S60 SDL

Markus Mertama - mertama@mbnet.fi

S60 SDL is a Simple DirectMedia Layer adaptation for S60. SDL is a cross-platform multimedia library: Applications and libraries built on SDL can easily be ported to another operating systems. SDL provides C (compatible with C++) interfaces to OS specific services, like drawing, audio, threads, timers etc. So basically application built on SDL can run on any system that has SDL and standard C libraries.

Usage of S60 SDL is not restricted to porting; S60 multimedia applications can be implemented without further knowledge of Symbian C++ native API and developer can use SDL and standard C interfaces. The SDL development supports both Nokia OpenC and Symbian stdlib (ESTLIB) C libraries.

With S60 SDL it is possible to port applications from other systems to S60 without a single code change. However in practice S60 devices usually has a small screen and limited input possibilities, and S60 SDL has a special S60 API that makes integration to a mobile platform easier; CSDL interface helps to do adaptation without changes to the original code base.

The S60 SDL supports also OpenGL ES development. Its possible to create SDL Surface for OpenGL ES content and manage that with SDL's platform independent OpenGL API. In S60 devices that does nott have hardware accelerated OpenGL, a software rendered is used automatically.

Getting started

SDL development in ten steps (more detailed in following pages):

1. Download and install Symbian SDK from Forum Nokia.
2. Download SDL from www.libsdl.org, extract it into SDK folder.
3. Download S60 SDL, extract it onto SDK folder.
4. Run install.bat
5. Write MMP, PKG, RSS and BLD.INF files (see below)
6. Import BLD.INF into Carbide
7. Write code
8. Compile
9. Create SISX packect from binaries.

10. Release

If only a latest sdl library is needed, a SDL S60 installation **sisx** packet is required (see S60 SDL page for latest release).

 The SDL sources are available at Simple DirectMedia Layer homepage. The archive contains common SDL sources and platform spesific sources are in their own archives inside. Download and extract SDL archive to root of development drive. (Its assumed that S60 SDK is installed into its own drive, typically substed folder) Then get latest S60 SDL sources, extract zip file which creates *S60_SDL_install* folder, goto there and run install.bat. The installation patches very latest S60 SDL version on official SDL release.

In SDL folder, there is symbian sub-folder. Run configure.bat it compiles and installs libraries and test applications and also builds installation packages into symbian/install folder.

For manually recompiling: The Symbian bld.inf and MMP files are located in symbian folder. The SDL test applications can be used by running *buildtest.bat* (e.g. *buildtest symbianc* to use ESTLIB) in its folder. It generates S60 spesific files for SDL test applications, builds them and then generates a **sisx** file to be installed. SDL test applications are examples how changes to original code are not needed at all.

Developing to S60 without code changes

When developing or porting applications to S60, a developer has have at least two files: build tools uses bld.inf and MMP files. If target binary is an application an additional Application Shell registration file that is needed to get its icon visible in Application Shell. For further information about those files, please see S60 3rd edition SDK documentation.

MMP file

The binary built is defined in MMP file. S60 SDL applications has to have some SDL libraries defined. There are three libraries that can be used:

- ✎ **Sdl.lib** has a common S60 SDL implementation as well as a CSDL interface. This is needed always when using S60 SDL.

- ✎ **Sdlexe.lib** contains implements S60 application framework; it makes a ported SDL application to be a native S60 application. This is good to use when implementing an application.

- ✎ **Sdlmain.lib** has an application entrypoint. (The Symbian OS has an entrypoint called E32Main – not a C/C++ standard main). When this version is used SDL main is run in application thread. If application is using special CSDL parameters, easiest way is to leave this entry libray away – and write own entry point function. See Entrypoint example.

Only a sdl.lib is needed if target if a library or application has its own S60 application framework implementation. Sdlexe.lib implements a minimal S60 application, however application needs an entry point from statically linked sdlmain.lib library. Application may also has its own entry point – then it is able to spesify flags for SDL and get access to CSDL instance by using a SDLMain interface. If application has its own entry point, linking to sdlmain.lib is not used.

example

```
TARGET testbitmap.exe
TARGETTYPE exe
UID 0 0xe0001005
```

```
CAPABILITY ReadUserData WriteUserData
EPOCHEAPSIZE 1000000 20000000

USERINCLUDE ..\include
SYSTEMINCLUDE \epoc32\include \epoc32\include\libc \SDL-1.2.11\include
SOURCEPATH ..\src
SOURCE testbitmap.c

SOURCEPATH .
START RESOURCE testbitmap_reg.rss
TARGETPATH \private\10003a3f\apps
END

LIBRARY euser.lib
LIBRARY estlib.lib
LIBRARY sdl.lib
LIBRARY sdlexe.lib

STATICLIBRARY sdlmain.lib
```

The S60 resouce in MMP file, e.g "testbitmap_reg.rss", is to get application visible in application shell.

When implementing real application additional libraries may be needed, e.g. libgles_cm.lib for OpenGL ES.

example

```
#include <appinfo.rh>

UID2 KUidAppRegistrationResourceFile
UID3 0xe0001005

RESOURCE APP_REGISTRATION_INFO
     {
     app_file = "testbitmap";
     }
```

so after those two Symbian and S60 spesific file you are ready to concentrate on SDL and dont have to worry more about Symbian development enviroment! Untill when you want to install your binary into phone and need a installation packet – but there should be plenty of documentation bundled with S60 C++ SDK that clarifies the issue.

Basically now you can just call your "main" and go ahead :-)

```
int main(int argc, char** argv)
     {
     Uint32 videoflags = SDL_SWSURFACE | SDL_ANYFORMAT | SDL_FULLSCREEN;
     int width = 640;
     int height = 480;
     int bpp = 32; // full color
     int i;
     SDL_Rect **modes;
     SDL_Surface* screen;
     SDL_Surface* bmp;
     SDL_Event event;
     int done = 0;
     SDL_Rect sourceRect;
     SDL_Rect targetRect;
```

```
if(SDL_Init(SDL_INIT_VIDEO) < 0 )
        {
        panic(NULL, "init video");
        }

modes = SDL_ListModes(NULL, SDL_FULLSCREEN);

if(modes == NULL)
        {
        panic(NULL, "No available video modes");
        }

if(modes != (SDL_Rect**) -1)
        {
        width = modes[0]->w;
        height =  modes[0]->h;
        }


screen = SDL_SetVideoMode(width, height, bpp, videoflags);

bmp = SDL_LoadBMP("hello_world.bmp");

sourceRect.x = 0;
sourceRect.y = 0;
sourceRect.width = bmp->w;
sourceRect.height = bmp->h;

targetRect.x = (screen->w – bmp->w) / 2;
targetRect.y = (screen->h – bmp->h) / 2;
targetRect.width = bmp->w;
targetRect.height = bmp->h;

SDL_BlitSurface(bmp, &sourceRect, screen, &targetRect);

while(!done)
        {
        while(SDL_WaitEvent(&event))
                {
                switch(event.type)
                        {
                        case SDL_KEYDOWN:
                        case SDL_QUIT:
                                done = 1;
                                break;
                        }
                }
        }

SDL_FreeSurface(bmp);
SDL_FreeSurface(screen);

SDL_Quit();
return 0;
}
```

Note: If you dont want to have a parameter query dialog, put a file "sdl_param.txt" in application "home" folder (c:\Private\<APPLICATION UID>). If your application does not use any parameter just let "sdl_param.txt"be empty. (If you use !:\ syntax in pkg file – note that parameter file has always to be in C: drive). See SDLEnv::SetMain.

Flags

You can manage Symbian SDL adaptation by using flags file – sdl_param.txt and sdl_flags.txt. sdl_param.txt has same format as command line dialog. SDL internal flags are separated using '|' characters. E.g. commandline can be "app_argument |SDL_ARGUMENT|", that is convenient at development phase for fast testing different flag effects. The flags can also be applied using sdl_flags.txt file.

Currently supported flags are:

```
"EnableFocusStop" – see  CSDL::EEnableFocusStop
"DrawModeDSB" – see  CSDL::EDrawModeDSB
"DrawModeGdi" – see  CSDL::EDrawModeGdi
"AutoOrientation"– see  CSDL::EAutoOrientation
"DisableVolumeKeys" – see  CSDL::EDisableVolumeKeys

"ParamQuery" – see  SDLEnv::EParamQuery
"AllowConsoleView" – see  SDLEnv::EAllowConsoleView
"VirtualMouse" – see  SDLEnv::EVirtualMouse}
"ParamQueryDialog" – see  SDLEnv::EParamQueryDialog
"FastZoomBlitter" – see  SDLEnv::EFastZoomBlitter
"EnableVirtualMouseMoveEvents"– see  SDLEnv::EEnableVirtualMouseMoveEvents
"HWABlitter" – see  SDLEnv::EHWABlitter
"HWABlitterNoScale" – see  SDLEnv::EHWABlitterNoScale
"HWABlitterRatioScale" – see  SDLEnv::EHWABlitterRatioScale
```

SDLMain interface

Defined in sdlmain.h

Library sdlexe.lib

SDLMain library implements a S60 application framework, runs a SDL application in its own thread and let SDL draw to application window. When application starts it creates a thread for SDL and then wait SDL thread to exit. When SDL thread is exit, also application is closed. The SDLMain pass Window Server events to SDL thread and allows to emulate a mouse pointer in non-touch enabled devices.

SDLEnv

Sdlexe.lib exports only one function. Application entry point, E32Main calls that function to engage S60 application. If a ported application requires some adaptation to S60 a **SDLEnv::SetMain** function can be called to pass flags or set a **MSDLMainObs** observer. The SDLEnv::SetMain is called instead of normally used **EikStart::RunApplication** function.

static TInt SetMain(const TMainFunc& aFunc, TInt aSdlFlags, MSDLMainObs* aObs = NULL, TInt aEnvFlags = EflagsNone)	
Creates a SDL thread and S60 application.	
aFunc	Pointer to main function, a **SDL_main** macro is defined in sdlepocapi.h.

aSDLFlags	Flags are defined in *sdlepocapi.h*, see **CSDL**.
aObs	Sets a **MSDLMainObs**

aEnvFlags	Sets enviroment flags.The possible values are:	
	EFlagsNone	No special flags.
	EViewParamQuery	The parameters are red from "sdl_param.txt" that should be located in application "home" folder, i.e. Its own private folder. (<INSTALLATION DRIVE>:\Private\<APPLICATION UID>). If such file is not found a parameter query dialog is viewed and the user entered string is passed to main function as standard C argc/argv form.
	EAllowConsoleView	If application uses console output, i.e. printf function, the text is put in its own window. However that window is not allowed to hide graphical window unless this flag is set.
	EVirtualMouse	The mouse pointer can be enabled by pressing a 'Yes' (a green) phone button. The pointer is moved using cursor keys and 'tapped' (left mouse key event generated) by using a middle key (rocker).
	EParamQueryDialog	The parameter dialog is always shown. If EViewParamQuery flag is set, paramers given in dialog are concatenated to paramters parsed from "sdl_param.txt" file.
	EFastZoomBlitter	Use faster blitter when zoomin up frames, CSDL::EMainThread and CSDL::EAllowImageResize should be set. (Not checked however, if some one writes more flexible blitter). Normally Symbian BitGDI slow DrawImage is used when zooming is requested for drawing. This flag should provide has faster alternative. (oh – just my own algorithm I quickly wrote, no magic :-)
	EEnableVirtualMouseMoveEvents	Emulates also mouse move (or pen move), when middle key (rocker) is pressed first time a pen is down and arrow changes to yellow. When second press is done arrow changes back to white and pen is up.
	EHWABlitter	Use HWA blitter - N8 onwards, requires non backwards compatible SDL upgrade. Is a bit faster than a normal SW blitter.
	EHWABlitterNoScale	As EHWABlitter, but does not do any scaling if

		requested surface does not fit in window.
	`EHWABlitterRatioScale`	As `EHWABlitter`, but preserves ratio and do centrifying while doing scaling inside given window.
Return value	A value to be returned from E32Main – see S60 SDK documentation.	

example

```
GLREF_C TInt E32Main()
    {
    return SDLEnv::SetMain(SDL_main, CSDL::EEnableFocusStop |
CSDL::EAllowImageResize, NULL, SDLEnv::EViewParamQuery | SDLEnv::EVirtualMouse);
    }
```

The example is sdlmain.lib implementation. The implementation can change flags and set an observer. You don't need this when sdlmain.lib is statically linked.

MSDLMainObs

Pure virtual class to receive SDL application change information and get access to CSDL instance.

TInt SDLMainEvent(TInt anEvent, TInt aParam, CSDL* aSdl)		
Receives SDLMain events.		
`anEvent`	Spesifies an event.	
	`EError`	An error has occured.
	`ESDLCreated`	SDL library is created, but main function is not yet called.
`aParam`	An effect spesific parameter. If anEvent is:	
	`EError`	An error code. See S60 SDK documentation about standard error codes.
	`ESDLCreated`	Not used.
`aSDL`	A pointer to **CSDL** instance. Can be NULL if instance creation is failed or not yet completed.	
Return value	Not used.	

CSDL interface

Defined in sdlepocapi.h

Library sdl.lib

CSDL interface is a low-level API to control SDL adaptation on S60. As Symbian and SDL has unique charasteristics as well as devices applications running are intented to be mobile. Therefore there are cases that Sdlexe and *CSdlEnv* interfaces do not provides good enough control and more low-level approach is needed. In *CSDL::SetMain* function its possible to provide rich set of flags to CSDL, but controlling SDL will let control e.g. Keycodes, thread handling, passing Window server events and blitting.

S60 SDL has several modes of drawing and each of those has their advantages. BitGDI provides very compatible drawing interface, DSA (Direct Screen Access) is marginally faster – actually so marginally that normally applications spends most of the time somewhere that DSA wont provide real advantage over BitGDI. However DSA is default mode. DSB (Direct Screen Bitmap) is faster than DSA – but some of its submodes wont work in all hardware and may be other compatibility issues as well.

SDL can run in application main thread or in its own thread. Normal applications work happily in main thread, the drawing etc. utilizes Symbian Active Object Framework and therefore there are certain limitations: SDL applications should call event request funtions or UI upgrade frequently to let AOs run. If application uses busy loops that wont call SDL functions, that application should run in its own SDL thread. Application thread is a default as it generally works well.

 The S60 SDL application has a main control window for SDL video surface. S60 devices have different screen sizes, and typically those are smaller than SDL applications are designed for, e.g. 640x480. To have a minimum porting effort, S60 SDL adaptation may resize its drawing to screen size, however that affects dramatically to frame rate. By default SDL uses Symbian graphics services for resizing, but resizing can be overridden with an user algorithm to gain a better performance.

SDL library has a internal keymapping based on PC QWERTY keys. CSDL has an interface to remap keys for limited input devices.

CSDL

Before SDL functions are used a CSDL instance has to be created. CSDL implements a S60 adaptation for SDL functions and has a set of utility functions to integrate SDL binary on S60 enviroment.

`static CSDL* NewL(TInt aFlags = CSDL::ENoFlags)`		
Create a CSDL instance.		
`aFlags`	Flags for SDL:	
	`ENoFlags`	No special flags.
	`EEnableFocusStop`	Stops the SDL thread when an application loses focus. The SDL thread is resumed when focus is gained.
	`EDrawModeDSB`	Not supported
	`EAllowImageResize`	Allow display to be resized if given window and SDL application requested surface size does not match. If not set, mismatching SDL video creation request results an error.

	`EDrawModeDSBDoubleBuffer`	Not supported
	`EDrawModeDSBIncrementalUpdate`	Not supported
	`EAllowImageResizeKeepRatio`	Display is resized, but aspect ratio is maintained.
	`EDrawModeGdi`	Use BITGDI drawing interface.
	`EDrawModeDSBAsync`	Not supported
	`EOwnThread`	Deprecated. SDL main funtion is running its own thread, not in S60 applicatation thread. **Important**: OpenGL ES does not work when this flag is used.
	`EMainThread`	Always run SDL mainfuntion in application thread. Default.
	`EImageResizeZoomOut`	Zooms automatically if screen surface is smaller that given window.
	`EAutoOrientation`	Set orientation automatically to match dimensions SDL screen to match better window size, affects only if TAppOrientation is EDefault
Return value	Address of SDL instance.	

void SetObserver(MSDLObserver* aObserver)	
Set current observer.	
`aObserver`	Pointer to observer. If NULL an observer is removed.

MSDLObserver* Observer()	
Get a current observer.	
Return value	Pointer to observer. If NULL there is no observer.

void SetContainerWindowL(RWindow& aWindow, RWsSession& aSession,

CWsScreenDevice& aDevice)
Set window for SDL video surface. The function must be called before creating of surface and should be recalled (propably in application HandleResourceChange) if screen device or its size changes. However SDL application has to handle changed window buffer size.

aWindow	Drawing window.
aSession	Used window server session.
aDevice	Used window device.

void DisableKeyBlocking(CAknAppUi& aAppUi)
Let application to receive multiple keypresses simultaneously.

aAppUi	Application UI class

~CSDL()
Destructor, should be called when SDL is not used anymore. SDL application thread must be terminated first.

TInt AppendWsEvent(const TWsEvent& aEvent)
Sends TWsEvents to SDL event queue. Typically events are passed from application HandleWsEventL function.

aEvent	A passed event.
Return value	S60 error code, if KErrOverflow, a buffer is full (size is 64 events) – other codes, see S60 SDK.

TInt GetSDLCode(TInt aScanCode)
Get SDL keycode mapped to given scan code. There is a default mapping close to common PC QWERTY codes. Function shall not be called before EEventKeyMapInit event is received.

aScanCode	A Symbian scan code, see S60 SDK. documentation about system scan codes.
Return value	SDL key code, see SDL documentation.

TInt SetSDLCode(TInt aScanCode, TInt aSDLCode)
Maps system scan code to SDL code. Shall not be called before EEventKeyMapInit event is received.

aScanCode	A S60 system scan code, see S60 SDK.
aSDLCode	A SDL code, see SDL documentation.
Return value	A Previous SDL key code, see SDL documentation.

TInt SDLCodesCount() const

Return number of SDL keycodes.

Return value	Number of SDL keycodes.

---

void ResetSDLCodes()
Removes all SDL – S60 scancode mappings to their defaults.

---

void SetOrientation(TOrientationMode aMode)		
Set a display orientation. If a view aspect ratio changes, SDL implementation should take care of that.		
aMode	EOrientation0	Normal view
	EOrientation90	Turn view 90°
	EOrientation180	Turn view 180°
	EOrientation270	Turn view 270°

---

void Resume()
Resumes a suspended SDL.

---

void Suspend()
Suspends a SDL.

TThreadId CallMainL(TRequestStatus& aStatus, const CDesC8Array& iArg, TParamFlags iFlags = CSDL::ENoParamFlags, TInt aStackSize = KDefaultStackSize)		
Initializes a SDL thread and calls its main function.		
aStatus	Request status object that expires when thread terminates.	
iArg	String array passed to SDL application in common C, C++ argc/argv format.	
aFlags	ENoFlags	No special flags
	ERequestResume	SDL thread is not started automatically. If EMainThread is explicitly set, then ERequestResume has no effect.
aStackSize	Used stack size in SDL thread, not that Symbian OS limits value maximum. In S60 3.0 edition value shall not exceed 81920 bytes.	
Return value	Thread identidier of created SDL thread.	

TThreadId CallMainL(TRequestStatus& aStatus, TParamFlags aFlags = CSDL::ENoParamFlags, TInt aStackSize = KDefaultStackSize)

```
TThreadId CallMainL(TParamFlags aFlags = CSDL::ENoParamFlags, TInt
aStackSize = KDefaultStackSize)
```

```
TThreadId CallMainL(const CDesC8Array& iArg, TParamFlags Flags =
CSDL::ENoParamFlags, TInt aStackSize = KDefaultStackSize)
```

void SDLPanic(const TDesC& aInfo, TInt aErr)	
Panics a SDL thread.	
aInfo	Displayed info string.
aErr	An Error code.

TInt SetBlitter(MBlitter* aBlitter)	
Set a user blitter function. Useful when realtime resizing is needed and a default implementation is not fast enough. DSB modes are not supported.	
aBlitter	Pointer to class that implements a blitter function. If NULL a default blitter is used.
Return value	S60 error code.

TInt AppendOverlay(MOverlay& aOverlay, TInt aPriority)	
Let SDL adaptation software to drawn on SDL window. DSB modes are not supported.	
aOverlay	Class that implements an overlay drawing.
aPriority	Set priority of overlay. Higher priority overlays are drawn on lower priority overlays. Maximum priority is 0.
Return value	S60 error code.

TInt RemoveOverlay(MOverlay& aOverlay)	
Removes a given overlay. DSB modes are not supported.	
Return value	S60 error code.

TInt RedrawRequest()	
Redraws the latest frame – can be asynchrounus.	
Return value	S60 error code.

void SetAppOrientation(CAknAppUi& aAppUi, TAppOrientation aAppOrientation)	
Set application orientation – unlike SetOrientation that just changes drawing direction. SetAppOrientation use system support for orientation change. Overrides EautoOrientation flag.	
aAppUi	ApplicationUi that contains current SDL application.
aAppOrientation	Orientation order.

TInt Extension_(TUint aExtensionId, TAny*& a0, TAny* a1);	
Maintaing SDL library binary compatability is demanding task – but still important as applications written on earlier version should be still runnable if SDL library itself is upgraded. Extenssion_ funtion provides a way to extend CSDL functinality safely. New functions can be added without hassle with freezing and ordinal numbers. E.g. If two developers upgrades and freezes a new functins in SDL simultaneously those changes cannot be merged wihtout BC break as ordinals will collide.	
aExtensionId	Identfier of function.
a0	Reference pointer to generic data
a1	Pointer to generic data
Return value	Genereric return value

void ChangeBgColor(TRgb& aColor);	
Change and retrieve background color. Color that used to wipe screen. If SDL screen surface does not cover all screen area this color is used to drawn background.	
TRgb	In the new color. Out an old color.

void SuspendRequestsLC(TInt aRunFirst = CActive::*EPriorityIdle*);	
Utility function to catch stray signals at Symbian side code. E.g. if native dialogs are called from SDL code that may initiate a stray signal. Calling this function prevents SDL signalling as long as subsequent CleanupStack::PopAndDestroy is called.	
TInt aRunFirst	Let select level where SDL signals are prevented. The default restrains all.

MSDLObserver

Observer to get information from SDL state. Certain functions can be called only after SDL is fully

initiated. The MSDLObserver has callbacks to both main and SDL threads since some functions can be called only in either thread.

TInt SdlEvent(TInt aEvent, TInt aParam)		
A SDL event callback called in Application context.		
aEvent	EEventWindowReserved	The SDL video surface is created to given window and nothing else should draw on window.
	EEventWindowNotAvailable	SDL surface is not yet available, usually can be ignored.
	EEventScreenSizeChanged	Screen size is changed, SDL_SetVideoMode should to be called.
	EEventSuspend	SDL thread will be stopped next.
	EEventResume	SDL thread resumed. Not used in thread context.
	EEventKeyMapInit	Keymap is inited and key can be remapped.
	EEventMainExit	SDL main has been exited, but thread context is still alive.
aParam	Not used.	
Return value	For some events may apply a special return value	
	EParameterNone	Default value to return.
	EScreenSizeChangedDefaultPalette	After EEventScreenSizeChanged let palette change to SDL default.
	ESuspendNoSuspend	After EEventSuspend, prevents the suspending.

TInt SdlThreadEvent(TInt aEvent, TInt aParam)
A SDL event callback called in SDL Thread context.

MOverlay

Overlays can be used to add some SDL independent content on SDL screen as mouse cursor, battery indicator or debugging information. Moverlay drawing are not resized and coordinates may differ from SDL video surface coordinates. (In case of mouse cursor S60 SDL event adaptation does pointer coordinate change automatically.)

void Draw(CBitmapContext& aGc, const TRect& aTargetRect, const TSize& aSize)
Let S60 adaptation draw on SDL window. The callback is called for each frame after SDL has updated its content. Note that function is called from SDL thread context and it must be ensured

that all used handles all valid in this context.

aGc	A bitmap context to draw.
aTargetRect	A display rectangle, set when **CSDL::SetContainerWindowL** was called.
aSize	The size of SDL video surface.

MBlitter

MBlitter implements an user defined blitter function. If resize of SDL surface is done before copying content in to system display a slow linear DDA is used. It works well with any screen sizes but slow perfomance by magnitude comparing to normal blitting. The application may use MBlitter for use more efficient algorithm.

TBool BitBlt(CBitmapContext& aGc, CFbsBitmap& aBmp, const TRect& aTargetRect, const TSize& aSize)	
User defined blitter function implementation. Note that function is called from SDL thread context and it must be ensured that all used handles all valid in this context.	
aGc	A bitmap context to use for drawing.
aBmp	A SDL video surface as a bitmap that should be drawn on given screen.
aTargetRect	A display rectangle, set when **CSDL::SetContainerWindowL** was called.
aSize	The actual size of the SDL video surface. If rectangle size differs from this size, aBmp should be resized to rectanngle size when blitting.
Return value	If EFalse, a default blitter will be used for this frame, otherwise it is assumed that this function does blitting.

GCCE Problems

The S60 SDK's have an old version of GCCE compiler that has certain problems with global data. Please see discussion at Symbian Developer Forum.

OpenGL ES example

Famous "Hello World" of OpenGL implemented using OpenGL ES and SDL.

```
#include <SDL_opengl.h>
#include <stdio.h>
#include <SDL.h>
```

```
static const GLbyte vertices[3 * 3] =
    {
    -1,   1,    0,
    1,    -1,   0,
    1,    1,    0
    };

static const GLubyte colors[3 * 4] =
    {
    255, 0,            0,            255,
    0,           255,  255,  255,
    0,           0,            255,  255
    };

int panic(char* str)
    {
    fprintf(stderr,"%s: %s\n",str, SDL_GetError());
    exit( 1 );
    return 0;
    }

void renderFrame()
    {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -5.f);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    }

int main(int argc, char** argv)
    {
    int width;
    int height;
    SDL_Surface* screen;
    SDL_Rect** modes;
    int videoFlags = SDL_OPENGL | SDL_FULLSCREEN;
    int done = 0;
    SDL_Event event;

    argc;
    argv;

    if( SDL_Init( SDL_INIT_VIDEO ) < 0 )
            {
            panic("SDL Init");
            }

    modes = SDL_ListModes(NULL, videoFlags);

    if(modes == NULL)
            {
            panic("No available video modes");
            }

    width = 240;
    height = 320;

    if(modes != (SDL_Rect**) -1)
            {
            width = modes[0]->w;
```

```
            height = modes[0]->h;
            }

    SDL_GL_LoadLibrary(NULL);

    screen = SDL_SetVideoMode(width, height, 16, videoFlags);

    if(screen == NULL)
            {
            panic("No GLES surface available");
            }

    glDisable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glClearColor(0.f, 0.f, 0.1f, 1.f);

    glVertexPointer(3, GL_BYTE, 0, vertices);
    glColorPointer(4, GL_UNSIGNED_BYTE, 0, colors);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glFrustumf(-1.f, 1.f, -1.f, 1.f, 3.f, 1000.f);
    glMatrixMode(GL_MODELVIEW);

    while(!done)
            {
            if(SDL_PollEvent(&event))
                    {
                    switch(event.type)
                            {
                            case SDL_KEYDOWN:
                            case SDL_QUIT:
                            done = 1;
                            break;
                            }
                    }
            else
                    {
                    GLenum gl_error;
                    renderFrame();
                    SDL_GL_SwapBuffers();
                    /* Check for error conditions. */
                    gl_error = glGetError();
                    if(gl_error != GL_NO_ERROR )
                            {
                            fprintf(stderr, "OpenGL error: %d\n", gl_error );
                            SDL_FreeSurface(screen);
                            SDL_Quit();
                            exit( 1 );
                            }
                    SDL_Delay(20);
                    }
            }

    SDL_FreeSurface(screen);
    /* Shutdown all subsystems */
    SDL_Quit();
    return 0;
    }
```