EcgSoft®

# Software Developer's Guide to EcgViewer

*Innovative ECG Software*
www.ecg-soft.com - e-mail: info@ecg-soft.com

This page is intentionally left blank

## Contents

# 1. Introduction

EcgViewer application provides you with a graphical interface to display, review, edit and print ECG files in SCP-ECG format and also in other standard or non-standard proprietary formats.

"Open Interface Access" is the key to the universal usage of the EcgViewer with various standard and non-standard ECG data formats.

Thanks to the "Open Interface Access" of the EcgViewer, proprietary ECG data can be viewed if a suitable plug in DLL (Dynamically Linked Library) is available or created conforming to "Open Interface Access".

# 2. About the Guide

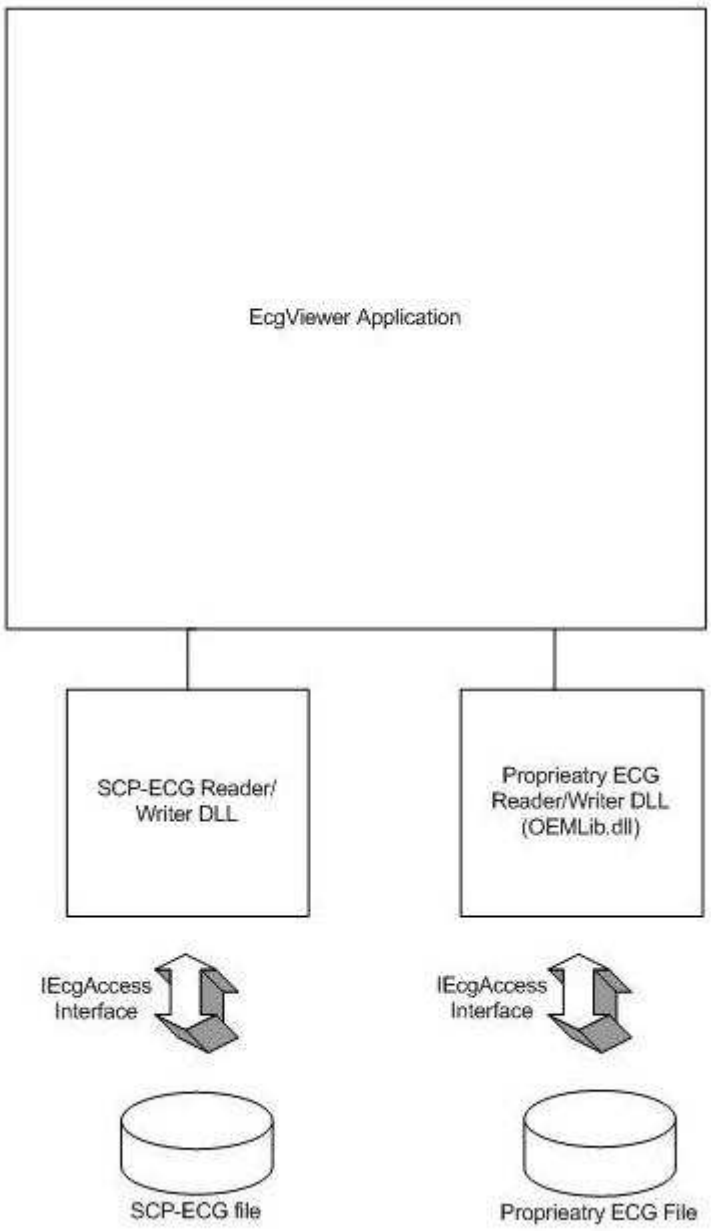
This guide will help you create a suitable DLL software for your proprietary ECG data with minimal effort and time on your side. Depending on the complexity of the proprietary ECG data format, software development time of a few hours to utmost a few days is sufficient for a programmer familiar with his proprietary ECG data format.

At present, EcgViewer supports SCP-ECG, Binary, MIT-BIH Arrhythmia Data Base formats as well as other proprietary formats of various ECG Manufacturers.

## 3. EcgViewer Open Interface Block Diagram

As illustrated in the diagram below, all data transfer to and from EcgViewer is realized over the IEcgAccess Interface.

For each ECG data format to be supported, one Reader/Writer DLL of the name "OEMLib.dll" needs to be created. The following example demonstrates the implementation of IEcgAccess interface to create an "OEMLib.dll" DLL for reading and viewing a binary ECG file.



## 4. Creating an "OEMLib.dll" for binary ECG data format

Let us assume that we have a proprietary binary ECG file where pure raw ECG data is stored in 2 bytes signed integer format. Also assume that data storage format is declared in C# language syntax as follows:

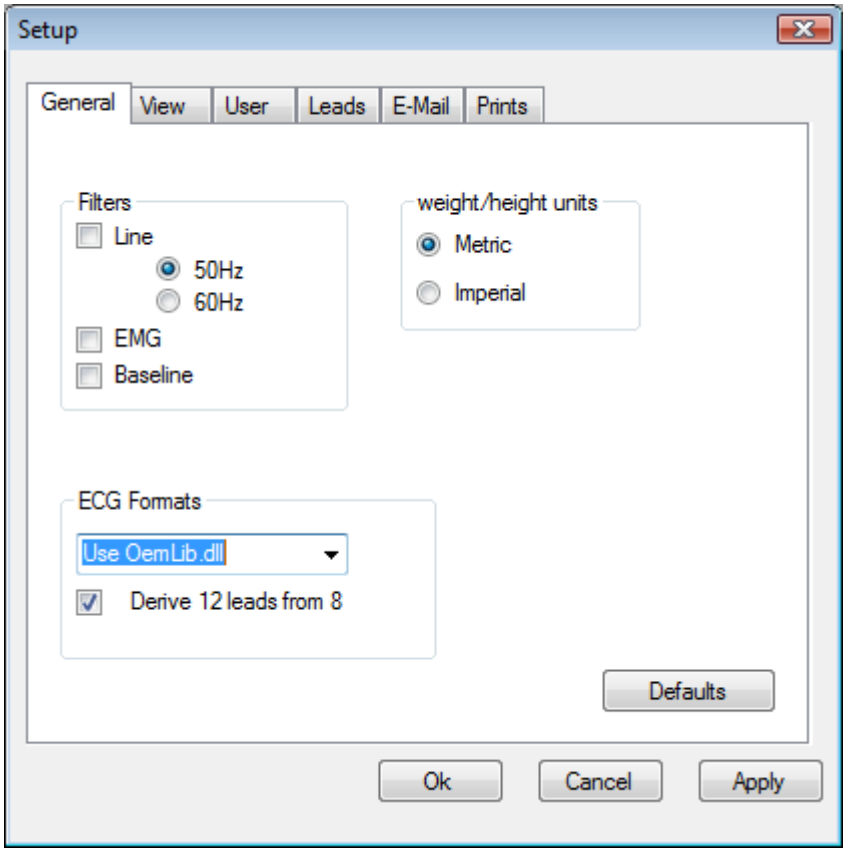
```
// first sample of  12 leads
short int     I;          //  2 bytes signed integer
short int     II;         //  2 bytes signed integer
short int     III;        //  2 bytes signed integer
short int     aVR;        //  2 bytes signed integer
short int     aVL;        //  2 bytes signed integer
short int     aVF;        //  2 bytes signed integer
short int     V1;         //  2 bytes signed integer
short int     V2;         //  2 bytes signed integer
short int     V3;         //  2 bytes signed integer
short int     V4;         //  2 bytes signed integer
short int     V5;         //  2 bytes signed integer
short int     V6;         //  2 bytes signed integer

// Second sample of 12 leads
short int     I;          //  2 bytes signed integer
short int     II;         //  2 bytes signed integer
short int     III;        //  2 bytes signed integer
short int     aVR;        //  2 bytes signed integer
short int     aVL;        //  2 bytes signed integer
short int     aVF;        //  2 bytes signed integer
short int     V1;         //  2 bytes signed integer
short int     V2;         //  2 bytes signed integer
short int     V3;         //  2 bytes signed integer
short int     V4;         //  2 bytes signed integer
short int     V5;         //  2 bytes signed integer
short int     V6;         //  2 bytes signed integer

// Nth sample 12 leads
.
.
.
```

This format is straightforward where each lead data for the particular sample is written consecutively in 2 bytes signed integer format. And then next sample is stored similarly.

Note that there is no patient info, diagnostic info or ECG signal info (i.e. sampling rate, number of leads, etc) stored in the binary file.

As illustrated in the following C# code pages , you need to implement these steps:

1. Create a new class library solution and name the solution as OemLib.
2. Let the OemEcg class implement the IEcgAccess interface as: `public class OemEcg :IEcgAccess`
3. Implement the methods of IEcgAccess as related to your ECG data format
4. Compile the solution and copy the "OemLib.dll" into the application folder of EcgViewer at "..\ProgramFiles\EcgSoft\EcgViewer"
5. Finally, select from Setupmenu, General->ECG Formats->Use OemLib.dll option so that EcgViewer would use your implementation as default.




You may also download and modify this solution from www.ecg-soft.com web site. There is also another solution available at the same site for reading MIT-BIH Database Arrhythmia recordings.

IEcgAccess interface comprises Patient Info, Diagnostic Info, ECG Signal Info and ECG data. Of particular importance are the ECG `SignalInfo` structure and its members:

1. `public string Leads`

   Define your lead names here. Do not put extra spaces or characters in between. Example: Leads = "I,II,III,aVR,aVL,aVF,V1,V2,V3,V4,V5,V6";

2. `public double SRATE`

   Sampling rate in samples per second

3. `public double ADU`

   Number (Analog/Digital ConversionValue) corresponding to 1 mv ECG signal

```
/// <summary>
/// OemEcg Class example for read/write of 12 leads ECG data stored in a
/// binary extension file (*.bin). IEcgAccess interface is implemented.
/// Use this example to implement your own data format class as OemEcg.
/// Then create OemLib.dll and copy it to the application folder of viewer.
/// Next from Setupmenu, select General->ECG Formats->Use OemLib.dll option
/// so that viewer would use your implementation as default.
/// </summary>
/// <remarks>
/// All 12 leads are assumed to be  simultaneously recorded. Each sample is
/// 2 byte long samples are saved as:
/// [Sample1:(I,I,II,aVR,aVL,aVF,V1,V2,V3,V4,V5,V6), Sample2:(..),
///  Sample3:(..),....]
/// Only pure ECG data is saved in this example
/// </remarks>
///

public class OemEcg : IEcgAccess
{
    // string for file extension filtering in Open file dialog box
    // of the EcgViewer

    public static string fileFilter = "Binary-Ecg (*.bin)|*.bin";

    Person myPatient;
    SignalInfo myEcgInfo;
    ArrayList ecgDataList;
    Diagnostic myDiagnoseInfo;

    // Default constructor
    public OemEcg()
    {

        myPatient = new Person();     //create a new patient info structure
        myEcgInfo = new SignalInfo(); //create a new ecg info structure
        myDiagnoseInfo = new Diagnostic();
        ecgDataList = new ArrayList();

        // fill patient data with some dummy patient info. (i.e no patient
        // info is stored in the binary file)
```

```
        myPatient.ID = "Demo Patient";
        myPatient.FirstName = "John";
        myPatient.LastName = "Begg";
        myPatient.Height = 180;
        myPatient.Weight = 75;
        myPatient.WeightUnit = 0;
        myPatient.Sex = 1;

        // some dummy remarks
        myDiagnoseInfo.FreeText  =  " This is a demo record to represent a
        .bin type ecg record where  2 bytes x 12 channel Ecg data is stored
        without any patient info";

        // fill Ecg info
        myEcgInfo.Acq.Year = 1962;
        myEcgInfo.Acq.Month = 16;
        myEcgInfo.Acq.Day = 11;
        myEcgInfo.Acq.Hour = 13;
        myEcgInfo.Acq.Minute = 25;
        myEcgInfo.Acq.Second = 46;

        myEcgInfo.ADU = 5464.48;  //number representing 1 mv of ECG signal
        myEcgInfo.SRATE = 1000;   //sampling rate in samples per second
        myEcgInfo.DeviceSN = "Demo Device";
        myEcgInfo.Leads = "I,II,III,aVR,aVL,aVF,V1,V2,V3,V4,V5,V6";
    }


    /*----------- Implement IEcgAccess interface methods ----------------*/

    // gets or sets Patient Information as specified by IEcgAccess
    // interface and Person Structure
    public Person PatientInfo
    {
        get { return myPatient; }
        set { myPatient = value; }
    }

    // gets or sets pyhsician remarks as specified by IEcgAccess interface
    public Diagnostic DiagnoseInfo
    {
        get { return myDiagnoseInfo; }
        set { myDiagnoseInfo = value; }
    }

    // gets or sets Ecg related data as specified by IEcgAccess interface
    // and SignalInfo structure
    public SignalInfo EcgInfo
    {
        get { return myEcgInfo; }
        set { myEcgInfo = value; }
    }

    // gets or sets Ecg data as specified by IEcgAccess interface
    public ArrayList EcgData
    {
        get { return ecgDataList; }
        set { ecgDataList = value; }
    }
```

```
/// <summary>
/// read the entire ECG data of the given fileName
/// </summary>
/// <param name="fileName"> name of Binary (*.bin) file </param>
/// <returns>true if successful </returns>
///
public bool Read(string fileName)
{
    // Open a file stream and a Binary Reader associated with this
    // stream
    FileStream myStream = new FileStream(fileName, FileMode.Open,
    FileAccess.Read);
    BinaryReader myReader = new BinaryReader(myStream);

    // First clear the collection
    ecgDataList.Clear();
    short[] ecgArray;

    myStream.Position = 0;

    // each block is 12 channels by 2 bytes)
    long length = myStream.Length / (12 * 2);

    // Now read the ECG data
    for (int j = 0; j < length; j++)
    {
        ecgDataList.Add(new short[12]);
        ecgArray = (short[])ecgDataList[j];

        ecgArray[0] = myReader.ReadInt16();
        ecgArray[1] = myReader.ReadInt16();
        ecgArray[2] = myReader.ReadInt16();
        ecgArray[3] = myReader.ReadInt16();
        ecgArray[4] = myReader.ReadInt16();
        ecgArray[5] = myReader.ReadInt16();
        ecgArray[6] = myReader.ReadInt16();
        ecgArray[7] = myReader.ReadInt16();
        ecgArray[8] = myReader.ReadInt16();
        ecgArray[9] = myReader.ReadInt16();
        ecgArray[10] = myReader.ReadInt16();
        ecgArray[11] = myReader.ReadInt16();
    }

    // Close the file stream and associated binary reader
    myReader.Close();
    myStream.Close();
    return true;
}


/// <summary> write the entire Ecg data into the binary file with
              specified file name  </summary>
/// <param name="fileName"> name of Binary (*.bin) file </param>
/// <returns>true if successful </returns>
///
public bool Write(string fileName)
{
    // Open a file stream and a Binary Reader associated with this
    // stream
```

```
            FileStream myStream = new FileStream(fileName, FileMode.Create);
            BinaryWriter myWriter = new BinaryWriter(myStream);

            // Write the 12 channel raw ECG data
            short[] ecgArray;
            for (int i = 0; i < ecgDataList.Count; i++)
            {
                ecgArray = (short[])ecgDataList[i];
                for (int j = 0; j < ecgArray.Length; j++)
                    myWriter.Write(ecgArray[j]);
            }

            // Close the file stream and associated binary writer
            myWriter.Close();
            myStream.Close();
            return true;
        }
    }
```

## 5. IEcgAccess Open Interface Version 1.2

Implementing the open interface enables a proprietary or standard based ECG data to be viewed by the EcgViewer. Below is the C# code for the open interface IEcgAccess:

```
/// <summary>
/// IEcgAccess Interface to implement read/write API
/// </summary>
public interface IEcgAccess
{
    /// <summary>
    /// Reads entire record at the given path and returns true if
    /// successfull
    /// </summary>
    bool Read(string fileName);

    /// <summary>
    /// Writes entire record to the given path and returns true if
    /// successfull
    /// </summary>
    bool Write(string fileName);

    /// <summary>
    /// Patient related info is grouped here
    /// </summary>
    Person PatientInfo
    {
        get;
        set;
    }

    /// <summary>
    /// ECG related info, excluding ecg data, is grouped here
    /// </summary>
    SignalInfo EcgInfo
    {
        get;
        set;
    }


    /// <summary>
    /// Diagnose related info is grouped here
    /// </summary>
    Diagnostic DiagnoseInfo
    {
        get;
        set;
    }


    /// <summary> gets or sets ECG data as specified by IEcgAccess
    /// interface
    /// </summary>
    ArrayList EcgData
    {
        get;
        set;
    }
```

```
}
/// <summary>
/// Simple DateTime structure
/// </summary>
public struct DateAndTime
{
    public int Year;
    public int Month;
    public int Day;
    public int Hour;
    public int Minute;
    public int Second;
}

/// <summary>
/// Personal Details of the patient.
/// </summary>
public struct Person
{
    /// <summary>Patient Identification Number, reasonable length
    /// 40 characters
    /// </summary>
    public string ID;

    /// <summary>First Name </summary>
    public string FirstName;

    /// <summary>Second Last Name </summary>
    public string SecondLastName;

    /// <summary>Last Name </summary>
    public string LastName;

    /// <summary>
    /// 0.Not known 1.Male, 2.Female, 9.Unspecified
    /// </summary>
    public int Sex;



    /// <summary>
    /// 0.Unspecified, 1.Caucasian, 2.Black, 3.Oriental
    /// </summary>
    public byte Race;

    /// <summary> Weight in specified units </summary>
    public int Weight;

    /// <summary>
    /// 0.Unspecified, 1.Kilogram, 2.Gram, 3.Pound, 4.Ounce
    /// </summary>
    public byte WeightUnit;
    /// <summary> Height in specified units </summary>
    public int Height;

    /// <summary>
    /// 0.Unspecified, 1.Centimeters, 2.Inches, 3.Millimeters
    /// </summary>
    public byte HeightUnit;
```

```
    /// <summary>
    /// Date of birth specified in DateAndTime structure
    /// </summary>
    public DateAndTime Birth;

    /// <summary> Age in years </summary>
    public int Age;
}

/// <summary>
/// Details related to ECG signal characteristics and data
/// acquisition
/// </summary>

public struct SignalInfo
{
    /// <summary>
    /// Enter Lead names EXACTLY as decribed in Lead
    /// Identification Codes table.
    /// Do not put extra spaces or characters in between.
    /// Example: "I,II,III,aVR,aVL,aVF,V1,V2,V3,V4,V5,V6"
    /// </summary>
    public string Leads;

    /// <summary>
    /// If all lead recording lengths are not same, create and
    /// set this array to specify lead lengths in samples.
    /// If same, no need to use this field.
    /// </summary>
    public uint[] LeadLengths;

    /// <summary>
    /// A/D value corresponding to 1 mv at the input
    /// </summary>
    public double ADU;

    /// <summary> Sampling rate in samples per second </summary>
    public double SRATE;



    /// <summary> Date and Time of Acquisition in DateAndTime
    /// structure
    /// </summary>
    public DateAndTime Acq;

    /// <summary>
    /// "cut-off" frequency (-3 db) of the high pass baseline
    /// filter in Hz
    /// </summary>
    public double BaseLineFilter

    /// <summary>
    /// "cut-off" frequency (-3db) of the low pass filter in Hz
    /// </summary>
    public double LowPassFilter;

    /// <summary>
```

```
        /// if other filters, which were not defined above, indicated
        /// here.
        /// Bit 0:60 Hertz notch filter, Bit 1:50 Hertz notch filter
        /// Bit 2:Artifact filter Bit 3:Baseline filter (e.g.
        /// adaptive filter or spline filter)
        /// </summary>
        public byte FilterBitMap;

        /// <summary>
        /// Text model description. Up to 5 bytes of text
        /// </summary>
        public string TextModel;

        /// <summary>
        /// Language Support Code (one byte). This bit map indicates
        /// the supported character sets.
        /// Refer to SCP-ECG standart for the explanation of codes
        /// </summary>
        public byte LanguageCode;

        /// <summary>
        /// Capabilities of the ECG Device (one byte bit map).
        /// Bit 0-3 reserved, 4 printing, 5 analysis, 6 storage, 7
        /// acquisition
        /// </summary>
        public byte CapabilitiesBitMap;

        /// <summary>
        /// Specifies frequency of the AC Mains
        /// 0: unspecified, 1: 50Hz, 2: 60Hz
        /// </summary>
        public byte ACMainsFrequency;

        /// <summary> Serial Number of Acquisition Device </summary>
        public string DeviceSN;

        /// <summary> Acquisition device SCP implementation software
        /// identifier (maximum 24 characters) </summary>
        public string DeviceSCP;




        /// <summary> Manufacturer of Acquisition Device </summary>
        public string DeviceMF;

    }

    /// <summary>
    /// Details related to Diagnosis and Medical history
    /// </summary>

    public struct Diagnostic
    {
        /// <summary> Systolic blood pressure in mmHg </summary>
        public ushort Systolic;

        /// <summary> Diastolic blood pressure in mmHg </summary>
        public ushort Diastolic;

        /// <summary> Acquiring Institution Description  </summary>
```

```
    public string AcquiringInstitute;

    /// <summary> Referring Physician </summary>
    public string ReferringPhysician;

    /// <summary> Diagnosis or the referral indication</summary>
    public string Diagnosis;

    /// <summary> Medical History</summary>
    public string MedicalHistory;

    /// <summary> Remarks and free text</summary>
    public string FreeText;
}
```

## 6. How to call EcgViewer (Embeddable Version) from your application

In some applications, it may be desirable to automatically launch the EcgViewer and view a specific ECG file without user interruption.

For example, you may have an ECG database application where you may need to view an ECG file by running the EcgViewer within your application.

Embeddable version of EcgViewer lets you input an ECG file name as a command line parameter and run it in the syntax: " EcgViewer.exe FileName "

Following C# example illustrates this by using System.Diagnostics name space of .Net Framework:

```
using System.Diagnostics;
.
.

// ECG file to be viewed (just an example)
string fullFilePath = "C:\\myECGDataFolder\\mySampleEcg.scp";

// Keep the string intact including spaces, otherwise each space results in
// creation of a new argument
fullFilePath = string.Format("\"{0}\"", fullFilePath);

// path of the ProgramFiles folder
string pathProgramFiles = Environment.GetFolderPath(
                          Environment.SpecialFolder.ProgramFiles);
// path of the EcgViewer.exe application
string exePath = pathProgramFiles + "\\EcgSoft\\EcgViewer";

Process p = new Process();
p.StartInfo.UseShellExecute = false;
p.StartInfo.FileName = exePath + "\\EcgViewer.exe";
p.StartInfo.CreateNoWindow = true;        // do not display ms-dos window
p.StartInfo.Arguments = fullFilePath      // ECG file to be viewed
p.EnableRaisingEvents = false;

// warn user if EcgViewer editor does not exist or can not be run
try
{
      p.Start();
      p.WaitForExit();        // wait until EcgViewer is closed
}
catch
{
      MessageBox.Show("Can not run EcgViewer !");
}
```