Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 09:36:01 GMT
View Forum Message <> Reply to Message

i'll try to..

Quote:
whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

well, SDL is in fact not quite the same as simply framebuffer and /dev/input, its more sort of X11, you allocate a SDL_Surface, and need to process input events..

as far as got through that whole thing, SystemDraw is the implementation is draw backend and DoMouse & DispatchKey are already backends... to these i tried to connect.

so far i tried to sum up the whole WIN32 thing by runtime tracing in debugger: correct if i am wrong somewhere..

//////////////////////////////////////////////////////// ////
SYSTEM BACKEND WIN32

win32 api current state and important stations, analyzing and tracking the GUI_APP_MAIN macro.

Ctrl::InitWin32(HINSTANCE hInstance)
init/ register der window classes and a seperate timer class, instantiating of timer class with lpfnWndProc = &Ctrl::UtilityProc; the window classes with lpfnWndProc = (WNDPROC)Ctrl::WndProc;, they are the message processors. calls to create windows are done with win function CreateWindow later.

void AppInitEnvironment__()
language and environment specific inits like charsets

GuiMainFn()
applications specific stuff, global intis, here, Run() of a TopWindow class should be called

TopWindow::Run()
basicly forwards to create the window class instances in Open() call, Show()s it and sets up the EventLoop() to run, which only exits when app quits. prepares the ctrl exit.

void TopWindow::Open()
creates the window instance using the Create() subsystem for the main window instance

void Ctrl::Create0(Ctrl::CreateBox *cr)
actually creates and shows the window instance with win32 OS means, sets up the drag and drop stuff and calls a first RefreshDeepLayout(), which triggers a WM_PAINT, which is processed in Ctrl::WndProc

LRESULT CALLBACK Ctrl::WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
takes care of processing WM_CREATE und WM_DESTROY, otherwise forwards to the hwnd specific Ctrl (mapping exists), which is a TopWindow ctrl.. forwards to the WindowProc of the Ctrl, which is virtual

LRESULT TopWindow::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
takes care of some minimal TopWindow specific stuff like close behaviour, then, forwards to Ctrl::WindowProc

LRESULT Ctrl::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
processes the win32 message queue messages, here comes in WM_PAINT, which uses a SystemDraw to paint the main / top Ctrl's area to the win32 GDC surface of the hWnd, here also comes in WM_'mousestuff' which forwards to DoMouse, which takes care of translating and delivering the messages to the ctrls. further, WM_'keystuff' comes in, which forwards to DispatchKey..

void Ctrl::EventLoop0(Ctrl *ctrl)
is called in Run() (via EventLoop) and returns when application is about to finish, basicly calls in a loop ProcessEvents() which calls ProcessEvent() which calls sProcessMSG()

void Ctrl::sProcessMSG(MSG& msg)
runs the win32 API typical TranslateMessage / DispatchMessage duo, which ends up calling WndProc and dispatching / evaluating messages

LRESULT CALLBACK Ctrl::UtilityProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
processes the WM_TIMER message. the timer is used to wake up the ProcessEvents stuff (TranslateMessage / DispatchMessage) which will also process what?? why does it need to wakeup? process the postcallbacks? becuse, user input and repaint stuff comes directly as WM_* messages, which wake up the queue as well

sorry for the long post, just need to clarify i dont miss things.