

## Übersicht über die Funktionen dieser Druckformatvorlage:

### Absatzformate:

- Ctrl-Shift-N: Normal
- Ctrl-Shift-2: Eingerückt
- Ctrl-Shift-H: Große Überschrift
- Ctrl-Shift-K: Kleine Überschrift
- Ctrl-Shift-C: Courier (Proportionalschrift)

### Makros für Fußnoten:

- Ctrl-Shift-R: Raute
- Ctrl-Shift-P: Plus
- Ctrl-Shift-D: Dollar
- Ctrl-Shift-A: K (gesprochen: kAAAh)
- Ctrl-Shift-X: ! (engl. Exclamation Mark)

### Makros für Zeichenformate:

# **LGD API**

## **The Screen Saver Module Programmers Interface (SSMPI)**

Supported Programming Languages

Saver Modules: General Information

Function ScreenSaverID

Function ScreenSaverOptions

Function ScreenSaver

Function ScreenSaverAbout

Saver Windows Classes: LgdDefProc

Using 256 Colours with LGD

## Supported Programming Languages

The saver modules are Windows DLLs. Therefore you may use any programming language that is capable of creating DLLs and importing functions from other DLLs.

I include a sample saver for Borland Pascal 7.0 which should work with Turbo Pascal for Windows 1.5 as well (early versions did a few months ago). This sample demonstrates using 256 colours.

A sample for (Borland) C is included, but it's just a simple Blackness screen-saver.

Whatever language you choose, you should take a look at the Pascal sample 'Crawler'. Experienced C and/or Windows programmers should have no problems with reading this sample.

## Saver Modules: General Information

When started, LGD searches for files matching SS\_\*.LGD (in the directory where LGD.EXE resides). (Tip (especially for Turbo Pascal): edit **WEEP.INI** and insert **DEBUG=1** in the **[LGD]** section. LGD will then search for SS\_\*.DLL - so that you don't have to rename the DLLs created by Turbo Pascal. C programmers can write a make file which will automatically create \*.LGD files.)

For every module found the ScreenSaverID function is called. This function gives LGD name and description and additional information, i.e.

- if the saver features an Info dialogue (About box),
- if it features an Options dialogue,
- if the screen mustn't be black when the saver starts (e.g. Cast, Melting Ice, &c.),
- if the saver leaves the screen black (like Blackness or Fall Out).

Names and descriptions are saved in the LGD.CFG file. LGD updates this information only when the list in LGD.CFG doesn't match the savers found in the directory (if you change the name or description of a saver you wrote just delete LGD.CFG to have LGD update its information; LGD.CFG is updated automatically if you create a new saver).

The ScreenSaver function actually starts the screen-saver. One parameter is the period of time the saver shall be active. It is up to you if your saver finishes earlier (like Fall Out, when the screen is blank) or later (like Cast or Apple, when Time Out is not active) than specified.

The ScreenSaver function normally does the same as WinMain in a Windows application. The Crawler sample is written to create either an .EXE file (which simplifies debugging) or a .DLL file.

# Function ScreenSaverID

The saver DLL has to export this function as index 17.

The ScreenSaverID function extracted from the Crawler sample:

```
Procedure ScreenSaverID (var wMagic:integer;
                        var fFunctions:LongInt;
                        achName:pchar;
                        cchName:integer;
                        achDesc:pchar;
                        cchDesc:integer);

export;
begin
    wMagic := $6874;
```

Requires exactly this "magic" word for identification.

```
fFunctions := 4+2+1; { 1: about, 2:options, 3:both }
                { 4: non-blank (screen must not be black) }
                { 8: leaves screen blank }
```

fFunctions contains a combination of bit flags describing the saver.

```
StrLCopy (achName, 'CCrawler', cchName - 1);
{ the first character is not displayed
  its for sort ordering only. }
```

achName returns the name of the saver. The first character is not displayed, but is used for sorting the savers and should be the same as the first letter of the visible name.

Exceptions are Turbo Arty (which should remain the first saver) and Blackness (which should remain at the end).

cchName specifies how many characters may be copied.

Don't forget the zero byte at the end!

```
StrLCopy (achDesc, 'Crawler:'#10#10'Screen saver with support for 256
colours.'#10#10+
                'Sources (Pascal) included!'#10,
                cchDesc - 1);
```

achDesc returns the description for the saver. The line feed character (#10, 0x0a) results in line breaks. cchDesc contains the length of the buffer.

Please note: the text must fit into LGD's window (about 10 lines with about 30 characters each).

```
end;
```

End of the function

# Function ScreenSaverOptions

The saver DLL has to export this function as index 18.

This function will be called only if the corresponding flag is set.

This function receives the handle of the calling program's window. Use this window as parent of your dialogue window.

The scheme of this function:

- Read the settings from the INI-File
- Show the options dialogue
- Write (changed) settings to the INI-File

Following is an extraction from the Crawler sample (the function ReadProfile and WriteProfile can be seen in the sample source):

:

```
Procedure ScreenSaverOptions (window: hWND);  
export;  
var Proc: TFarProc;  
begin  
    ReadProfile;
```

**Read settings from WEEP.INI.**

```
    Proc := MakeProcInstance(@Options, HInstance);  
    DialogBox(HInstance, 'OPTIONBOX', Window, Proc);
```

**Show the dialogue (using resources in the saver module DLL).**

```
    FreeProcInstance(Proc);  
    WriteProfile;
```

**Write changes to WEEP.INI.**

```
end;
```

# Function ScreenSaver

The saver DLL has to export this function as index 19.

This function actually does what the WinMain function of other Windows programs does. Crawler was written to be compiled as an .EXE-File or a .DLL-LGD-module (depending on conditional defines).

The .EXE-version is convenient for debugging because it can be used without LGD.

All my saver modules follow a simple scheme:

- Read settings from the .INI-File
- Register a window class
- Create a window
- Wait until the window is closed (or time expired) and draw something using WM\_TIMER messages or PeekMessage function calls

The DLL **LM\_UTIL.DLL** exports the LgdDefProc function. The window function of your window class has to call this function first!

Messages not handled by LgdDefProc can be handled by your window function (which should call DefWindowProc for any message it doesn't handle).

A screen-saver doesn't behave like a normal Windows program. It fills the whole screen and it is terminated if any key is pressed.

Therefore my screen-savers do \*NOT\* support WM\_PAINT messages!

The return value of ScreenSaver is very important. It indicates if the saver module terminated "normally" (because the time slice expired) or if the user terminated it.

Randomizer requires this information to know if it should run another saver module.

These return values are currently used:

- 0: Time slice expired (Randomizer will run another saver module)
- 1: Terminated by user (Randomizer will terminate as well)
- 2: Saver overlapped by other windows (\*)

(\*) Saver modules return 2 if terminated by a WM\_KILLFOCUS message.

The first parameter to this function is the time slice (in seconds) specifying how long the saver module should run.

The second parameter is a pointer to an interface structure if Windows is running with 256 or more colours. This structure is defined in SSCOMMON.PAS. For further information see Using 256 Colours with LGD.

## Function ScreenSaverAbout

The saver DLL has to export this function as index 20.

This function resembles ScreenSaverOptions. Its parameter is a window handle. Use this function to display your copyright notice.

```
Procedure ScreenSaverAbout (window: hWnd);  
export;  
begin  
    messagebox (window, 'Saver module for ''The Lights Go Down''#10'(C)  
        1993 Leo Minor', 'Crawler', mb_Ok or mb_ApplModal);  
end;
```



# Saver Window Classes

## Using LgdDefProc

The LgdDefProc function responds to some window messages. Your window function should call LgdDefProc first. If LgdDefProc returns TRUE, your window function should return the value it received from LgdDefProc.

If LgdDefProc returns FALSE, you may handle the message in your window function or call DefWindowProc.

This concept proved to be useful. For version 2.0 I implemented the password protection. I only had to change LGD.EXE and LM\_UTIL.DLL (LgdDefProc) - existing saver modules didn't require any modifications!

Prototype (C Notation):

```
BOOL LgdDefProc (LPLONG lRet,
                 HWND hWnd,
                 WORD msg,
                 WORD wParam,
                 LONG lParam);
```

Pascal Notation:

```
Function LgdDefProc (var lRet:LongInt;
                    window: hwnd;
                    msg:word;
                    wParam:word;
                    lParam:LongInt): boolean;
```

Turbo Pascal programmers may use the LM\_LGD import unit, while C programmers need an IMPORT entry in their .DEF file and a prototype in any .H file.

Calling LgdDefProc in Pascal:

```
if (LgdDefProc (lRet, window, Message, wParam, lParam)) then
begin
    WindowProc := lRet;
    exit;
end;
```

C programmers would write:

```
if (LgdDefProc (&lRet, window, Message, wParam, lParam))
    return lRet;
```

Messages your saver module has to respond to:

```
wm_KillFocus,
wm_KeyDown,      { any key will stop the screen saver }
wm_Close,
wm_lButtonDown,
wm_mButtonDown,
wm_rButtonDown
```

All the messages listed above terminate the screen-saver.

Respond to the WM\_QUERYNEWPALETTE and WM\_PALETTECHANGED messages if your saver supports 256 Colours.

## Using 256 Colours

If Windows is running a 256 colours mode, only one colour palette is available for all applications. Even though only one LGD module can be running at any time, the output of more than one screen saver may be visible at the same time. To avoid problems, all saver modules have to use the colour palette offered by LGD (if they want to use 256 colours).

### What does LGD's palette look like?

The primary colours Red, Green, and Blue may have values in the range from 0 through 5. The palette index for such a colour is calculated as

$$\text{INDEX} = 10 + \text{RED} * 36 + \text{GREEN} * 6 + \text{BLUE}$$

This colour scheme uses 216 (6 to the power of 3) entries of the palette. For seven colours (Red, Green, Yellow, Magenta, Cyan, White (Grey)) more shades were defined giving 8 for any of these colours. If you want to use these colours (as Apple does), then use the following structure (defined in SSSCOMMON.PAS):

```
ColorIndexesColorScales :array[0..55] of byte =
  (11, 12, 235, 13, 236, 14, 237, 15,  { blue }
   16, 22, 229, 28, 230, 34, 231, 40,  { green }
   17, 24, 241, 31, 242, 38, 243, 45,  { cyan }
   46, 82, 226,118, 227,154, 228,190,  { red }
   47, 84, 238,121, 239,158, 240,195,  { magenta }
   52, 94, 232,136, 233,178, 234,220,  { yellow }
   10, 53,  96,244, 139,245, 182,225 );{ white }
```

Your program may work with True Colour internally, using values ranging from 0 through 255 for the primary colours Red, Green, and Blue. Use PALETTE\_RGB to find the colour that matches best.

### How to use the palette in your saver module

When the ScreenSaver function is called, you may receive a pointer to an interface structure containing the colour palette. The Crawler sample illustrates how to access this structure.

Your window has to respond to WM\_QUERYNEWPALETTE and WM\_PALETTECHANGED messages. If you haven't worked with 256 colours yet, you should take a close look at the Crawler sample.

Crawler uses a little hat trick when creating its window: the window is sized 1x1 when opened for the first time. Later, before drawing, it gets 'zoomed'. If no colour palette is used or if Crawler is called by Randomizer (it does the same trick) then the window is maximised right from the start.

Why do that?

If the colour palette changes, all visible windows that use colour palettes of their own receive messages so that they can re-paint their windows using the new palette. Crawler uses a maximised 'see through' window - The other windows are still visible for you, because Crawler doesn't erase its background, but they are not visible for Windows and thus are not re-painted.

By creating a 1x1 window they still are visible and thus have a chance to re-paint using the LGD palette.

