",P_INT,&selection);
   save_property(hFile,"string",P_STRING,string);

 //  the order of the properties is not critical,  properties could be loaded or saved at another
 //  position also. For float values, use the type P_FLOAT.
}
```

For objects that hold large amounts of data (like color palettes, tone scales, signal segments etc.) it would make no sense to store all this data in an alphanumerical format in the configuration file. Therefore, only a filename that points to a seperate file should be stored in such cases. De-initialisation of the previous setting and initialisation of a new setting may be necessary in the load() method. (like freeing a buffer, allocating new memory for a function, ...). Examples are the *filter*- and *magnitude*- elements.

Next, the *incoming_data* - method provides handling of data coming into the inputs ports of our element :

```
void DEMOOBJ::incoming_data(int port, float value)
{
  if (value != INVALID_VALUE)            // discard INVALID_VALUEs
  {
    if (port==0)   in1 = value;          // store value received at input port 1
    if (port==1)   in2 = value;          // store value received at input port 2
  }
}
```

The *work* method calculates the output according to combobox-selection and gain value of the scrollbar an passes it to elements connected to the output port:

```
void DEMOOBJ::work(void)
{
    switch (selection)          // generate output value according to combo-selection
    {
       case 0:   pass_values(0,  in1 * pos / 100.0f);  break;
       case 1:   pass_values(0,  in2 * pos / 100.0f );  break;
       case 2:   pass_values(0,  (in1+in2) * pos );  break;
    }
}
```

The class implementation is finished with the object deconstructor which cleans up allocated memory etc:

```
DEMOOBJ::~DEMOOBJ() {}    // nothing special in the deconstructor (nothing to free etc)
```

All needed methods are implemented by now. For demonstartion purpose, usage of the additional methods will be shown:

```
void DEMOOBJ::update_inports(void)      //  will be called when input ports are connected
{                                       //   or disconnected
      inports=count_inports(this);      //  get the number of the greatest connected port
                                        //  -> this will lead to dynamic number of input ports

      height=CON_START+inports*CON_HEIGHT+5;    // new size of the element
      InvalidateRect(ghWndDesign,NULL,TRUE);    //  repaint design window
}

void DEMOOBJ::session_start(void)               // will be called when Play -button
{   report ("Session has been started");  }     // in the status bar was pressed

void DEMOOBJ::session_stop(void)                // will be called when Stop- button
{   report ("Session has been stopped");  }     // in the status bar was pressed

void DEMOOBJ::session_reset(void)               // will be called when Reset- button
{   report ("Session has been reset"); }        // in the status bar was pressed

void DEMOOBJ::session_pos(long pos)             // will be called when Positioning the
{                                               // archive in the status bar was done
    char sztemp[50];
    sprintf(sztemp,"Session has been positioned to %d.",pos);
    report(sztemp);
}
```

Finally, we implement our dialog handler to process user interaction. The dialog box is created by the *make_dialog* method shown above.

```
LRESULT CALLBACK DemoDlgHandler(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
{
  DEMOOBJ * st;
  st = (DEMOOBJ *) actobject;              // first get a pointer to the active element:
                                           //   actobject, was selected by the user

  switch( message )
  {
   case WM_INITDIALOG:                     //  the user dialog is to be created
    SCROLLINFO lpsi;
    lpsi.cbSize=sizeof(SCROLLINFO);     // set the ranges for the scrollbar to 1 - 100
    lpsi.fMask=SIF_RANGE|SIF_POS;
    lpsi.nMin=1; lpsi.nMax=100;
    SetScrollInfo(GetDlgItem(hDlg,IDC_SCROLLBAR),SB_CTL,&lpsi, TRUE);

    //  set the scroll-position to the current value stored in pos
    SetScrollPos(GetDlgItem(hDlg,IDC_SCROLLBAR), SB_CTL,st->pos, TRUE);

    SetDlgItemText(hDlg, IDC_EDIT, st->string);        // display string in edit field

    //  now, fill the combobox with our three possible selections and select current value:
    SendDlgItemMessage( hDlg, IDC_COMBO, CB_ADDSTRING, 0,(LPARAM) "IN1") ;
    SendDlgItemMessage( hDlg, IDC_COMBO, CB_ADDSTRING, 0,(LPARAM) "IN2") ;
    SendDlgItemMessage( hDlg, IDC_COMBO, CB_ADDSTRING, 0,(LPARAM) "IN1+2") ;
    SendDlgItemMessage(hDlg,  IDC_COMBO, CB_SETCURSEL, st->selection, 0L ) ;
   break;

   case WM_CLOSE:                                  // the user dialog is to be closed
      EndDialog(hDlg, LOWORD(wParam));
      return TRUE;

   case WM_COMMAND:                                // a dialog Item receives a message
     switch (LOWORD(wParam))                       // get the ID of the dialog Item
     {
        case IDC_COMBO:                            // combo box  has been changed
         if (HIWORD(wParam)==CBN_SELCHANGE)
           st->selection=                          // store the new selection
           SendMessage(GetDlgItem(hDlg, IDC_COMBO), CB_GETCURSEL , 0, 0);
           break;

       case IDC_EDIT:                              // the edit field has been changed
           GetDlgItemText(hDlg, IDC_ÊDIT, st->string, 50);      // store the new text
           break;

       case IDC_BUTTON:                            // the button has been pressed
            report (st->string) ;                  // display message with current text
            break;
     }
     return TRUE;

   case WM_HSCROLL:                                // a scrollbar has been changed
   {
     int nNewPos= get_scrollpos(wParam,lParam);    // get the new value of the scrollbar
     // is this our scrollbar ? ( this is only necessary when there are more scrollbars)
     if (lParam == (long) GetDlgItem(hDlg,IDC_SCROLLBAR))
                 st->pos=nNewPos ;                 // yes: store value in pos
   }
   break;
```

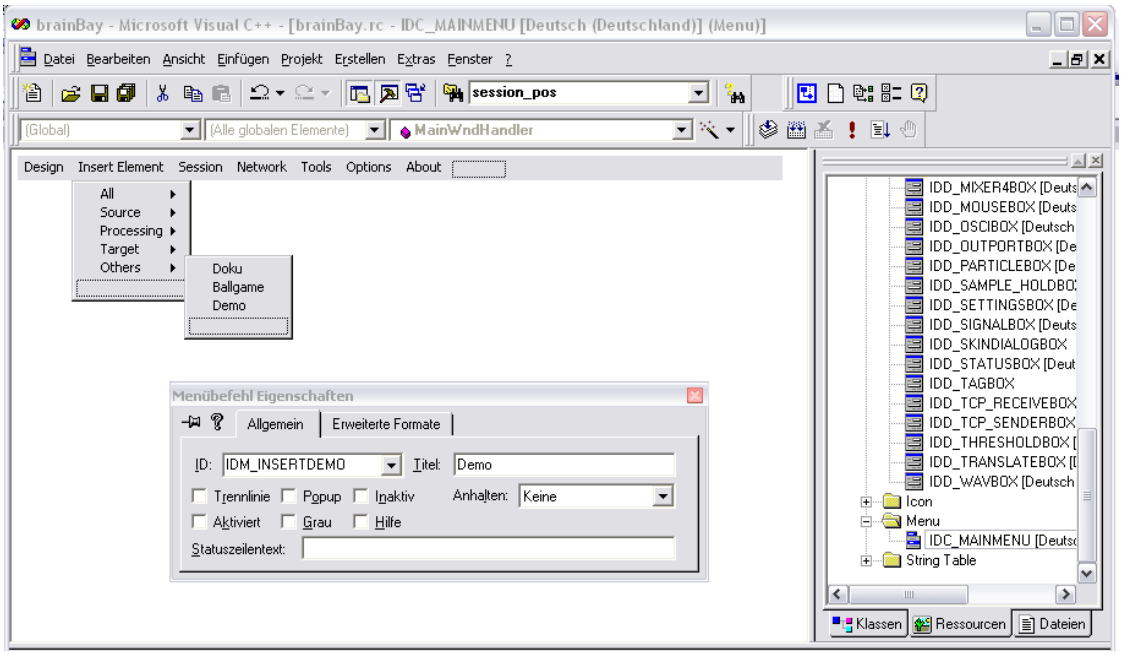
```
//  what remains is to take care of our user dialog:

     case WM_SIZE:                          // when the user dialog was moved or sized:
     case WM_MOVE:
          update_toolbox_position(hDlg);   // save the new position
          break;
   return(TRUE);
   }
return (FALSE);
}
```

That's it for the implementation :-)

5.4 The Menu-entry and Call for the new element

Use the resource-editor to create a menu entry for the demo-element:
Select the *Insert Element* - menu and the submenu the element belongs to (here *Others* was chosen). Add tiltle and identifier for the menupoint using the properties-winow :



Now, change to the source file *brainbay.cpp* which contains the main dialog handler. Browse to the location where the menu-points are processed and add a line for the new element :

```
( ... )
case IDM_INSERTFILE_WRITER:create_object(OB_FILE_WRITER);  break;
case IDM_INSERTDEVIATION:create_object(OB_DEVIATION);  break;
case IDM_INSERTMCIPLAYER:create_object(OB_MCIPLAYER);  break;
case IDM_INSERTDEMO:create_object(OB_DEMO);  break;
( ... )
```

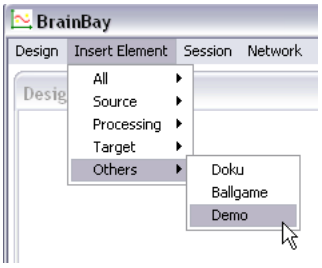
Change to the source file *globals.h*, and add the header file to the include-section:

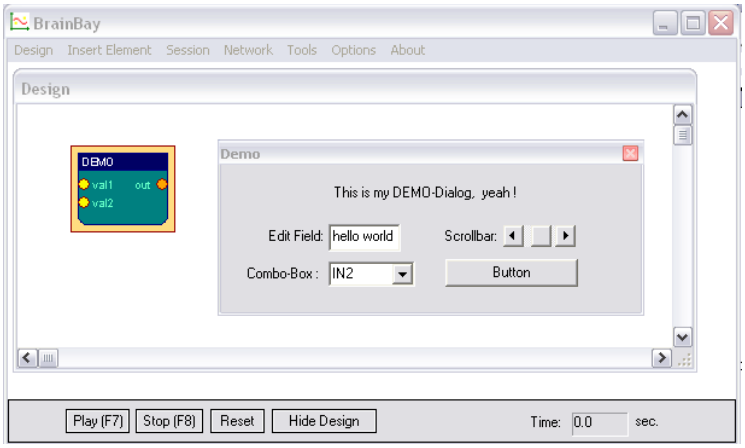
```
( ... )
#include "ob_file_writer.h"
#include "ob_deviation.h"
#include "ob_mci.h"
#include "ob_demo.h"
```

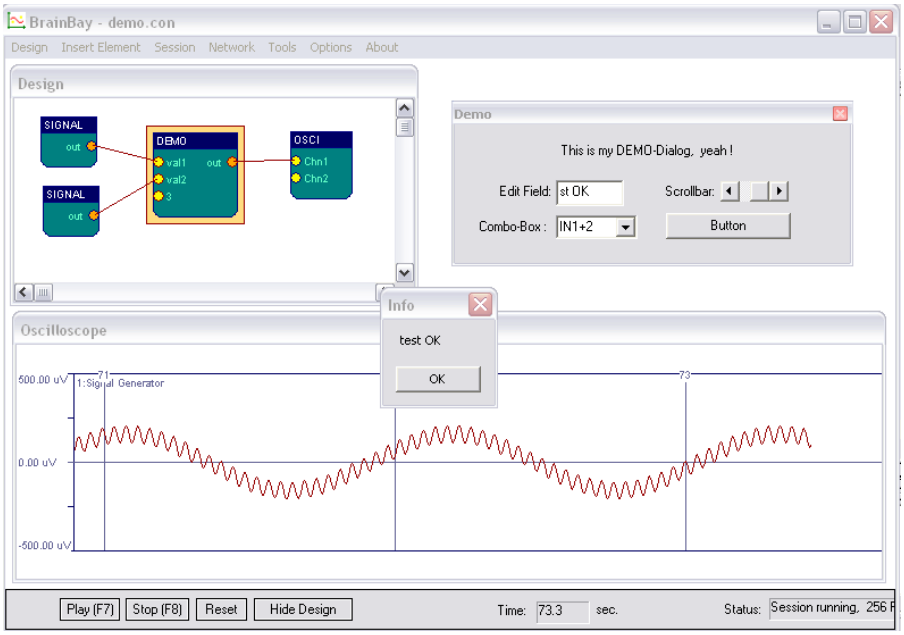
Browse to the *create_object* - function and add a line for the new element:

```
( ... )
case OB_FILE_WRITER: actobject=new FILE_WRITEROBJ(GLOBAL.objects); break;
case OB_DEVIATION:   actobject=new DEVIATIONOBJ(GLOBAL.objects); break;
case OB_MCIPLAYER:  actobject=new MCIOBJ(GLOBAL.objects); break;
case OB_DEMO:          actobject=new DEMOOBJ(GLOBAL.objects); break;
```

Now all is ready to compile the project and test the new element :







.. the demo-element in action ..

6. File Formats and Transmission Protocols

6.1 Structure of the European Data Format (EDF)

The EDF - File Standard is used by the EDF-File Reader, EDF-File Writer, TCP-Receive and TCP-Send - elementes. It allows a dynamic number of channels and signal ranges. The file header consists of 256 + <ns> * 256 bytes, where <ns> is the number of signals in the recording:

HEADER RECORD :

8 ascii : version of this data format (0)
80 ascii : local patient identification
80 ascii : local recording identification
8 ascii : startdate of recording (dd.mm.yy)
8 ascii : starttime of recording (hh.mm.ss)
8 ascii : number of bytes in header record
44 ascii : reserved
8 ascii : number of data records (-1 if unknown)
8 ascii : duration of a data record, in seconds
4 ascii : number of signals (ns) in data record

// ns.. number of signals (channels)
ns * 16 ascii : labels (e.g. EEG FpzCz or Body temp)
ns * 80 ascii : transducer types (e.g. AgAgCl electrode)
ns * 8 ascii : physical dimensions (e.g. uV or degreeC)
ns * 8 ascii : physical minimums (e.g. -500 or 34)
ns * 8 ascii : physical maximums (e.g. 500 or 40)
ns * 8 ascii : digital minimums (e.g. -2048)
ns * 8 ascii : digital maximums (e.g. 2047)
ns * 80 ascii : prefiltering infos (e.g. HP:0.1Hz LP:75Hz)
ns * 8 ascii : nr of samples / segment
ns * 32 ascii : reserved

The header is followed by channel data in binary format:

channel[1] * integer : first signal in the data record
channel[2] * integer : second signal
..
..
channel[ns] * integer : last signal

6.2 P2 - Firmware Protocol

The P2 Firmware Protocol was the inital transmission protocol of the OpenEEG project, used by ModularEEG. It is compatible with the ElectricGuru application. P2 uses 17 data bytes to transmit 6 channels of EEG data:

```
1:  sync0;          // synchronisation byte 1 = 0xa5
2:  sync1;          // synchronisation byte 2 = 0x5a
3:  version;        // version number = 2
4:  count;          // packet counter. Increases by 1 each packet.
5:  Chn1high        // channel 1 high byte
6:  Chn1low         // channel 2 low byte
7:  Chn2high        // channel 2 high byte
8:  Chn2low         //  ...
9:  Chn3high
10: Chn3low
11: Chn4high
12: Chn4low
13: Chn5high
14: Chn5low
15: Chn6high
16: Chn6low         // channel 6 low byte
17: switches;       // State of PD5 to PD2, in bits 3 to 0.
```

6.3 P3 - Firmware Protocol

ModularEEG Packet Format Version 3 is the successor of the P2 protocol. It provides a packet transmission that could achive higher sampling rates with the same baudrate. One packet can have zero, two, four or six channels (or more). The default is a 6-channel packet, shown below.

```
0ppppppx    packet header
0xxxxxxx
0aaaaaaa    channel 0 LSB
0bbbbbbb    channel 1 LSB
0aaa-bbb    channel 0 and 1 MSB
0ccccccc    channel 2 LSB
0ddddddd    channel 3 LSB
0ccc-ddd    channel 2 and 3 MSB
0eeeeeee    channel 4 LSB
0fffffff    channel 5 LSB
1eee-fff    channel 4 and 5 MSB
```

Key:

1 and 0 = sync bits. Note that the '1' sync bit is in the last byte of the packet, regardless of how many channels are in the packet.
p = 6-bit packet counter
x = auxillary channel byte
a - f = 10-bit samples from ADC channels 0 - 5
- = unused, must be zero

There are 8 auxillary channels that are transmitted in sequence. The 3 least significant bits of the packet counter determine what channel is transmitted in the current packet.

Aux Channel Allocations:
0: Zero-terminated ID-string (ASCII, currently "mEEGv1.0")
4: Port D status bits (Aux channels 1-3 and 4-7 are curently free)

6.4 P21 - Firmware Protocol

In P21 protocol, a packet has variable length, it depends on how many channels we want to receive. Each channel can be set individually via backward communication. This bi-direction option is used by the MonolithEEG amplifier to process also changes of the sampling- and baudrate as setting the digital output lines of the microcontroller.

Packet format

```
1cccnnnn
0pppaaaa
0bbbbbbb
....
0pppaaaa
0bbbbbbb

ccc  - number of channels in packet, total packet length = 1 + (2 * ccc);
nnnn - 4 bit control sequence number
ppp  - channel id, 0-5 are A/D channels,
6 is for backward information (requested or error code)
7 - not used now
aaaa - if the ppp is 0-5, then the lower three bits of aaaa contains 3 highest bits of 10bit
channel value, the highest bit of aaaa is set to 0
      if the ppp is 6, then it contains id/selector of backward information
bbbbbbb - if the ppp is 0-5, then it contains 7 lower bits of 10bit value
        if the ppp is 6, then it contains backward information value
```

7. Credits and Links

Joerg Hansmann initiated the OpenEEG project with his marvellous hardware-design of the ModularEEG. Find further information about the amplifier design at http://openeeg.sourceforge.net/doc/modeeg/modeeg.html

Reiner Muench developed the MonolithEEG, which improved some features of the ModularEEG and provides a lightweight and small SMD-design. Find his great work at: http://people.freenet.de/moosec/projekte/simpleeeg/index-Dateien/MonolithEEG13_e.htm

Contributions to the BrainBay Application were made by Jim Peters by providing the filter library. Find his work at http://uazu.net.

Jeremy Wilkerson and Lester John did a great job in developing some useful BrainBay elements.

For other project related information visit:
http://openeeg.sf.net
http://brainbay.lo-res.org/links.htm

8. Disclaimer - Nonliability

8.1 Content

The author reserves the right not to be responsible for the topicality, correctness, completeness or quality of the information provided. Liability claims regarding damage caused by the use of any information provided, including any kind of information which is incomplete or incorrect, will therefore be rejected. All offers are not-binding and without obligation. Parts of the pages or the complete publication including all offers and information might be extended, changed or partly or completely deleted by the author without separate announcement.

8.2 Referrals and links

The author is not responsible for any contents linked or referred to from his pages - unless he has full knowledge of illegal contents and would be able to prevent the visitors of his site from viewing those pages. If any damage occurs by the use of information presented there, only the author of the respective pages might be liable, not the one who has linked to these pages. Furthermore the author is not liable for any postings or messages published by users of discussion boards, guestbooks, blogs or mailinglists provided on his page.

8.3 Copyright

The Source Code of the Application is provided in terms of the GNU GPL licence, see chapter 9. The author intended not to use any copyrighted material for the publication or, if not possible, to indicatethe copyright of the respective object. The copyright for any material created by the author is reserved. Any duplication or use of objects such as diagrams, sounds or texts in other electronic or printed publications is not permitted without the author's agreement.

8.4 Privacy policy

If the opportunity for the input of personal or business data (email addresses, name, addresses) is given, the input of these data takes place voluntarily. The use of all offered services are permitted - if and so far technically possible and reasonable - without specification of any personal data or under specification of anonymized data or an alias. The use of published postal addresses, telefone or fax numbers and email addresses for marketing purposes is prohibited, offenders sending unwanted spam messages shall be punished.

8.5 Legal validity of this disclaimer

This disclaimer is to be regarded as part of the internet publication which you were referred

9. GNU GENERAL PUBLIC LICENSE

Version 2, June 1991
Terms and Conditions for copying, distribution and modification

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation inthe term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

9.1 You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

9.2 You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 9.1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program it self is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your

rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

9.3 You may copy and distribute the Program (or a work based on it, under Section 9.2 in object code or executable form under the terms of Sections 9.1 and 9.2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 9.1 and 9.2 above on a medium customarily used for software interchange; or,
b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 9.1 and 9.2 above on a medium customarily used for software interchange; or,
c) Accompany it with the information you received as to the offer to distribute corre sponding source code. (This alternative isallowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

9.4 You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9.5 You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

9.6 Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

9.7 If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9.8 If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9.9 The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

9.10 If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

9.11 BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

9.12 IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9.13 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can re-distribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found:

COPYRIGHT
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more de-tails.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. Also add information on how to contact you by electronic and paper mail.

This General Public License does not permit incorporating your program into proprie-tary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

If sections or individual terms of this statement are not legal or correct, the content or validity of the other parts remain uninfluenced by this fact.