


```
void OnEnter1(wxCommandEvent& event);
void OnEnter2(wxCommandEvent& event);
void OnEnter3(wxCommandEvent& event);
```

private:

```
// begin wxGlade: MyFrame::methods
void set_properties();
void do_layout();
// end wxGlade
```

protected:

```
// begin wxGlade: MyFrame::attributes
wxCheckBox* checkbox_1;
wxCheckBox* checkbox_2;
wxCheckBox* checkbox_3;
wxCheckBox* checkbox_4;
wxCheckBox* checkbox_5;
wxCheckBox* checkbox_6;
wxCheckBox* checkbox_7;
wxCheckBox* checkbox_8;
wxCheckBox* checkbox_9;
wxButton* button_1;
wxButton* button_2;

wxStaticBox* sizer_3_staticbox;
wxStaticText* label_3;
wxTextCtrl* tctrl_3;
wxStaticText* label_4;
wxTextCtrl* tctrl_4;
wxStaticText* label_5;
wxTextCtrl* tctrl_5;
// end wxGlade
```

```
}; // wxGlade: end class
```

```
#endif // TEST_H
```

```
.....
```

```
MyFrame.cpp
```

```
.....
```

```
// -*- C++ -*- generated by wxGlade HG on Tue May 29 16:10:07 2012
```

```
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <string>
```

```
#include "MyFrame.hpp"
#include "Angles_wx.hpp"
```

```
// begin wxGlade: ::extracode
// end wxGlade
```

```
const int ID_CHK_1 = 101;
const int ID_CHK_2 = 102;
const int ID_CHK_3 = 103;
const int ID_CHK_4 = 104;
const int ID_CHK_5 = 105;
const int ID_CHK_6 = 106;
const int ID_CHK_7 = 107;
const int ID_CHK_8 = 108;
const int ID_CHK_9 = 109;
const int ID_BTN_1 = 110;
const int ID_BTN_2 = 111;
```

```
const int ID_TXT_1 = 301;
const int ID_TXT_2 = 302;
const int ID_TXT_3 = 303;
```

```
int flags[20];
```

```
MyFrame::MyFrame(wxWindow* parent, int id, const wxString& title, const wxPoint& pos, const wxSize& size, long style):
```

```
    wxFrame(parent, id, title, pos, size, wxDEFAULT_FRAME_STYLE)
{
    // begin wxGlade: MyFrame::MyFrame
    checkbox_1 = new wxCheckBox(this, ID_CHK_1, wxT("draw_angle_tree"));
    checkbox_2 = new wxCheckBox(this, ID_CHK_2, wxT("plot_circle_angle"));
    checkbox_3 = new wxCheckBox(this, ID_CHK_3, wxT("plot_line_angle"));
    checkbox_4 = new wxCheckBox(this, ID_CHK_4, wxT("calc_statistics"));
    checkbox_5 = new wxCheckBox(this, ID_CHK_5, wxT("plot_statistics"));
    checkbox_6 = new wxCheckBox(this, ID_CHK_6, wxT("plot_residual_errors"));
    checkbox_7 = new wxCheckBox(this, ID_CHK_7, wxT("calc_uscale_statistics"));
    checkbox_8 = new wxCheckBox(this, ID_CHK_8, wxT("plot_uscale_statistics"));
    checkbox_9 = new wxCheckBox(this, ID_CHK_9, wxT("plot_uscale_residual_errors"));
    button_1 = new wxButton(this, ID_BTN_1, wxT("Execute"));
    button_2 = new wxButton(this, ID_BTN_2, wxT("Close"));

    Connect(ID_CHK_1, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck1));
    Connect(ID_CHK_2, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck2));
    Connect(ID_CHK_3, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck3));
    Connect(ID_CHK_4, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck4));
    Connect(ID_CHK_5, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck5));
    Connect(ID_CHK_6, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck6));
    Connect(ID_CHK_7, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck7));
    Connect(ID_CHK_8, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck8));
    Connect(ID_CHK_9, wxEVT_COMMAND_CHECKBOX_CLICKED, wxCommandEventHandler(MyFrame::OnCheck9));
    Connect(ID_BTN_1, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(MyFrame::OnButton1));
    Connect(ID_BTN_2, wxEVT_COMMAND_BUTTON_CLICKED, wxCommandEventHandler(MyFrame::OnButton2));
```

```
    sizer_3_staticbox = new wxStaticBox(this, -1, wxT("Input Parameters"));
```

```
    label_3 = new wxStaticText(this, wxID_ANY, wxT("nIter \t\t = "));
    label_4 = new wxStaticText(this, wxID_ANY, wxT("GnuTerm \t = "));
    label_5 = new wxStaticText(this, wxID_ANY, wxT("th \t\t\t = "));
```

```
    tctrl_3 = new wxTextCtrl(this, ID_TXT_1, wxEmptyString);
    tctrl_4 = new wxTextCtrl(this, ID_TXT_2, wxEmptyString);
    tctrl_5 = new wxTextCtrl(this, ID_TXT_3, wxEmptyString);
```

```
    Connect(ID_TXT_1, wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(MyFrame::OnEnter1));
    Connect(ID_TXT_2, wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(MyFrame::OnEnter2));
    Connect(ID_TXT_3, wxEVT_COMMAND_TEXT_UPDATED, wxCommandEventHandler(MyFrame::OnEnter3));
```

```
    set_properties();
    do_layout();
    // end wxGlade
}
```

```
void MyFrame::set_properties()
{
    // begin wxGlade: MyFrame::set_properties
    SetTitle(wxT("frame_1"));
    // end wxGlade
}
```

```
void MyFrame::do_layout()
```

```

{
// begin wxGlade: MyFrame::do_layout
wxStaticBoxSizer* sizer_3 = new wxStaticBoxSizer(sizer_3_staticbox, wxVERTICAL);

wxBoxSizer*      sizer_4 = new wxBoxSizer(wxVERTICAL);

wxBoxSizer*      sizer_7 = new wxBoxSizer(wxHORIZONTAL);
wxBoxSizer*      sizer_6 = new wxBoxSizer(wxHORIZONTAL);
wxBoxSizer*      sizer_5 = new wxBoxSizer(wxHORIZONTAL);

sizer_5->Add(label_3, 0, wxADJUST_MINSIZE, 0);
sizer_5->Add(tctrl_3, 0, wxADJUST_MINSIZE, 0);
sizer_4->Add(sizer_5, 1, wxEXPAND, 0);

sizer_6->Add(label_4, 0, wxADJUST_MINSIZE, 0);
sizer_6->Add(tctrl_4, 0, wxADJUST_MINSIZE, 0);
sizer_4->Add(sizer_6, 1, wxEXPAND, 0);

sizer_7->Add(label_5, 0, wxADJUST_MINSIZE, 0);
sizer_7->Add(tctrl_5, 0, wxADJUST_MINSIZE, 0);
sizer_4->Add(sizer_7, 1, wxEXPAND, 0);

sizer_3->Add(sizer_4, 1, wxEXPAND, 0);

// SetSizer(sizer_3);
// sizer_3->Fit(this);

wxBoxSizer* sizer_1 = new wxBoxSizer(wxVERTICAL);
sizer_1->Add(checkbox_1, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_2, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_3, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_4, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_5, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_6, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_7, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_8, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(checkbox_9, 0, wxADJUST_MINSIZE, 0);

sizer_1->Add(sizer_3, 1, wxADJUST_MINSIZE, 0);

sizer_1->Add(button_1, 0, wxADJUST_MINSIZE, 0);
sizer_1->Add(button_2, 0, wxADJUST_MINSIZE, 0);

// SetSizer(sizer_1);
// sizer_1->Fit(this);
// Layout();

SetSizer(sizer_1);
sizer_1->Fit(this);
Layout();

// end wxGlade
}

void MyFrame::OnCheck1(wxCommandEvent& event)
{
    if (checkbox_1->GetValue()) flags[0] = 1;
    else flags[0] = 0;
}

void MyFrame::OnCheck2(wxCommandEvent& event)
{
    if (checkbox_2->GetValue()) flags[1] = 1;
    else flags[1] = 0;
}

```

```

void MyFrame::OnCheck3(wxCommandEvent& event)
{
    if (checkbox_3->GetValue()) flags[2] = 1;
    else flags[2] = 0;
}

void MyFrame::OnCheck4(wxCommandEvent& event)
{
    if (checkbox_4->GetValue()) flags[3] = 1;
    else flags[3] = 0;
}

void MyFrame::OnCheck5(wxCommandEvent& event)
{
    if (checkbox_5->GetValue()) flags[4] = 1;
    else flags[4] = 0;
}

void MyFrame::OnCheck6(wxCommandEvent& event)
{
    if (checkbox_6->GetValue()) flags[5] = 1;
    else flags[5] = 0;
}

void MyFrame::OnCheck7(wxCommandEvent& event)
{
    if (checkbox_7->GetValue()) flags[6] = 1;
    else flags[6] = 0;
}

void MyFrame::OnCheck8(wxCommandEvent& event)
{
    if (checkbox_8->GetValue()) flags[7] = 1;
    else flags[7] = 0;
}

void MyFrame::OnCheck9(wxCommandEvent& event)
{
    if (checkbox_9->GetValue()) flags[8] = 1;
    else flags[8] = 0;
}

void MyFrame::OnButton1(wxCommandEvent& event)
{
    std::cout << std::endl;
    std::cout << "flags[0] = " << flags[0] << std::endl;
    std::cout << "flags[1] = " << flags[1] << std::endl;
    std::cout << "flags[2] = " << flags[2] << std::endl;
    std::cout << "flags[3] = " << flags[3] << std::endl;
    std::cout << "flags[4] = " << flags[4] << std::endl;
    std::cout << "flags[5] = " << flags[5] << std::endl;
    std::cout << "flags[6] = " << flags[6] << std::endl;
    std::cout << "flags[7] = " << flags[7] << std::endl;
    std::cout << "flags[8] = " << flags[8] << std::endl;

    testAngles(flags, 3, 0.0);
}

void MyFrame::OnButton2(wxCommandEvent& event)
{
    Close();
}

```

```

void MyFrame::OnEnter1(wxCommandEvent& event)
{
    long n;
    std::cout << (tctrl_3->GetValue()).mb_str() << "\n";
    tctrl_3->GetValue().ToLong(&n);
    std::cout << n << "\n";
}

void MyFrame::OnEnter2(wxCommandEvent& event)
{
    std::cout << atoi((tctrl_4->GetValue()).mb_str()) << "\n";
}

void MyFrame::OnEnter3(wxCommandEvent& event)
{
    double th;
    std::cout << (tctrl_5->GetValue()).mb_str() << "\n";
    tctrl_5->GetValue().ToDouble(&th);
    std::cout << th << "\n";
}

```

```

:::::::::::::
Angles_wx.hpp
:::::::::::::
int testAngles(int *flags, int nIter = 3, double th = 0.0);
:::::::::::::

```

```

Angles_wx.cpp
:::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>

using namespace std;

#include "cordic.hpp"
#include "Angles.hpp"

```

```
extern string GnuTerm;
```

```

//-----
// Purpose:
//
// Explore Angles Space using Class Angles
//
// Discussion:
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 2012.05.11
//
// Author:
//
// Young Won Lim
//
// Parameters:
//
//-----

```

```

int testAngles(int *flags, int nIter = 3, double th = 0.0)
{
    // -----
    // nIter   : Number of Iteration = Height of binary angle tree
    // nAngle  : Number of Angles    = Number of Leaf Nodes
    // -----

    int nAngle = 1 << nIter;

    GnuTerm = "wxt";

    std::cout << "-----\n";
    std::cout << "Angles_tb [nIter] [GnuTerm] [th]" << std::endl;
    std::cout << "-----\n";
    std::cout << "          nIter   = " << nIter << " ";
    std::cout << "          nAngle  = " << nAngle << std::endl;
    std::cout << "          GnuTerm = " << GnuTerm << std::endl;
    std::cout << "          th      = " << th << std::endl;
    std::cout << "-----\n";

    // -----
    // A   : contains the angles of leaf nodes in binary angle tree
    // All  : contains the angles of all nodes in binary angle tree
    // -----
    double *A, *All;
    int level, leaves;
    int i, j, k;

    A = (double *) malloc ((1<<nIter) * sizeof (double));
    All = (double *) malloc (2* (1<<nIter) * sizeof (double));

    for (j=0; j<nAngle; ++j) {
        A[j] = compute_angle(j, nIter);
        // std::cout << "A[" << j << "]= " << setw(12) << setprecision(8) << A[j] << std::endl;
    }

    for (i=0, k=0; i<=nIter; ++i) {
        level = i;
        leaves = 1 << level;

        // std::cout << "level = " << level << "leaves = " << leaves << std::endl;

        for (j=0; j<leaves; ++j) {
            All[j+k] = compute_angle(j, level);
            // std::cout << "All[" << j+k << "] = " << All[j+k] << std::endl;
        }

        k += leaves;
    }

    // -----
    // LeafAngles : Angles Class for leaf nodes only
    // AllAngles  : Angles Class for all nodes (internal nodes included)
    // -----
    Angles LeafAngles(A, nIter, nAngle);
    Angles AllAngles(All, nIter, 2*nAngle-1);

    // -----
    // Plot Binary Angle Tree
    // -----
    if (flags[0]) draw_angle_tree (nIter, nAngle);

    // -----

```

```

// Plot angle vectors on a unit circle
// -----
if (flags[1]) LeafAngles.plot_circle_angle();
if (flags[1]) AllAngles.plot_circle_angle();

// -----
// Plot angle vectors on a linear scale
// -----
if (flags[2]) LeafAngles.plot_line_angle();
if (flags[2]) AllAngles.plot_line_angle();

LeafAngles.setThreshold(th);
AllAngles.setThreshold(th);

// -----
// Find Angles Statistics --> member data
// -----
if (flags[3]) LeafAngles.calc_statistics();
if (flags[3]) AllAngles.calc_statistics();

// -----
// Plot Delta Distribution & Angle-Delta
// -----
if (flags[4]) LeafAngles.plot_statistics();
if (flags[4]) AllAngles.plot_statistics();

// -----
// plot residual errors
// Residuals-Angle Plot & Residuals-Index Plot
// -----
if (flags[5]) LeafAngles.plot_residual_errors();
if (flags[5]) AllAngles.plot_residual_errors();

// -----
// Calculate Uniform Scale Statistics --> member data
// -----
if (flags[6]) LeafAngles.calc_uscale_statistics(1.0, 1.0);
if (flags[6]) AllAngles.calc_uscale_statistics(1.0, 1.0);

// -----
// Plot Uniform Scale Statistics
// -----
if (flags[7]) LeafAngles.plot_uscale_statistics();
if (flags[7]) AllAngles.plot_uscale_statistics();

// -----
// Plot residue errors at the leaf node angles
// -----
if (flags[8]) LeafAngles.plot_uscale_residual_errors(1, 2);
if (flags[8]) AllAngles.plot_uscale_residual_errors(1, 2);

return 0;
}

```