void init()

{

HAL::Init();

Display::Init();

Keyboard::Init();

Menu::Init();

}

void update()

{

Menu::Input::Update();

Display::Update();

}

class Application : public wxApp

{

public:

virtual bool OnInit() wxOVERRIDE;

};

class Frame : public wxFrame

{

public:

Frame(const wxString& title);

void OnSize(wxCommandEvent &);

void OnQuit(wxCommandEvent &);

void OnSCPI(wxCommandEvent &);

void OnAbout(wxCommandEvent &);

void OnTimer(wxTimerEvent &);

void OnTimerLong(wxTimerEvent &);

void OnClose(wxCloseEvent &);

static Frame \*Self();

private:

wxTimer timer;

// Таймер для "длинного нажатия" кнопок

wxTimer timerLongPress;

void DrawFPS();

void HandlerEvents();

public:

void OnDown(wxCommandEvent &event);

void OnUp(wxCommandEvent &event);

static Frame \*self;

};

enum

{

FILE\_SIZE = wxID\_HIGHEST + 1,

FILE\_QUIT = wxID\_EXIT,

TOOL\_SCPI,

HELP\_ABOUT = wxID\_ABOUT

};

wxIMPLEMENT\_APP\_NO\_MAIN(Application);

Frame \*Frame::self = nullptr;

int main(int argc, char \*\*argv)

{

setlocale(LC\_ALL, "Russian");

return wxEntry(argc, argv);

}

bool Application::OnInit()

{

if (!wxApp::OnInit())

{

return false;

}

init();

return true;

}

Frame::Frame(const wxString& title)

: wxFrame(NULL, wxID\_ANY, title)

{

SetIcon(wxICON(sample));

wxMenu \*fileMenu = new wxMenu;

//fileMenu->Append(File\_Size, "&Size", "Resize screen");

fileMenu->Append(FILE\_QUIT, "E&xit\tAlt-X", "Quit this program");

wxMenu \*toolsMenu = new wxMenu;

toolsMenu->Append(TOOL\_SCPI, "SCPI");

wxMenu \*helpMenu = new wxMenu;

helpMenu->Append(HELP\_ABOUT, "&About\tF1", "Show about dialog");

wxMenuBar \*menuBar = new wxMenuBar();

menuBar->Append(fileMenu, "&File");

menuBar->Append(toolsMenu, "Инструменты");

menuBar->Append(helpMenu, "&Help");

SetMenuBar(menuBar);

CreateStatusBar(2);

SetStatusText("Welcome to wxWidgets!");

Bind(wxEVT\_MENU, &Frame::OnSCPI, this, TOOL\_SCPI);

Bind(wxEVT\_MENU, &Frame::OnQuit, this, FILE\_QUIT);

Bind(wxEVT\_MENU, &Frame::OnSize, this, FILE\_SIZE);

Bind(wxEVT\_MENU, &Frame::OnAbout, this, HELP\_ABOUT);

Bind(wxEVT\_CLOSE\_WINDOW, &Frame::OnClose, this);

timer.Bind(wxEVT\_TIMER, &Frame::OnTimer, this);

timer.Start(0);

timerLongPress.Bind(wxEVT\_TIMER, &Frame::OnTimerLong, this);

self = this;

ConsoleSCPI::Self()->Show();

}

Frame \*Frame::Self()

{

return self;

}

void Frame::OnTimer(wxTimerEvent&)

{

update();

HandlerEvents();

DrawFPS();

}

void Frame::DrawFPS()

{

static int count = 0;

static unsigned int prevTime = 0;

count++;

if (TIME\_MS - prevTime > 1000)

{

float fps = (float)count / (float)(TIME\_MS - prevTime) \* 1000.0F;

char buffer[100];

sprintf(buffer, "fps %f", fps);

SetStatusText(buffer);

prevTime = TIME\_MS;

count = 0;

}

}

void Frame::OnSize(wxCommandEvent&)

{

}

void Frame::OnQuit(wxCommandEvent& WXUNUSED(event))

{

Close(true);

}

void Frame::OnClose(wxCloseEvent &)

{

ConsoleSCPI::Self()->Destroy();

timer.Stop();

Destroy();

}

void Frame::OnAbout(wxCommandEvent& WXUNUSED(event))

{

wxMessageBox(wxString::Format

(

"Welcome to %s!\n"

"\n"

"This is the minimal wxWidgets sample\n"

"running under %s.",

wxVERSION\_STRING,

wxGetOsDescription()

),

"About wxWidgets minimal sample",

wxOK | wxICON\_INFORMATION,

this);

}

void Frame::OnSCPI(wxCommandEvent &)

{

ConsoleSCPI::Self()->SwitchVisibility();

}

enum

{

FILE\_SIZE = wxID\_HIGHEST + 1,

FILE\_QUIT = wxID\_EXIT,

TOOL\_SCPI,

HELP\_ABOUT = wxID\_ABOUT

};

wxIMPLEMENT\_APP\_NO\_MAIN(Application);

Frame \*Frame::self = nullptr;

int main(int argc, char \*\*argv)

{

setlocale(LC\_ALL, "Russian");

return wxEntry(argc, argv);

}

bool Application::OnInit()

{

if (!wxApp::OnInit())

{

return false;

}

init();

return true;

}

Frame::Frame(const wxString& title)

: wxFrame(NULL, wxID\_ANY, title)

{

SetIcon(wxICON(sample));

wxMenu \*fileMenu = new wxMenu;

//fileMenu->Append(File\_Size, "&Size", "Resize screen");

fileMenu->Append(FILE\_QUIT, "E&xit\tAlt-X", "Quit this program");

wxMenu \*toolsMenu = new wxMenu;

toolsMenu->Append(TOOL\_SCPI, "SCPI");

wxMenu \*helpMenu = new wxMenu;

helpMenu->Append(HELP\_ABOUT, "&About\tF1", "Show about dialog");

wxMenuBar \*menuBar = new wxMenuBar();

menuBar->Append(fileMenu, "&File");

menuBar->Append(toolsMenu, "Инструменты");

menuBar->Append(helpMenu, "&Help");

SetMenuBar(menuBar);

CreateStatusBar(2);

SetStatusText("Welcome to wxWidgets!");

Bind(wxEVT\_MENU, &Frame::OnSCPI, this, TOOL\_SCPI);

Bind(wxEVT\_MENU, &Frame::OnQuit, this, FILE\_QUIT);

Bind(wxEVT\_MENU, &Frame::OnSize, this, FILE\_SIZE);

Bind(wxEVT\_MENU, &Frame::OnAbout, this, HELP\_ABOUT);

Bind(wxEVT\_CLOSE\_WINDOW, &Frame::OnClose, this);

timer.Bind(wxEVT\_TIMER, &Frame::OnTimer, this);

timer.Start(0);

timerLongPress.Bind(wxEVT\_TIMER, &Frame::OnTimerLong, this);

self = this;

ConsoleSCPI::Self()->Show();

}

Frame \*Frame::Self()

{

return self;

}

void Frame::OnTimer(wxTimerEvent&)

{

update();

HandlerEvents();

DrawFPS();

}

void Frame::DrawFPS()

{

static int count = 0;

static unsigned int prevTime = 0;

count++;

if (TIME\_MS - prevTime > 1000)

{

float fps = (float)count / (float)(TIME\_MS - prevTime) \* 1000.0F;

char buffer[100];

sprintf(buffer, "fps %f", fps);

SetStatusText(buffer);

prevTime = TIME\_MS;

count = 0;

}

}

void Frame::OnSize(wxCommandEvent&)

{

}

void Frame::OnQuit(wxCommandEvent& WXUNUSED(event))

{

Close(true);

}

void Frame::OnClose(wxCloseEvent &)

{

ConsoleSCPI::Self()->Destroy();

timer.Stop();

Destroy();

}

void Frame::OnAbout(wxCommandEvent& WXUNUSED(event))

{

wxMessageBox(wxString::Format

(

"Welcome to %s!\n"

"\n"

"This is the minimal wxWidgets sample\n"

"running under %s.",

wxVERSION\_STRING,

wxGetOsDescription()

),

"About wxWidgets minimal sample",

wxOK | wxICON\_INFORMATION,

this);

}

void Frame::OnSCPI(wxCommandEvent &)

{

ConsoleSCPI::Self()->SwitchVisibility();

}

enum

{

ID\_LINE

};

static wxTextCtrl \*text = nullptr;

static wxTextCtrl \*line = nullptr;

static ConsoleSCPI \*self = nullptr;

pString ConsoleSCPI::DIRECT\_PROMT = ">>> ";

pString ConsoleSCPI::REVERSE\_PROMT = "<<< ";

ConsoleSCPI::ConsoleSCPI(wxFrame \*parent) : wxFrame(parent, wxID\_ANY, wxT("SCPI"))

{

text = new wxTextCtrl(this, wxID\_ANY, wxEmptyString, wxDefaultPosition, { 600, 300 }, wxTE\_MULTILINE | wxTE\_READONLY);

line = new wxTextCtrl(this, ID\_LINE, wxEmptyString, wxDefaultPosition, wxDefaultSize, wxTE\_PROCESS\_ENTER);

line->SetFocus();

wxFont fnt(11, wxFONTFAMILY\_DEFAULT, wxFONTSTYLE\_NORMAL, wxFONTWEIGHT\_BOLD, false, wxT("Courier New"));

line->SetFont(fnt);

text->SetFont(fnt);

Bind(wxEVT\_SIZE, &ConsoleSCPI::OnSize, this);

line->Bind(wxEVT\_TEXT\_ENTER, &ConsoleSCPI::OnTextEnter, this, ID\_LINE);

line->Bind(wxEVT\_KEY\_DOWN, &ConsoleSCPI::OnTextControlKeyDown, this, ID\_LINE);

Bind(wxEVT\_CLOSE\_WINDOW, &ConsoleSCPI::OnClose, this);

Show();

if (ComPort::Open())

{

AddLine("Обнаружено внешнее устройство");

timerComPort.Bind(wxEVT\_TIMER, &ConsoleSCPI::OnTimerComPort, this);

timerComPort.Start(10);

}

else

{

AddLine("Внешнее устройство не обнаружено. Работает эмулятор");

}

timerTest.Bind(wxEVT\_TIMER, &ConsoleSCPI::OnTimerTest, this);

}

ConsoleSCPI::~ConsoleSCPI()

{

ComPort::Close();

}

void ConsoleSCPI::OnSize(wxSizeEvent &)

{

wxPoint clientOrigin = GetClientAreaOrigin();

wxSize clientSize = GetClientSize();

int heightLine = line->GetSize().y;

wxSize sizeText = clientSize;

sizeText.y -= heightLine;

text->SetSize(sizeText);

line->SetPosition({ clientOrigin.x, clientSize.y - heightLine });

line->SetSize({ text->GetSize().x, heightLine });

}

static int Consist0D(char \*buffer, int size)

{

int i = 0;

while (\*buffer != 0x0d)

{

i++;

buffer++;

if (i == size)

{

return -1;

}

}

return i;

}

void ConsoleSCPI::OnTimerComPort(wxTimerEvent &)

{

if (ComPort::IsOpened())

{

static char fullBuffer[4096 \* 16] = { 0 }; // Полный текст

char buffer[4096];

int n = ComPort::Receive(buffer, 4095);

if (n)

{

int positionOD = Consist0D(buffer, n);

if (positionOD < 0)

{

buffer[n] = '\0';

std::strcat(fullBuffer, buffer);

}

else

{

String message(">>> %s", fullBuffer);

AddText(message.c\_str());

buffer[positionOD] = 0;

AddText(buffer);

fullBuffer[0] = 0;

}

}

}

}

ConsoleSCPI \*ConsoleSCPI::Self()

{

if (!self)

{

self = new ConsoleSCPI(nullptr);

}

return self;

}

void ConsoleSCPI::OnTextEnter(wxCommandEvent &)

{

history.Add(line->GetLineText(0));

AddText(DIRECT\_PROMT);

AddLine(line->GetLineText(0));

SendToSCPI(line->GetLineText(0).c\_str());

line->Clear();

}

void ConsoleSCPI::OnTimerTest(wxTimerEvent &)

{

const char \*names[] =

{

"NONE",

"FUNCTION",

"MEASURE",

"MEMORY",

"SERVICE",

"1",

"2",

"TIME",

"START",

"TRIG",

"DISPLAY",

"RANGE1+",

"RANGE1-",

"RSHIFT1+",

"RSHIFT1-",

"RANGE2+",

"RANGE2-",

"RSHIFT2+",

"RSHIFT2-",

"TBASE+",

"TBASE-",

"TSHIFT+",

"TSHIFT-",

"TRIGLEV+",

"TRIGLEV-",

"LEFT",

"RIGHT",

"UP",

"DOWN",

"ENTER",

"F1",

"F2",

"F3",

"F4",

"F5"

};

String message(":key:press %s", names[(rand() % 34) + 1]);

SendToSCPI(message.c\_str());

}

void ConsoleSCPI::SendToSCPI(const char \*txt)

{

if (ComPort::IsOpened())

{

String message("%s\x0d", txt);

ComPort::Send(message.c\_str());

}

else

{

}

}

void ConsoleSCPI::StartTest()

{

AddLine("Тест стартовал");

srand((uint)time(0));

timerTest.Start(500);

}

void ConsoleSCPI::StopTest()

{

AddLine("Тест завершён");

timerTest.Stop();

}

void ConsoleSCPI::OnTextControlKeyDown(wxKeyEvent &event)

{

if (event.GetKeyCode() == WXK\_UP)

{

wxString txt = history.Prev();

if (txt.size())

{

line->Clear();

line->WriteText(txt);

}

}

else if (event.GetKeyCode() == WXK\_DOWN)

{

wxString txt = history.Next();

if (txt.size())

{

line->Clear();

line->WriteText(txt);

}

}

else

{

event.Skip();

}

}

void ConsoleSCPI::AddLine(const wxString &str)

{

AddText(str);

AddText(wxT("\n"));

}

void ConsoleSCPI::AddText(const wxString &str)

{

text->WriteText(str);

}

void ConsoleSCPI::SwitchVisibility()

{

Self()->Show(!Self()->IsShown());

}

void ConsoleSCPI::OnClose(wxCloseEvent &)

{

Self()->Show(false);

}

void ConsoleSCPI::History::Add(const wxString &txt)

{

if ((history.size() == 0) ||

(history[history.size() - 1].compare(txt) != 0))

{

history.push\_back(txt);

position = history.size() - 1;

}

}

wxString ConsoleSCPI::History::Next()

{

if (history.size() == 0)

{

return "";

}

wxString result = history[position];

position++;

if (position == history.size())

{

position = 0;

}

return result;

}

wxString ConsoleSCPI::History::Prev()

{

if (history.size() == 0)

{

return "";

}

wxString result = history[position];

position = (position == 0) ? (history.size() - 1) : (position - 1);

return result;

}

// Очередь сообщений - здесь все события органов управления

#define MAX\_ACTIONS 100

static Control actions[MAX\_ACTIONS];

// Количество уже имеющихся сообщений

static int numActions = 0;

static bool needStartTimerLong = false;

static bool needStopTimerLong = false;

// Здесь имя нажатой кнопки

static Control::E pressedKey = Control::None;

bool Keyboard::Init()

{

return true;

}

static void AddAction(Control control, Control::Action::E action)

{

if (action != Control::Action::Press)

{

return;

}

control.action = action;

actions[numActions++] = control;

}

void Keyboard::AppendControl(const Control &control)

{

AddAction(control, control.action.value);

}

void Frame::OnDown(wxCommandEvent &event)

{

Control::E key = (Control::E)event.GetId();

//std::cout << "down " << Control(key).Name() << std::endl;

event.Skip();

AddAction(key, Control::Action::Press);

needStartTimerLong = true;

pressedKey = key;

}

void Frame::OnUp(wxCommandEvent &event)

{

Control::E key = (Control::E)event.GetId();

//std::cout << "up " << Control(key).Name() << std::endl;

event.Skip();

AddAction(key, Control::Action::Release);

needStopTimerLong = true;

pressedKey = Control::None;

}

bool Keyboard::Empty()

{

return numActions == 0;

}

Control Keyboard::NextControl()

{

if (Empty())

{

return Control();

}

Control result = actions[0];

for (int i = 1; i < numActions; i++)

{

actions[i - 1] = actions[i];

}

--numActions;

return result;

}

String Control::Name() const

{

static const char \*names[Control::E::Count] =

{

/\* 1 \*/ "Ручка нажать",

/\* 2 \*/ "РЕЖИМ",

/\* 3 \*/ "ИНДИКАЦИЯ",

/\* 4 \*/ "<-",

/\* 5 \*/ "->",

/\* 6 \*/ "КАНАЛЫ",

/\* 7 \*/ "ENTER",

/\* 8 \*/ "СЕРВИС",

/\* 9 \*/ "Ручка лево",

/\* 10 \*/ "Ручка право",

/\* 11 \*/ "ТЕСТ",

/\* 12 \*/ "АВТО",

/\* 13 \*/ "NULL"

};

return String(names[value]);

}

void Keyboard::Lock()

{

Menu::Input::SetFuncUpdate(Menu::Input::FuncEmptyUpdate);

}

void Keyboard::Unlock()

{

Menu::Input::SetFuncUpdate(Menu::Input::FuncUpdate);

}

class GovernorGUI : public wxPanel

{

public:

// code - код ручки

GovernorGUI(wxWindow \*parent, const wxPoint &position);

private:

static const int radius = 32;

static const float stepDegree;

void OnPaint(wxPaintEvent &);

void OnMouseLeftDown(wxMouseEvent &);

void OnMouseLeftUp(wxMouseEvent &);

void OnMouseMove(wxMouseEvent &);

void OnTimer(wxTimerEvent &);

// Возвращает true, если курсор мыши находится над изображением ручки

bool MouseOnGovernor(wxMouseEvent &);

// Синус от градусов

float Sin(float);

// Косинус от градусов

float Cos(float);

// Эта фунция вызывается при переключении ручки в следующую/предыдущую позицию

void FuncRotate(int delta);

// Эта функция вызываетс при нажатии ручки

void FuncPress();

struct StructCursor

{

bool leftIsDown; // true, если левая кнопка нажата

POINT position; // Позиция курсора

int state; // Состояние VK\_LBUTTON

// Рассчитывает dX и dY между position и текущей позицией

int CalculateDelta();

// Возвращает true, если сейчас нажата левая кнопка мыши

bool LeftIsDown();

// Вызывается при нажатии левой кнопки мыши

void OnPressLeftButton();

} cursor;

float angleFull = 0.0F; // Реальный угол поворота ручки (без градаций переключения)

float angleDiscrete = 0.0F; // Отображаемый угол поворота ручки (у учётом градаций переключения)

wxTimer timer;

bool needEventPress = true; // Если true, то при отпускании мыши нужно посылать событие нажатия ручки

};

GovernorGUI::GovernorGUI(wxWindow \*parent, const wxPoint &position) : wxPanel(parent, wxID\_ANY, position), timer(this, 1)

{

angleDiscrete = ((float)(std::rand() % 100) - 100.0F) \* stepDegree;

cursor = { false, {0, 0}, 0 };

SetSize({ radius \* 2 + 1, radius \* 2 + 1 });

SetDoubleBuffered(true);

Bind(wxEVT\_PAINT, &GovernorGUI::OnPaint, this);

Bind(wxEVT\_LEFT\_DOWN, &GovernorGUI::OnMouseLeftDown, this);

Bind(wxEVT\_LEFT\_UP, &GovernorGUI::OnMouseLeftUp, this);

Bind(wxEVT\_MOTION, &GovernorGUI::OnMouseMove, this);

Bind(wxEVT\_TIMER, &GovernorGUI::OnTimer, this);

timer.Start(0);

}

void GovernorGUI::OnPaint(wxPaintEvent &)

{

wxPaintDC dc(this);

wxBrush brush({ 0, 0, 0 }, wxTRANSPARENT);

dc.SetBrush(brush);

dc.DrawCircle(radius, radius, radius);

float r = static\_cast<float>(radius) \* 0.6F;

float x = static\_cast<float>(radius) + Sin(angleDiscrete) \* r;

float y = static\_cast<float>(radius) + Cos(angleDiscrete) \* r;

dc.DrawCircle(static\_cast<int>(x), static\_cast<int>(y), radius / 5);

}

void GovernorGUI::OnMouseLeftDown(wxMouseEvent &event)

{

if(MouseOnGovernor(event))

{

needEventPress = true;

::SetCursor(LoadCursor(NULL, IDC\_HAND)); //-V2571

cursor.OnPressLeftButton();

}

}

void GovernorGUI::OnMouseLeftUp(wxMouseEvent &)

{

if (needEventPress)

{

FuncPress();

}

needEventPress = false;

}

void GovernorGUI::OnMouseMove(wxMouseEvent &event)

{

if(MouseOnGovernor(event))

{

::SetCursor(LoadCursor(NULL, IDC\_HAND)); //-V2571

}

}