ScreenSaver Expert for Delphi 2.0

Contacting the Author

ScreenSaver Expert for Delphi is a copyrighted work of

Pat Ritchey
70007.4660@compuserve.com

To report bugs, you can send EMail to the above address, but be warned that at this time the ScreenSaver Expert project is something that I'm doing in my spare time, so you may get no reply concerning a bug report.

Installation Notes

Welcome to the first beta version of Delphi ScreenSaver Expert.

This is a Delphi 2.0 product ONLY.

Until all reported bugs are fixed, the release of this expert will include the DLL and DCU files only. Source will be considered for release when the product is solid.

ScreenSaver Expert consists of the following files:

SSEXPERT.DLL	the expert DLL that assists in the creation of a Delphi based screen saver.
SCRNSAVE.DCU	a Delphi unit that contains most of the functionality of Microsoft's SCRNSAVE.LIB.
SSINIT.DCU	a small unit that is used by SCRNSAVE.DCU. Due to the way that SCRNSAVE.DCU works, there is code that must be in the initialization section of a separate unit.
SSEXPERT.HLP	This file.
SETUP.EXE	A setup utility that copies the above files to the appropriate directories and sets various registry entries as required by Delphi.

If you haven't already done so, unzip all of the files in SSEXPERT.ZIP into a temporary directory and then run SETUP.EXE.

Note that Delphi 2.0 must not be running when you run the ScreenSaver Expert SETUP utility.

Within SETUP.EXE, you will be prompted for directories in which to place the DLL and the two DCU files. By default, both directories will be a subdirectory named "SSExpert" that is below your Delphi 2.0 base directory.

After selecting the directories, press the Install button. The files will be copied to the appropriate directories, the registry updated, and the files in the temporary directory will be removed.

At this point, you should be able to start Delphi and use the ScreenSaver Expert.

Using the Expert

To use the expert, start Delphi 2.0 and then select the "File|New..." menu entry. From the dialog presented, select the "Projects" page. At this point there should be an entry for the Screen Saver Expert. If there isn't, something failed during installation.

Double-Click on the ScreenSaver Icon. This should cause the ScreenSaver expert dialog to appear.

On the first page of the expert dialog, you'll be asked to enter a path in which to store the project that you're creating, and the name of the project. The name of the project will become the DPR file name, so it must be a valid Pascal identifier. Don't forget that with the new file open dialogs you can right click and create a new directory (folder). I highly recommend that each project be placed in it's own directory.

After entering the project path and project name, click on the "Next" button, you'll then be presented with the second (and final) page of the ScreenSaver Expert. There are two checkboxes.

The first checkbox, "Generate Idle event Handler" allows you to conditionally include code in the screen saver main form to support the "Application.OnIdle" event. When checked, all an OnIdle event handler will be created so that all that you need to do is add your screen painting code within that event handler.

The second checkbox, "Generate Configuration Dialog" will generate a simple dialog form (containing an OK and Cancel button) that you can further customize with your own configuration settings. If you'll need a configuration dialog, it's recommended that you let the expert create it for you. The expert will add the configuration dialog to the project's custom main source file (.DPR file) correctly.

After checking the appropriate checkboxes, click on the "Generate" button (or the "Back" button if you wish to update the entries on the first page). When "Generate" is clicked the expert will create the needed source files. These files will always be created:

<project-name>.dpr
Main.pas
Main.dfm

If you checked the "Generate Configuration Dialog" checkbox, these files will also be created:

Config.pas
Config.dfm

At this point, you may want to review the contents of the generated .DPR file. It is going to look a bit different than the typical Delphi .DPR file:

```
program DelphiSaver;

//
//  Note: ScrnSave MUST be the first unit in the uses list
//

uses
  ScrnSave,
  Forms,
  Main in 'Main.pas' {SaverForm},
  Config in 'Config.pas' {ConfigDlg};

{$R *.RES}

begin
```

```
  Application.Title := 'Random Dots';
  SaverFormClass := TSaverForm;
  ConfigDlgClass := ConfigDlg;
  RunScreenSaver;
end.
```

You'll see that there are no calls to Application.CreateForm, and there is no call to Application.Run. This is as designed. The functionality of those Application methods is contained within the SCRNSAVE unit procedure RunScreenSaver. If at any time you modify your project causing Delphi to insert either an Application.CreateForm or Application.Run statement, you MUST remove them. The screen saver will not function correctly unless the outer block of the project source looks like the one above.

Know Bugs and limitations

1. In the SETUP utility, the Browse buttons won't work under Windows NT 3.5x. This is a know limitation in that the WinNT 4.0/Win95 Shell Browser interface is used.

2. Actually, I don't know what if anything will work under WinNT 3.5x. I've tested all of this under Windows 95 (Service Pack 1) only.

The SCRNSAVE unit

The SCRNSAVE unit (SCRNSAVE.DCU) implements all of the messy details involved with writing a 32-bit screen saver. It is a close approximation to Microsoft's SCRNSAVE.LIB library included with the WIN32 SDK.

The unit defines a base screen save form class that is called **TSSBaseForm**. It's not important to know the methods that **TSSBaseForm** overrides, it's only important to know that your screen saver form must be a descendant of **TSSBaseForm** (rather than **TForm**) for everything to work.

In addition to the TSSBaseForm class, SCRNSAVE.DCU also defines five global variables and a single procedure. The run-time values of these global variables and the call to the single procedure are established by the screen saver expert, so the typical screen saver author rarely needs to interact with these global items.

The single interfaced procedure is **RunScreenSaver**. A call to this procedure is placed in the screen saver's DPR file by the screen saver expert, and should never be referenced elsewhere in the screen saver code.

The global variables are:

RunType: TRunType;

The **RunType** variable can be inspected at run-time to determine why your screen saver was invoked. A screen saver can be invoked for one of four reasons: Preview, Configuration, Password changing, and of course to actually run the screen saver. These four reasons are reflected by the values of the TRunType enumeration:

```
type
  TRunType = (ssUnknown,ssShow,ssConfig,ssPreview,ssPassword);
```

MiniWindow:HWND;

The **MiniWindow** variable contains the value of the parent window when **RunType** is **ssConfig**, **ssPreview** or **ssPassword**. It's used internally by the **TSSBaseForm** class and is really only an informational item outside of the the **TSSBaseForm** code.

SaverFormClass : TFormClass;

The form class that is the screen saver form is assigned to **SaverFormClass** by the screen saver expert.

ConfigDlgClass : TFormClass;

The form class that is the screen saver configuration dialog is assigned to **ConfigDlgClass** by the screen saver expert.

PasswordCheckingModule:String;

By default, **PasswordCheckingModule** is set to 'password.cpl', and an entry point withing password.cpl is called to perform password checking when a user attempts to terminate a screen saver that is running with **RunType** = **ssShow**. Note that this password checking within the screen saver occurs in Windows 95 only. In Windows NT, password checking is performed by the operating system. Normally this variable is informational only, but if custom password checking is desired you

can create your own DLL and assign the name of that DLL to **PasswordCheckingModule**. See Creating a Password Checking DLL for more information

Creating a Password Checking DLL

To create a password checking DLL that overrides the built-in functionality of the Windows 95 password.cpl library one needs to write a DLL that has a single entry point defined as (in Pascal):

```
function VerifyScreenSavePwd(Wnd:HWND):integer; stdcall;
```

or in C:

```
DWORD WINAPI VerifyScreenSavePwd(HWND Wnd)
```

and assign the name of the DLL to the PasswordCheckingModule variable defined by SCRNSAVE.DCU. This assignment would usually been performed in the DPR file prior to the invocation of the RunScreenSaver procedure.

When Windows 95 asks the screen saver to verify the password, code in SCRNSAVE.DCU will load the DLL dynamically (via **LoadLibrary**), the address of the **VerifyScreenSavePwd** entry point will be obtained (via **GetProcAddress**), and the entry point will be called. The code in **VerifyScreenSavePwd** should then do password verification (typically by obtaining the password from the user) and then return either zero or a non-zero value. A result of zero is an indication that the incorrect password was entered. A non-zero value indicates that the password was correctly entered.

A simple "hard-coded" password verifcation DLL might look something like:

```
Library PWC;

uses
  Messages,Windows;

{$R PWC.RES}
// Where PWC.RES contains a single dialog resource of the form:
// ------------------
// DIALOG_1 DIALOG 32, 59, 162, 64
// STYLE DS_SYSMODAL | DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
// CAPTION "Password Check"
// FONT 8, "MS Sans Serif"
// LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
// {
//  LTEXT "Enter Password:", -1, 6, 7, 60, 8
//  EDITTEXT 101, 7, 21, 141, 12, ES_PASSWORD | WS_BORDER | WS_TABSTOP
//  DEFPUSHBUTTON "OK", 1, 77, 43, 37, 14
//  PUSHBUTTON "Cancel", 2, 123, 43, 33, 14
// }

var
  UserInput:array[0..255] of char;

function DlgBoxProc(DlgWnd:HWND; Msg:Cardinal; wParam:Cardinal;
lParam:longint):BOOL; stdcall;
var
  CtlID:Word;
begin
  Result := false;
  CtlID := LOWORD(wParam);
```

```
  if Msg = WM_COMMAND then
    if CtlID <= 2 then
       begin
       if CtlID = 1 then
          GetDlgItemText(DlgWnd,101,UserInput,255);
       EndDialog(DlgWnd,CtlID)
       end;
end;

function VerifyScreenSavePwd(Wnd:HWND):integer; stdcall;
begin
   Result := 0;
   if DialogBox(hInstance,'DIALOG_1',Wnd,@DlgBoxProc) <> 1 then exit;
   if lstrcmpi(UserInput,'WORKTODO') = 0 then Result := 1;
end;

exports
  VerifyScreenSavePwd;

begin
end.
```