A short introduction to GUI programming with the MVC concept using wxWindows

M. Bernreuther

Summer term 2000

Keywords wxWindows, C++ , GUI programming, MVC concept

Abstract

The article starts with an introduction to the C++ library wxWindows. The first small example wxHello, is the well known "Hello world" type example. Another example wxKoch will be extended to show the implementation of a Model-View-Controller concept.

Contents

1 Introduction to wxWindows

1.1 What is wxWindows

wxWindows is a C++ library which allows software development for several platforms. It's main target is the programming of the Graphical User Interface (GUI). To get more information look at the homepage of wxWindows `http://www.wxwindows.org/`. The examples are written for the windows version using Microsoft's Visual C++ compiler. It should also compile on all other supported platforms, like Unix (GTK+ or Motif) or Apple's Macintosh. For more information on the GTK+ based wxWindows version (called wxGTK), which is often used with GNU/Linux, look at `http://www.freiburg.linux.de/~wxxt/`.

Currently version 2.2.0 is available. For downloads look at `http://www.wxwindows.org/dl_msw2.htm`, the main ftp site `ftp://www.remstar.com/pub/wxwin/` or a mirror site. We will use the setup executable `ftp://ftp.mapsy.nat.tu-bs.de/pub/mirror/wxwin/2.2.0/wxMSW-2.2.0-setup.zip`. You may also get `wxWindows-2.2.0-WinHelp.zip` and add the files after the installation procedure.

There is also a quite helpful mailing list. Look at `http://www.wxwindows.org/maillst2.htm` for information.

1.2 Installation

After unzipping and doing the common installation procedure using setup.exe, the library has to be build. There are two ways:

1. using makefiles
2. using VC project and workspace files (`%wxwin%\src\wxvc.ds?`)

Help can generally be found at `%wxwin%\docs`, where `%wxwin%` is the wxWindows root directory. `%wxwin%\docs\msw\install.txt` contains information about the installation procedure.

Do "rebuild all" for the "debug" and "release" configuration of "wxvc". Additional to `wxvc.ds?`, which generates static libraries, there's `wxvc_dll.ds?` for a dynamic link library (dll) version.

To find out about the features of wxWindows, the samples are very helpful. There are found at `%wxwin%\samples`. To build them all, `SamplesVC.dsw` can be used.

2 The first programm

The first example is the famous "hello world" application. It can be found at `http://www.wxwindows.org/hworld2.txt`. A similar program is explained at `http://www.wxwindows.org/hello.htm`. You might also have a look at the `minimal` sample.

It's important that after creating a new project with VC++, the project settings are adapted to needs of wxWindows, e. g. the search path have to include the wxWindows include and library files. One possibility is to (mis)use an existing sample. To simplify the process a tool is developed here called wxProjectGenerator to generate the *.ds? files, which can be opened with VC++. Let's generate a project called wxHello. There is a resource file called "wxHello.rc" with only one line

```
#include "wx/msw/wx.rc"
```

which will be generated automatically.

The source code is found in "wxHello.cpp". In this first example "wxHello.h" does not exist and all declarations will be made in the source file. Let's go through it:

```
// wxHello.cpp
// Robert Roebling, Martin Bernreuther

#include <wx/wx.h>
```

First the wxWindows library has to be included. There is a class called wxApp defined there, which represents the application.

```
class wxHelloApp : public wxApp
{
    virtual bool OnInit();
};
```

Now the wxHelloApp has to be derived from the generic wxApp. At the startup of the application a method "OnInit" is called automatically. Therefore "OnInit" has to be overloaded to plug in the initialization stuff.

```
IMPLEMENT_APP(wxHelloApp)
```

is a macro and (alternativly) stands for

```
wxApp *wxCreateApp() { return new wxHelloApp; }
wxAppInitializer wxTheAppInitializer((wxAppInitializerFunction)
                                     wxCreateApp);
wxHelloApp& wxGetApp() { return *(wxHelloApp*)wxTheApp; }
```

`wxTheAppInitializer` and `wxCreateApp` are used by the library itself. The global function `wxGetApp()` can be used to get a pointer to the (one and only)

wxApp (respectively wxHelloApp)-Object. Now a window with a menu is needed. To get this, the application has to create a kind of wxFrame-Object. The specialized version wxHelloFrame is derived:

```
class wxHelloFrame : public wxFrame
{
public:
    wxHelloFrame(const wxString& title, const wxPoint& pos
                 , const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
};
```

The wxHello-Application should understand two commands:

1. "Quit" to exit and
2. "About" to display a message box with the program name

Therefore there will be two methods "OnQuit" and "OnAbout", which will be executed in case the menu item is selected. But this will take two steps:

1. If you select the menu item, an event is initiated
2. The event is tied to a method

Therefore two event numbers have to be defined:

```
enum
{
    ID_Quit=1,
    ID_About
};
```

Now the methods have to be implemented:

```
// Implementation----------------------------------------------

bool wxHelloApp::OnInit()
{
    wxHelloFrame *frame = new wxHelloFrame("Hello World"
                                , wxPoint(50,50), wxSize(450,350));

    // dynamic events-----------------------------------------
    frame->Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxHelloFrame::OnQuit );
    frame->Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxHelloFrame::OnAbout );
    //------------------------------------------------------------

    frame->Show(true);
```

```
    SetTopWindow(frame);
    return true;
}
```

As already mentioned "wxHelloApp::OnInit" is the first method being executed. Here the frame window with title "Hello World" will be created at a certain position and size. The "Connect" method dynamically connects an event to a method. If the event occurs, the method is executed. But something has to raise an event... This can be done with a button or through a menu item. If the frame is created, its constructor is executed:

```
wxHelloFrame::wxHelloFrame(const wxString& title
                  , const wxPoint& pos, const wxSize& size)
    : wxFrame((wxFrame*)NULL,-1,title,pos,size)
{
    // create menubar
    wxMenuBar *menuBar = new wxMenuBar;
    // create menu
    wxMenu *menuFile = new wxMenu;
    // append menu entries
    menuFile->Append(ID_About,"&About...");
    menuFile->AppendSeparator();
    menuFile->Append(ID_Quit,"E&xit");
    // append menu to menubar
    menuBar->Append(menuFile,"&File");
    // set frame menubar
    SetMenuBar(menuBar);

    // create frame statusbar
    CreateStatusBar();
    // set statusbar text
    SetStatusText("Demo for wxWindows");
}
```

Here the menubar is created and menus are appended. A menu can have several menu items, which are associated to an event number. If a menu item is selected, the specified event occurs. The event method will then be executed. The & char marks the key shortcut. Here Alt–F will open the File menu and x will afterwards exit the application. The status bar is found at the bottom of the frame window and the message "Demo for wxWindows" is written there.

```
void wxHelloFrame::OnQuit(wxCommandEvent& event)
{
    Close(true);
}

void wxHelloFrame::OnAbout(wxCommandEvent& event)
{
```

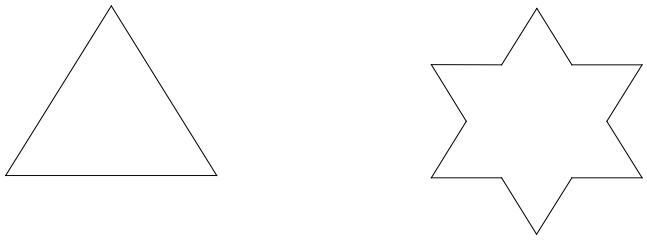
```
    wxMessageBox("wxWindows Hello World example."
                 ,"About Hello World", wxOK|wxICON_INFORMATION
                 , this);
}
```

"OnQuit" just closes the application and "OnAbout" shows a messagebox titled "About Hello World" with the text "wxWindows Hello World example.".

3 The Koch snowflake

3.1 What is a Koch snowflake?

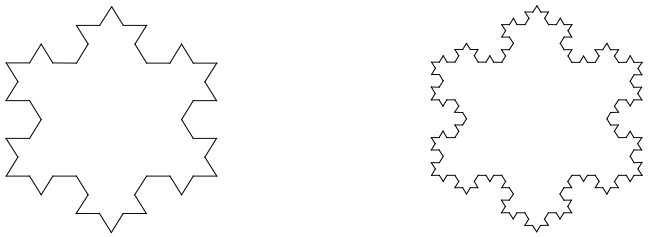
The Koch snowflake is a nice recursive figure. It starts with a equal sided triangle. This triangle shown in figure 1(a) will be referred to as a Koch snowflake of level 0. Every edge is divided in three parts of equal length.



(a) level 0 (b) level 1

Figure 1: Koch snowflakes level 0 & 1

The middle one is replaced with the two other edges of another equal sided triangle. This Koch snowflake of level 1 is shown in figure 1(b). The procedure is repeated and Koch snowflakes of level n are obtained recursivly (see 2).



(a) level 2 (b) level 3

Figure 2: Koch snowflakes level 2 & 3

3.2 wxKoch0: The first approach

A program showing a Koch snowflake for a given level will be developed. In contrast to the program wxHello in section 2, a separate header file contains the declarations:

```
// wxKoch0.h
// Martin Bernreuther, Jan. 2000
// wxWindows 2
// draw a Koch snowflake recursivly
// wxKoch0: no model and no repaint

#ifndef WXKOCHH
#define WXKOCHH

#include <wx/wx.h>

class wxKochFrame : public wxFrame
{
public:
    wxKochFrame(const wxString& title, const wxPoint& pos
                , const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    void OnInpLevel(wxCommandEvent& event);
};
```

```
class wxKochApp : public wxApp
{
    wxKochFrame *m_pframe;
    virtual bool OnInit();
public:
    void Draw(unsigned int n);
    void DrawEdge(wxDC& dc,unsigned int n,int x1,int y1
                                          ,int x2,int y2);
};

#endif
```

The wxKochApp stores the pointer to its frame m_pframe, which is created in the OnInit method. "Draw" will draw a Koch snowflake of level n and makes use of the DrawEdge method, which calls itself recursively.

Looking at the source file, the first part is similar to wxHello:

```
// wxKoch0.cpp
// Martin Bernreuther, Jan. 2000
// wxWindows 2

#include <wx/wx.h>
#include <math.h>

#include "wxKoch0.h"

IMPLEMENT_APP(wxKochApp)

enum
{
    ID_Quit=1,
    ID_About,
    ID_InpLevel
};

bool wxKochApp::OnInit()
{
    m_pframe = new wxKochFrame("Koch Snowflake", wxPoint(50,50)
                               , wxSize(450,350));
    m_pframe->Show(true);

    SetTopWindow(m_pframe);
    return true;
}

wxKochFrame::wxKochFrame(const wxString& title, const wxPoint& pos
                         , const wxSize& size)
    : wxFrame((wxFrame*)NULL,-1,title,pos,size)
{
```

```
    SetIcon(wxICON(KochIcon));

    // create menubar
    wxMenuBar *menuBar = new wxMenuBar;
    // create menu
    wxMenu *menuFile = new wxMenu;
    wxMenu *menuHelp = new wxMenu;
    // append menu entries
    menuFile->Append(ID_InpLevel,"Input &Level...\tCtrl-D");
    menuFile->AppendSeparator();
    menuFile->Append(ID_Quit,"E&xit");
    menuHelp->Append(ID_About,"&About...");
    // append menu to menubar
    menuBar->Append(menuFile,"&File");
    menuBar->Append(menuHelp,"&Help");
    // set frame menubar
    SetMenuBar(menuBar);

    // Connect event to callback function
    Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnQuit );
    Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnAbout );
    Connect( ID_InpLevel, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnInpLevel );

    // create frame statusbar
    CreateStatusBar();
    // set statusbar text
    SetStatusText("Generating Koch snowflakes");
}

void wxKochFrame::OnQuit(wxCommandEvent& event)
{
    Close(true);
}

void wxKochFrame::OnAbout(wxCommandEvent& event)
{
    wxMessageBox("Generating Koch snowflakes\nby M. Bernreuther"
                 ,"About wxKoch", wxOK|wxICON_INFORMATION, this);
}
```

To get the level an input of the user is required. The method wxKochApp::OnInpLevel is executed through the menu.

```
void wxKochFrame::OnInpLevel(wxCommandEvent& event)
{
    // long Result=wxGetNumberFromUser(Descriptionline,Label
    //              ,Init,Min,Max,ParentWindow,Defaultposition)
```

```
    long level = wxGetNumberFromUser( "","Level:", "Input Level"
                                      , 4, 0, 10, this );
    wxString msg;
    if ( level == -1 )
        msg = "Invalid number entered or dialog cancelled.";
    else
    {
        msg.Printf("Level: %lu",level);
        wxGetApp().Draw(level);
    }
    SetStatusText(msg);
}
```

For the input of an positive integer wxWindows offers the dialog wxGetNumberFromUser, which also can be found in the Dialogs sample. If a valid input exists, the Koch snowflake is drawn.

```
void wxKochApp::Draw(unsigned int n)
{
    int width,height;
    unsigned int d,x1,y1,x2,y3,x3;

    // determine size of Window
    m_pframe->GetClientSize(&width, &height);
    d=height;
    // calculate coordinates of initial triangle
    if(width<height) d=width;
    y1=.5*height+.25*d;
    y3=.5*height-.5*d;
    x1=.5*width-.25*d*sqrt(3.);
    x2=.5*width+.25*d*sqrt(3.);
    x3=.5*width;

    // Initialize device context
    wxClientDC dc(m_pframe);
    dc.Clear();
    dc.SetPen(wxPen(wxColour(0,0,0), 1, wxSOLID));
    dc.BeginDrawing();
    // draw the edges
    DrawEdge(dc,n,x1,y1,x2,y1);
    DrawEdge(dc,n,x2,y1,x3,y3);
    DrawEdge(dc,n,x3,y3,x1,y1);
    dc.EndDrawing();
}

void wxKochApp::DrawEdge(wxDC& dc,unsigned int n,int x1,int y1
                                                ,int x2,int y2)
{
    if(n>0)
    {
```

```
        // calculate new coordinates
        int x3,y3,x4,y4,x5,y5;
        x3=2.*x1/3.+x2/3.;
        y3=2.*y1/3.+y2/3.;
        DrawEdge(dc,n-1,x1,y1,x3,y3);
        x4=x1/3.+2.*x2/3.;
        y4=y1/3.+2.*y2/3.;
        x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
        y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
        // recursive calls
        DrawEdge(dc,n-1,x3,y3,x5,y5);
        DrawEdge(dc,n-1,x5,y5,x4,y4);
        DrawEdge(dc,n-1,x4,y4,x2,y2);
    }
    else
        // just draw a line
        dc.DrawLine(x1,y1,x2,y2);
}
```

The application will draw to the frame client window. The size of the window and the coordinates of the corners of the initial triangle are determined. Drawing operations need a device context. Here a wxClientDC is created and used. "Clear" first will result in an empty area and SetPen will specify the pen, which will be used for drawing. A black solid pen with width 1 is created here for this purpose. DrawEdge will receive the drawing context by reference and will do the recursion and drawing using DrawLine.

4 Model-View-Controller Concept

4.1 Overview

Following the Model-View-Controller (MVC) concept, an application consists of three parts (see 3), which communicate through messages.

The View and the Controller part are grown together in most applications. Important is the model part, which can be modelled as own class in C++. The properties of the model class represent the data used in the application. This can be drawing objects in a CAD like application or text and formatting information in a word processor. Since the MFC[1] was written for applications like MS-Word in the first place, a model is called "Document" there. wxWindows offers a MVC like environment, where the model also is called "document". This won't be used here. More information can be obtained looking at the help and the doc* samples.

[1]Microsoft Foundation Classes: a commercial GUI/Class library

An application can have multiple instances of model data or just one at a time. If there are multiple[2] allowed, usually there is an active one, which receives the changes.

Regarding the views, the window manager of the Windows operating system doesn't store the areas overdrawn through other windows in the front. The application has to take care of restore the invalid areas itself. There is an repaint event which is release from the window manager to tell the view (through the application) to restore the area.

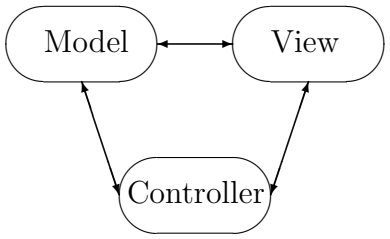


Figure 3: MVC concept

4.2 wxKoch1: Introducing a model and view class

Taking wxKoch0 (see 3.2), a model and a view class should be added like described in 4. The view class is a very specialized one and is able to draw Koch snowflakes. It might be better to have a general view and move the functionality to the model.

```
class KochView;

class KochModel
{
private:
    // level:
    unsigned int m_n;
    // the one and only view:
    KochView *m_pview;

public:
    KochModel();
    ~KochModel();

    void Setn(unsigned int n);
    unsigned int Getn() { return m_n; }

    void AddView(KochView *pview);
    void RemoveView(KochView *pview);

    void Draw();
};

class KochView : public wxWindow
```

[2]this is often referred to as MDI (Multiple Document Interface)

```
{
private:
    // the model the view belongs to:
    KochModel *m_pmodel;
    wxPen m_pen;
    unsigned int m_x1,m_y1,m_x2,m_x3,m_y3;

    void CalcStartCoordinates(unsigned int width
                              ,unsigned int height
                              ,int& x1,int& y1
                              ,int& x2,int& x3,int& y3);

    void Draw(wxDC& dc,int x1,int y1,int x2,int x3,int y3);
    void DrawEdge(wxDC& dc,unsigned int n
                  ,int x1,int y1,int x2,int y2);

public:
    KochView(wxWindow *pparent, KochModel *pmodel);
    virtual ~KochView();
    void SetStatusText(wxString text)
        { ((wxFrame*)GetParent())->SetStatusText(text); }
    void Draw() { Refresh(); }
    void OnPaint(wxPaintEvent& event);
};
```

The application class stores a pointer to the actual model:

```
class KochApp : public wxApp
{
    KochFrame *m_pframe;
    KochModel *m_pmodel;
    virtual bool OnInit();
    virtual int OnExit();
public:
    KochFrame* GetpFrame() { return m_pframe; }
    KochModel* GetpModel() { return m_pmodel; }
    void SetStatusText(wxString text)
        { if(m_pframe) m_pframe->SetStatusText(text); }
};
```

which is created in KochApp::OnInit through:

```
    m_pmodel=new KochModel();
```

and deleted in KochApp::OnExit, which is called from wxWindows at the program termination:

```
int KochApp::OnExit()
{
    if(m_pmodel) delete m_pmodel;
    // application return code
```

```
    return 0;
}
```

The model creates its own view:

```
KochModel::KochModel()
{
    m_pview=NULL;
    m_n=4;
    new KochView(wxGetApp().GetpFrame(), this);
}

KochModel::~KochModel()
{
    if(m_pview)
        m_pview->Destroy();
}
```

Instead of deleting the view, the Destroy() or Close() method should be used instead. The KochView automatically registers and unregisters itself at KochModel in the constructor resp. destructor

```
KochView::KochView(wxWindow *pparent, KochModel *pmodel):
 wxWindow(pparent, -1), m_pmodel(pmodel)
{
    wxColour col(0,0,0);
    m_pen = wxPen(col, 1, wxSOLID);

    if(m_pmodel) m_pmodel->AddView(this);

    Connect( -1, wxEVT_PAINT
             , (wxObjectEventFunction) &KochView::OnPaint );
}

KochView::~KochView()
{
    if(m_pmodel) m_pmodel->RemoveView(this);
}
```

using KochModel::AddView resp. KochModel::RemoveView

```
void KochModel::AddView(KochView *pview)
{
    if(m_pview) delete m_pview;
    m_pview=pview;
}

void KochModel::RemoveView(KochView *pview)
{
    if(pview==m_pview) m_pview=NULL;
}
```

KochModel::Setn just sets a new level, which is the data the model has to store:

```
void KochModel::Setn(unsigned int n)
{
    if(n!=m_n)
    {
        m_n=n;
        if(m_pview)
        {
            wxString msg;
            msg.Printf("Level changed to %lu", m_n );
            wxGetApp().SetStatusText(msg);
            m_pview->Draw();
        }
    }
}
```

It is called from KochFrame::OnInpLevel. Since the view is responsible for the visualization, KochModel::Draw just calls the KochView::Draw method:

```
void KochModel::Draw()
{
    if(m_pview) m_pview->Draw();
}
```

The view receives the repaint event to update the drawing area through KochView::OnPaint. A pen is needed for the drawing procedure. The data comes from the model.

```
KochView::KochView(wxWindow *pparent, KochModel *pmodel):
 wxWindow(pparent, -1), m_pmodel(pmodel)
{
    wxColour col(0,0,0);
    m_pen = wxPen(col, 1, wxSOLID);

    if(m_pmodel) m_pmodel->SetView(this);

    Connect( -1, wxEVT_PAINT
            , (wxObjectEventFunction) &KochView::OnPaint );
}
```

The calculation of the first three vertices coordinates of the triangle is done in KochView::CalcStartCoordinates depending on the size of the view. The drawing is started with KochView::Draw and done in the recursive KochView::OnPaint:

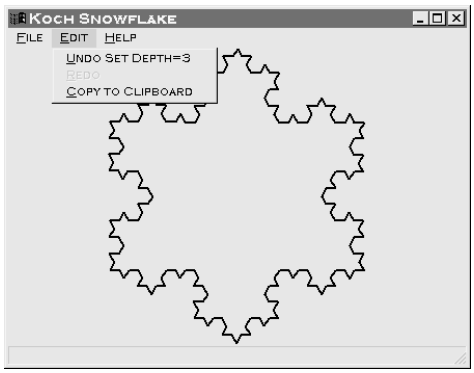
```
void KochView::CalcStartCoordinates(unsigned int width
                                    ,unsigned int height
                                    ,int& x1,int& y1
                                    ,int& x2,int& x3,int& y3)
{
```

```
    if(width<height)
    {
        y1=.5*height+.25*width;
        y3=.5*(height-width);
        x1=(.5-.25*sqrt(3.))*width;
        x2=(.5+.25*sqrt(3.))*width;
    }
    else
    {
        y1=.75*height;
        y3=0.;
        x1=.5*width-.25*height*sqrt(3.);
        x2=.5*width+.25*height*sqrt(3.);
    }
    x3=.5*width;
}

void KochView::Draw(wxDC& dc,int x1,int y1
                    ,int x2,int x3,int y3)
{
    if (!m_pmodel) return;

    int n=m_pmodel->Getn();

    DrawEdge(dc,n,x1,y1,x2,y1);
    DrawEdge(dc,n,x2,y1,x3,y3);
    DrawEdge(dc,n,x3,y3,x1,y1);
}

void KochView::DrawEdge(wxDC& dc,unsigned int n
                        ,int x1,int y1,int x2,int y2)
{
    if(n>0)
    {
        int x3,y3,x4,y4,x5,y5;
        x3=2.*x1/3.+x2/3.;
        y3=2.*y1/3.+y2/3.;
        DrawEdge(dc,n-1,x1,y1,x3,y3);
        x4=x1/3.+2.*x2/3.;
        y4=y1/3.+2.*y2/3.;
        x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
        y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
        DrawEdge(dc,n-1,x3,y3,x5,y5);
        DrawEdge(dc,n-1,x5,y5,x4,y4);
        DrawEdge(dc,n-1,x4,y4,x2,y2);
    }
    else
        dc.DrawLine(x1,y1,x2,y2);
}
```

```
void KochView::OnPaint(wxPaintEvent& event)
{
    if(m_pmodel)
    {
        int width,height;
        GetClientSize(&width, &height);
        int x1,y1,x2,x3,y3;
        CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);

        wxPaintDC dc(this);
        dc.Clear();
        dc.SetPen(m_pen);
        Draw(dc,x1,y1,x2,x3,y3);
    }
}
```

KochView::OnPaint is called every time a repaint event is released.

5 wxKoch: The final version



5.1 Added features

5.1.1 Change line thickness and color

The line thickness is described with an integer number. The input is therefore analog to the input of the level. To choose a color, wxWindows offers a color dialog, which is used here. For common dialogs look at the "dialogs" sample.

5.1.2 Printing and Clipboard copy

wxWindows offers special device contexts for printing and the clipboard. Instead of drawing to the screen, KochView::Draw draws to the printer or in the clipboard.

5.1.3 Undo/Redo

Undo/Redo is implemented through classes for each command. The class CmdSetLevel for example sets and restores the level with its Do and Undo method. The command processor object manages all command objects.

5.2 The sources

5.2.1 wxKoch.h

```
// wxKoch.h
// Martin Bernreuther, July 2000
// using wxWindows 2.2.0
// draws & prints the recursive Koch snowflake structure

// wxKoch1 -> wxKoch: added the possibility of
//       setting Pen width & color
//       printing capabilities
//       the possibility of copying to the clipboard
//       Undo/Redo


#ifndef WXKOCHH
#define WXKOCHH

#include <wx/wx.h>
// define wxCommand:
#include <wx/docview.h>

class KochView;

class KochModel
{
private:
    // level:
    unsigned int m_n;
    // the one and only view:
    KochView *m_pview;

public:
    KochModel();
```

```
    ~KochModel();

    void Setn(unsigned int n);
    unsigned int Getn() { return m_n; }

    void AddView(KochView *pview);
    void RemoveView(KochView *pview);

    void Draw();

    unsigned int GetPenWidth();
    wxColor GetPenColor();
    void SetPenWidth(int w);
    void SetPenColor(wxColor c);
    void Print();
    void Copy2Clipboard();
};

class KochView : public wxWindow
{
private:
    // the model the view belongs to:
    KochModel *m_pmodel;
    wxPen m_pen;
    wxFont m_font;
    unsigned int m_x1,m_y1,m_x2,m_x3,m_y3;

    void CalcStartCoordinates(unsigned int width
                              ,unsigned int height
                              ,int& x1,int& y1
                              ,int& x2,int& x3,int& y3);

    void Draw(wxDC& dc,int x1,int y1,int x2,int x3,int y3);
    void DrawEdge(wxDC& dc,unsigned int n
                  ,int x1,int y1,int x2,int y2);

public:
    KochView(wxWindow *pparent, KochModel *pmodel);
    virtual ~KochView();
    void SetStatusText(wxString text)
        { ((wxFrame*)GetParent())->SetStatusText(text); }
    void Draw() { Refresh(); }
    void OnPaint(wxPaintEvent& event);

    unsigned int GetPenWidth();
    wxColour GetPenColor();
    void SetPenWidth(int w);
    void SetPenColor(wxColor c);
    void Print();
```

```
    void Copy2Clipboard();
};


class KochFrame : public wxFrame
{
private:
    wxCommandProcessor m_cmdprocessor;
public:
    KochFrame(const wxString& title, const wxPoint& pos
                                    , const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    void OnInpLevel(wxCommandEvent& event);

    void OnSetPenWidth(wxCommandEvent& event);
    void OnSetPenColor(wxCommandEvent& event);
    void OnPrint(wxCommandEvent& event);
    void OnCopyClipboard(wxCommandEvent& event);
    void OnUndo(wxCommandEvent& event);
    void OnRedo(wxCommandEvent& event);
};


class KochApp : public wxApp
{
private:
    KochFrame *m_pframe;
    KochModel *m_pmodel;
    wxColourData m_coldata;
    wxPrintDialogData m_prndata;
    virtual bool OnInit();
    virtual int OnExit();
public:
    KochFrame* GetpFrame() { return m_pframe; }
    KochModel* GetpModel() { return m_pmodel; }
    void SetStatusText(wxString text)
        { if(m_pframe) m_pframe->SetStatusText(text); }
    wxColourData* GetpColdata() { return &m_coldata; }
    wxPrintDialogData* GetpPrndata() { return &m_prndata; }
};

class CmdSetLevel : public wxCommand
{
private:
    unsigned int m_level, m_oldlevel;
public:
    CmdSetLevel(unsigned int d)
        : wxCommand(TRUE,"Input_Level"),m_level(d) {}
```

```
    virtual bool Do();
    virtual bool Undo();
};

class CmdSetPenWidth : public wxCommand
{
private:
    unsigned int m_penwidth, m_oldpenwidth;
public:
    CmdSetPenWidth(unsigned int w)
        : wxCommand(TRUE,"Set Pen Width"),m_penwidth(w) {}
    virtual bool Do();
    virtual bool Undo();
};

class CmdSetPenColor : public wxCommand
{
private:
    wxColourData m_coldata, m_oldcoldata;
public:
    CmdSetPenColor(const wxColourData& c)
        : wxCommand(TRUE,"Set Pen Color"),m_coldata(c) {}
    virtual bool Do();
    virtual bool Undo();
};

#endif
```

5.2.2 wxKoch.cpp

```
// wxKoch.cpp
// Martin Bernreuther, July 2000
// using wxWindows 2.2.0

#include <wx/wx.h>
#include <math.h>

#include <wx/colordlg.h>
#include <wx/printdlg.h>
#include <wx/metafile.h>
#include <wx/clipbrd.h>


#include "wxKoch.h"

IMPLEMENT_APP(KochApp)

enum
{
```

```
    ID_Quit=1,
    ID_About,
    ID_InpLevel,

    ID_SetPenWidth,
    ID_SetPenColor,
    ID_CopyClipboard,
    ID_Print
};

//KochApp----------------------------------------------------------

bool KochApp::OnInit()
{
    m_pframe = new KochFrame("Koch_Snowflake", wxPoint(50,50)
                             , wxSize(450,350));
    m_pframe->Show(true);
    SetTopWindow(m_pframe);
    m_pmodel = new KochModel();

    // initialize ColourDialogData
    m_coldata.SetChooseFull(TRUE);
    for (int i = 0; i < 16; i++)
    {
        wxColour col(i*16, i*16, i*16);
        m_coldata.SetCustomColour(i, col);
    }

    // initialize PrintDialogData
    m_prndata.EnableSelection(FALSE);
    m_prndata.EnablePageNumbers(FALSE);

    return true;
}


int KochApp::OnExit()
{
    if(m_pmodel) delete m_pmodel;
    // application return code
    return 0;
}

//KochFrame--------------------------------------------------------

KochFrame::KochFrame(const wxString& title
                     , const wxPoint& pos, const wxSize& size)
        : wxFrame((wxFrame*)NULL,-1,title,pos,size)
{
```

```
SetIcon(wxICON(KochIcon));

// create menubar
wxMenuBar *menuBar = new wxMenuBar;
// create menu
wxMenu *menuFile = new wxMenu;
wxMenu *menuEdit = new wxMenu;
wxMenu *menuHelp = new wxMenu;
// append menu entries
menuFile->Append(ID_InpLevel,"Input &Level...\tCtrl-D");
menuFile->Append(ID_SetPenWidth,"Set Pen &Width...");
menuFile->Append(ID_SetPenColor,"Set Pen &Color...");
menuFile->Append(ID_Print,"&Print...\tCtrl-P");
menuFile->AppendSeparator();
menuFile->Append(ID_Quit,"E&xit\tAlt-F4");
menuEdit->Append(wxID_UNDO,"&Undo");
menuEdit->Append(wxID_REDO,"&Redo");
menuEdit->Append(ID_CopyClipboard,"&Copy to Clipboard");
menuHelp->Append(ID_About,"&About...");
// append menu to menubar
menuBar->Append(menuFile,"&File");
menuBar->Append(menuEdit,"&Edit");
menuBar->Append(menuHelp,"&Help");
// add Undo/Redo entries to menuEdit & Initialize
m_cmdprocessor.SetEditMenu(menuEdit);
m_cmdprocessor.Initialize();
// set frame menubar
SetMenuBar(menuBar);

Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnQuit );
Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnAbout );
Connect( ID_InpLevel, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnInpLevel );
Connect( ID_SetPenWidth, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnSetPenWidth );
Connect( ID_SetPenColor, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnSetPenColor );
Connect( ID_Print, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnPrint );
Connect( ID_CopyClipboard, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnCopyClipboard );
Connect( wxID_UNDO, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnUndo );
Connect( wxID_REDO, wxEVT_COMMAND_MENU_SELECTED
         ,(wxObjectEventFunction) &KochFrame::OnRedo );

// create frame statusbar
```

```
    CreateStatusBar();
    // set statusbar text
    SetStatusText("Generating Koch snowflakes");
}

void KochFrame::OnQuit(wxCommandEvent& event)
{
    Close(true);
}

void KochFrame::OnAbout(wxCommandEvent& event)
{
    wxMessageBox("Generating Koch snowflakes\nby M. Bernreuther"
                 ,"About Koch", wxOK|wxICON_INFORMATION, this);
}

void KochFrame::OnInpLevel(wxCommandEvent& event)
{
    // long Result=wxGetNumberFromUser(Descriptionline,Label
    //              ,Init,Min,Max,ParentWindow,Defaultposition)
    long res = wxGetNumberFromUser( "","Level:", "Input Level"
               , wxGetApp().GetpModel()->Getn() , 0, 10, this );
    wxString msg;
    if ( res == -1 )
        msg = "Invalid number entered or dialog cancelled.";
    else
    {
        msg.Printf("Level: %lu", res );
        m_cmdprocessor.Submit(new CmdSetLevel(res));
        wxGetApp().GetpModel()->Draw();
    }
    SetStatusText(msg);
}

void KochFrame::OnSetPenWidth(wxCommandEvent& event)
{
    long res = wxGetNumberFromUser( "","Pen Width:", "Input Pen Width"
               , wxGetApp().GetpModel()->GetPenWidth() , 0, 10, this );
    wxString msg;
    if ( res == -1 )
        msg = "Invalid number entered or dialog cancelled.";
    else
    {
        msg.Printf("Pen width: %lu", res );
        m_cmdprocessor.Submit(new CmdSetPenWidth(res));
        wxGetApp().GetpModel()->Draw();
    }
    SetStatusText(msg);
}
```

```
void KochFrame::OnSetPenColor(wxCommandEvent& event)
{
    wxColourDialog coldialog(this, wxGetApp().GetpColdata());
    wxString msg;
    if (coldialog.ShowModal() == wxID_OK)
    {
        msg="Pen Color set";
        m_cmdprocessor.Submit(new CmdSetPenColor(coldialog.GetColourData()));
        wxGetApp().GetpModel()->Draw();
    }
    else
        msg="Color Dialog canceled";
    SetStatusText(msg);
}

void KochFrame::OnPrint(wxCommandEvent& event)
{
    wxGetApp().GetpModel()->Print();
}

void KochFrame::OnCopyClipboard(wxCommandEvent& event)
{
    wxGetApp().GetpModel()->Copy2Clipboard();
}

void KochFrame::OnUndo(wxCommandEvent& event)
{
    m_cmdprocessor.Undo();
}

void KochFrame::OnRedo(wxCommandEvent& event)
{
    m_cmdprocessor.Redo();
}

//KochModel-----------------------------------------------------------

KochModel::KochModel()
{
    m_pview=NULL;
    m_n=4;
    /*m_pview=*/
    new KochView(wxGetApp().GetpFrame(),this);
}


KochModel::~KochModel()
{
```

```
    if(m_pview)
        //delete m_pview;
        m_pview->Destroy();
        //m_pview->Close();
}

void KochModel::Setn(unsigned int n)
{
    if(n!=m_n)
    {
        m_n=n;
        if(m_pview)
        {
            wxString msg;
            msg.Printf("Level changed to %lu", m_n );
            wxGetApp().SetStatusText(msg);
            m_pview->Draw();
        }
    }
}

void KochModel::AddView(KochView *pview)
{
    if(m_pview) delete m_pview;
    m_pview=pview;
}

void KochModel::RemoveView(KochView *pview)
{
    if(pview==m_pview) m_pview=NULL;
}

void KochModel::SetPenWidth(int w)
{
    if(m_pview) m_pview->SetPenWidth(w);
}

unsigned int KochModel::GetPenWidth()
{
    unsigned int w=0;
    if(m_pview) w=m_pview->GetPenWidth();
    return w;
}

void KochModel::SetPenColor(wxColor c)
{
    if(m_pview) m_pview->SetPenColor(c);
}
```

```
wxColor KochModel::GetPenColor()
{
    if(m_pview)
        return m_pview->GetPenColor();
    else
        return wxNullColour;
}

void KochModel::Draw()
{
    if(m_pview) m_pview->Draw();
}

void KochModel::Print()
{
    if(m_pview) m_pview->Print();
}

void KochModel::Copy2Clipboard()
{
    if(m_pview) m_pview->Copy2Clipboard();
}

//KochView-------------------------------------------------------------

KochView::KochView(wxWindow *pparent, KochModel *pmodel):
 wxWindow(pparent, -1), m_pmodel(pmodel)
{
    SetPenColor(wxGetApp().GetpColdata()->GetColour());

    if(m_pmodel) m_pmodel->AddView(this);

    Connect( -1, wxEVT_PAINT
             , (wxObjectEventFunction) &KochView::OnPaint );
}


KochView::~KochView()
{
    if(m_pmodel) m_pmodel->RemoveView(this);
}

void KochView::SetPenWidth(int w)
{
    m_pen.SetWidth(w);
    Refresh();
}

unsigned int KochView::GetPenWidth()
```

```
{
    return m_pen.GetWidth();
}

void KochView::SetPenColor(wxColor c)
{
    m_pen.SetColour(c);
    Refresh();
}

wxColour KochView::GetPenColor()
{
    return m_pen.GetColour();
}

void KochView::CalcStartCoordinates(unsigned int width
                                    , unsigned int height
                                    , int& x1, int& y1
                                    , int& x2, int& x3, int& y3)
{
    if(width<height)
    {
        y1=.5*height+.25*width;
        y3=.5*(height-width);
        x1=(.5-.25*sqrt(3.))*width;
        x2=(.5+.25*sqrt(3.))*width;
    }
    else
    {
        y1=.75*height;
        y3=0.;
        x1=.5*width-.25*height*sqrt(3.);
        x2=.5*width+.25*height*sqrt(3.);
    }
    x3=.5*width;
}

void KochView::Draw(wxDC& dc,int x1,int y1
                            ,int x2,int x3,int y3)
{
    if (!m_pmodel) return;

    int n=m_pmodel->Getn();

    DrawEdge(dc,n,x1,y1,x2,y1);
    DrawEdge(dc,n,x2,y1,x3,y3);
    DrawEdge(dc,n,x3,y3,x1,y1);
}
```

```
void KochView::DrawEdge(wxDC& dc,unsigned int n
                        ,int x1,int y1,int x2,int y2)
{
    if(n>0)
    {
        int x3,y3,x4,y4,x5,y5;
        x3=2.*x1/3.+x2/3.;
        y3=2.*y1/3.+y2/3.;
        DrawEdge(dc,n-1,x1,y1,x3,y3);
        x4=x1/3.+2.*x2/3.;
        y4=y1/3.+2.*y2/3.;
        x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
        y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
        DrawEdge(dc,n-1,x3,y3,x5,y5);
        DrawEdge(dc,n-1,x5,y5,x4,y4);
        DrawEdge(dc,n-1,x4,y4,x2,y2);
    }
    else
        dc.DrawLine(x1,y1,x2,y2);
}

void KochView::OnPaint(wxPaintEvent& event)
{
    if(m_pmodel)
    {
        int width,height;
        GetClientSize(&width, &height);
        int x1,y1,x2,x3,y3;
        CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);

        wxPaintDC dc(this);
        dc.Clear();
        dc.SetPen(m_pen);
        Draw(dc,x1,y1,x2,x3,y3);
    }
}

void KochView::Print()
{
    wxWindow *pparent=wxGetApp().GetpFrame();
    wxPrintDialogData *pprndata=wxGetApp().GetpPrndata();
    wxPrintDialog prndialog(pparent,pprndata);
    if(prndialog.ShowModal()!=wxID_OK)
    {
        wxGetApp().SetStatusText("Printer␣Dialog␣cancelled");
        return;
    }
    *pprndata = prndialog.GetPrintDialogData();
    wxDC *pdc = prndialog.GetPrintDC();
```

```
    if(!pdc->Ok())
    {
        wxGetApp().SetStatusText(
                        "Error creating device context");
        return;
    }

    int width,height;
    pdc->GetSize(&width, &height);
    int x1,y1,x2,x3,y3;
    CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);

    pdc->SetPen(m_pen);
    m_font.SetPointSize(int(height/100));
    pdc->SetFont(m_font);
    pdc->SetTextForeground(wxColour(0,0,0));

    int xpos,ypos,twidth,theight;
    xpos=width/20;
    ypos=height/15;
    wxString msg="Koch snowflake";
    pdc->GetTextExtent(msg,&twidth,&theight);

    pdc->StartDoc("wxKoch printout");
    pdc->StartPage();

    pdc->DrawText(msg,xpos,ypos);

    ypos+=theight+5;
    msg.Printf("Depth %lu",wxGetApp().GetpModel()->Getn());
    //pdc->GetTextExtent(msg,&twidth,&theight);
    pdc->DrawText(msg,xpos,ypos);

    pdc->GetFont().SetPointSize(int(height/200));
    msg="by M. Bernreuther";
    pdc->GetTextExtent(msg,&twidth,&theight);
    xpos=.99*width-twidth;
    ypos=.99*height-theight;
    pdc->DrawText(msg,xpos,ypos);

    Draw(*pdc,x1,y1,x2,x3,y3);

    pdc->EndPage();
    pdc->EndDoc();
}


void KochView::Copy2Clipboard()
{
```

```
    int x1,y1,x2,x3,y3;
    CalcStartCoordinates(1000,1000,x1,y1,x2,x3,y3);

    wxEnhMetaFileDC dc;
    if(!dc.Ok())
    {
        wxGetApp().SetStatusText("wxMetafileDC not Ok");
        return;
    }

    dc.SetPen(m_pen);
    Draw(dc,x1,y1,x2,x3,y3);

    wxEnhMetaFile *pmf = dc.Close();
    if (!pmf)
    {
        wxGetApp().SetStatusText("Could not create wxMetafile");
        return;
    }
    if(pmf->Ok())
    {
        if(wxTheClipboard->Open())
        {
            if(wxTheClipboard->SetData(new wxEnhMetaFileDataObject(*pmf)))
                wxGetApp().SetStatusText("Clipboard data set");
            else
                wxGetApp().SetStatusText("Error setting Clipboard data");
            wxTheClipboard->Close();
        }
        else
            wxGetApp().SetStatusText(
                "Error opening Clipboard");
    }
    else
        wxGetApp().SetStatusText("wxMetafile not Ok");

    delete pmf;
}

//CmdSetDepth-----------------------------------------------------

bool CmdSetLevel::Do()
{
    m_oldlevel=wxGetApp().GetpModel()->Getn();
    wxGetApp().GetpModel()->Setn(m_level);
    m_commandName.Printf("Set Depth=%lu",m_level);
    return TRUE;
}
```

```
bool CmdSetLevel::Undo()
{
    wxGetApp().GetpModel()->Setn(m_oldlevel);
    return TRUE;
}

bool CmdSetPenWidth::Do()
{
    m_oldpenwidth=wxGetApp().GetpModel()->GetPenWidth();
    wxGetApp().GetpModel()->SetPenWidth(m_penwidth);
    m_commandName.Printf("Set Pen Width=%lu",m_penwidth);
    return TRUE;
}

bool CmdSetPenWidth::Undo()
{
    wxGetApp().GetpModel()->SetPenWidth(m_oldpenwidth);
    return TRUE;
}

bool CmdSetPenColor::Do()
{
    wxColourData *pcoldata=wxGetApp().GetpColdata();
    m_oldcoldata=*pcoldata;
    *pcoldata = m_coldata;
    wxGetApp().GetpModel()->SetPenColor(pcoldata->GetColour());
    return TRUE;
}

bool CmdSetPenColor::Undo()
{
    wxColourData *pcoldata=wxGetApp().GetpColdata();
    *pcoldata = m_oldcoldata;
    wxGetApp().GetpModel()->SetPenColor(pcoldata->GetColour());
    return TRUE;
}
```

6 Other C++ GUI-/Class-libraries

Qt In contrast to wxWindows the Qt library (`http://www.trolltech.com/products/qt/qt.html`) is a commercial product. For the Unix version there is a free edition (`http://www.trolltech.com/products/download/freelicense/`), which the K Desktop Environment `http://www.kde.org/` is based on.

GTK+ The Gimp ToolKit `http://www.gtk.org/` is a GUI toolkit based on C. The gnome desktop environment (`http://www.gnome.org/`) uses

gtk+. A Windows port exists (http://www.gimp.org/~tml/gimp/win32/), but is under development. Gtk– http://gtkmm.sourceforge.net/ is a C++ interface for gtk+.

FLTK The Fast Light Tool Kit Home Page can be found at http://www.fltk.org/.

V The V Homepage can be found at http://www.objectcentral.com/vgui/vgui.htm

Amulet The Amulet Homepage can be found at http://www.cs.cmu.edu/afs/cs/project/amulet/www/amulet-home.html

YACL Yet Another Class Library (http://www.cs.sc.edu/~sridhar/yacl/) is not developed any longer.

An overview of GUI libraries can be found at http://www.geocities.com/SiliconValley/Vista/7184/guitool.html.

You can also do the GUI programming in Java (http://www.sun.com/java/)...