

Screen Dream

A GUI Suite for Delphi Programmers

Copyright 1995 by John Reynolds and Custom Windows Programming. All rights reserved. Unauthorized duplication and distribution prohibited and will be prosecuted to the fullest extent of law, national and international. Shareware version may be distributed only in its original, unmodified and provided file.

Inquiries and orders

U.S.

John Reynolds
327 62nd St NW

(505) 836-1384
(505) 264-0708
Jreynd@rt66.com

European only:

Rafe Aldridge
89 Sharland Close
Grove, Oxon. OX12 OAF

Tel: (01235) 769781
cwp@rcai.demon.co.uk

Screen Dream Coding Completely Optional
Copyright 1995 by John ReynoldsCustom Windows Programming
All rights reserved

Licensing Statement

For each copy of Screen Dream purchased, you may run it on not more than one machine at one time. We realize that you may need work in separate locations and on separate computers; this is acceptable, providing that you not have copies of one purchased package running or being used simultaneously.

By purchasing this package, you are given the right to use it to develop application programs. You are also given the right to make one diskette copy of this product for archival means. However, it may not be distributed to others for any purpose, including but not limited to demonstration. Anyone desiring free demonstration copies can contact Custom Windows Programming at (505) 836-1384/(505) 264-0708 or by email at jreynd@rt66.com.

This package was designed to assist you, the developer, in developing and writing application programs. This software and/or source code you have received may not be distributed, in part or in whole, to others for the purpose of using it to develop software applications. It can only be distributed as a part of standalone software applications.

We will prosecute infringements on this agreement to the fullest extent of the law.

If you have any questions about this statement, please contact John Reynolds at (505) 836-1384 / (505) 264-0708, or Internet E-Mail at jreynd@rt66.com.

The shareware version of Screen Dream may only be run for evaluation purposes. It may not be distributed as part of an application, but may be distributed in the original file, in its unmodified form.

Disclaimer:

John Reynolds, Custom Windows Programming, Distributors, BBSes and/or On-line Providers or any other party(ies), are not liable for any damages or problems caused by this program, regardless of damage, regardless of reason. Damage liability if proven and required in any case is limited to replacement of the price you paid for this package. Every effort was made to ensure that this program runs as advertised on your machine and your target machines. Future problems (if any) with this software will be dealt with and fixed to the ability of John Reynolds and Custom Windows Programming, until the release of the next version and beyond that at the discretion of Custom Windows Programming. Unregistered users have no claim in any case; without registration no technical support, patches, bugfixes or other services or material will be made available.

Installation Instructions

To install these files, the following must be done: *Note: The following instructions assume that you installed Delphi to the default \Delphi directory.*

1. Copy all files with the HLP extension to your Delphi\Bin directory (Di.Hlp, TTileBgn.Hlp).
2. Copy all files with the KWF extension to your Delphi\Help directory.
3. Copy all files with the DCU and DCR extension to your Delphi\Lib directory or make a new directory and ensure that in your Options|Environment you reference the location of those files. These are the library files.
4. To ensure that your on-line help works, you must add the indexes to Delphi's help index. These are the files with the KWF extension. Run HelpInst.Exe, from your Delphi Group.
5. Load the file Delphi.Hdx from your \Delphi\Bin directory into HelpInst, and add the files DI.KWF and TTileBgn.KWF (from the \Delphi\Help directory) to the group of files already present.
6. Save the file (the keywords are automatically added).
7. Copy all the files with the BMP extension to a directory you consider suitable. You might make a subdirectory called Tiles under the Delphi\Images directory for this purpose if you like. Remember this location, as you'll find yourself using these files often.
8. Load Delphi and add the files Devc_Ind.Dcu, CliDn.Dcu, and Bgn.Dcu to your library.
9. Everything should be ready to go. If you haven't done so already, read these instructions and the on-line help. There are a few pitfalls to avoid, and if you read the instructions you'll be more prepared to continue.
10. Start using Screen Dream!

Help files can also be perused by either creating an icon in the Windows Program Manager, by choosing File|Run from the Program Manager, or by doubleclicking on the files from the File Manager. In the respective help directory (probably \DELPHI\BIN), these files are named: DI.HLP, and TTILEBGN.HLP. DI covers the TDevice_Independence component and TTILEBGN covers the TTileBgnd and TClientMove components.

Overview

The Screen Dream package was designed to assist you in developing Multimedia Packages. It is the first in a series called Coding Completely Optional. This first package contains three visual controls. It also contains source code to allow powerful, flexible and easy creation of toolbars.

TDevice_Independence: The TDevice_Independence control allows you to create resolution-independent applications. You simply drop it on the form to create a full-screen application for any resolution. There are a few other options that are available to you by setting properties. These will be detailed a little later.

TClientMove: The TClientMove control lets you create caption-less forms that can still be moved by the user. Its sole purpose is to allow the user to click in the client area of the form and drag it to another location on the screen.

TTileBgnd: The TTileBgnd component lets you add tiled bitmap backgrounds to your forms. It gives you an easy way to add texture and style to your applications. It also allows you to put borders on these forms. The properties specific to this control will be detailed later. The TTileBgnd component doesn't work on MDI forms, but does work on MDI Child forms.

Toolbars: The creation of toolbars is easily done using the TClientMove and TTileBgnd components and a prebuilt example that can be saved as a new file and modified. This was done instead of creating a toolbar VCL for the added flexibility. A VCL component for toolbars may be developed in the future if sufficient demand exists.

TDevice_Independence Control

Properties

- OrigFormWidth: Integer. This is the width of your form when the application started. It's read-only at run-time. Default: None.
- OrigFormHeight: Integer. This is the height of your form when the application started. It's read-only at run-time. Default: None.
- FrmWidth: Integer. This is identical to the TForm's **Width** property. It is used by the control to provide a quick access to the value and to avoid querying the actual form during loops. Default: None.
- FrmHeight: Integer. This is identical to the TForm's **Height** property. It is used by the control to provide a quick access to the value and to avoid querying the actual form during loops. Default: None.
- ScreenWidth: Integer. This is the width of the screen at the current resolution. If the system were running in 640X480 mode, this value would be 640. Default: None.
- ScreenHeight: Integer. This is the height of the screen at the current resolution. If the system were running in 640X480 mode, this value would be 480. Default: None.
- MaxScreenSize_Percent: Integer. You set this property at design-time or run-time. It tells the control not to allow the width of the form to go beyond MaxScreenSize_Percent percent of the screen's width. If the height of the form is greater than the screen height, the form's width is reduced to a value that will allow the form's height to be ScreenHeight or less. If this property's value is zero (0), no adjustment to the height is made after the form is resized; it is simply scaled to the new size. Default: zero (0).

TDevice_Independence Control (Continued)

- MinScreenSize_Percent:** Integer. You can set this value at design-time or run-time. It tells the control not to allow the width of the form to go below `MinScreenSize_Percent` percent of the screen's width. If this property's value is zero (0), no adjustment to the height is made after the form is resized; it is simply scaled to the new size. Default: zero (0).
- FormInUpperLeft:** Boolean. If this is true and `AutoStretch` is true, the form is automatically positioned to the upper-left corner upon start. It is assumed that most users will want a full-screen form, and this is to prevent having to code this location each time. If this property is set to False, the form will be positioned at the design-time setting. Depending on your screen's resolution, you may need to position your forms properly at run-time. Default: True.
- AutoStretch:** Boolean. This property allows you to turn resolution dependence on and off. When `AutoStretch` is off, if the form is resized no changes to the form are made and no record is kept of the change. This can have ill side-effects. The main purpose of this property is to allow you to create forms that are not the exact shape of the screen. When the form is first activated, you can turn `AutoStretch` off, resize the form to the screen size and turn `AutoStretch` back on. This will ensure that no gaps exist between the side or bottom of the form and the screen border. Default : True.

Making a Form Resolution Independent:

To make a form resolution independent, simply drop the TDevice_Independence control on the form. Its default settings will, at run-time, make the form fill the screen and resize the controls to make the form virtually look the same at any resolution.

If you want a full-screen form, it must be the **exact shape** of the screen. If your form's size is 640X480, 800X600, 1024X768, for example, it is the exact shape of the screen. 320 X 240 is, also. The reason for this is that the control resizes the form based on the form's width, and then sizes the height to match. If the height is greater than the screen's height, the height is changed to the height of the screen and the width is returned to its **original percentage** of the form's height. Similar to a bitmap's aspect ratio -- the form can't be stretched out of shape.

There is a way around that. It requires a couple of lines of code. When the form first shows it is resized to the screen's width and height or as close as possible to that. You can then turn AutoStretch off by using the command:

```
TDevice_Independence1.AutoStretch := False;
```

Then set the width and height like so:

```
If MyForm.Width <> Screen.Width then  
    MyForm.Width := Screen.Width;
```

```
if MyForm.Height <> Screen.Height then  
    MyForm.Height := Screen.Height;
```

Then turn AutoStretch back on:

```
TDevice_Independence1.AutoStretch := True;
```

Be careful with the AutoStretch property, however. When you do this, the control is **turned off!** If you leave it off, resize the form and turn it back on, it will be scaled based on the latest size occupied by the form. Not the size of the form at design, but the last current size. This means that interesting things can happen on your form. Don't turn AutoResize off unless you have a special reason for doing so. One special reason is the one I stated previously. Turning it off to make a small change doesn't have ill effects, if you turn it back on immediately afterwards.

Avoiding Screen Flash

When a Delphi form is displayed for the first time, it shows itself before the

TDevice_Independence component on that form has had time to initialize the size. This can cause screen flash, especially if there is a profound difference in size between the design-mode form and the run-time form.

There is a way around this. The LockWindowUpdate function, called with the form's handle, prevents the form from being drawn until it is called again and released. Here is some sample code to perform this, with a form called Form1.

```
procedure TForm2.Button1Click(Sender: TObject);

{Flag indicating first time through routine}
Const BeenHere : Boolean = False;

{Flag to indicate if window has been locked previously}
Const WindowLocked : Boolean = False;
begin
    if not BeenHere then
    begin
        {Lock window from writes; update flag}
        BeenHere := True;
        LockWindowUpdate(Form1.handle);
        WindowLocked := True;

        {Show the form and hide it again}
        Form1.Show;
        Form1.Hide;

        {Let paint events be complied with}
        Application.ProcessMessages;
    end;

    {If Window is locked, unlock it}
    if WindowLocked then
    begin
        LockWindowUpdate(0);
        WindowLocked := False;
    end;
    CmSoon.Show;
end;
```

Be careful when using the LockWindowUpdate function. It can cause the TTileBgnd component's 'painting' to get 'washed off' under certain situations. If this happens, you can use this code to prevent it after the fact:

```
SendMessage(Form1.Handle, WM_Paint, 0, 0);
```

Note: Upgrades to the TDevice_Independence component have virtually made screen flash a thing of the past.

Making a Form Remain a Specified Percentage of the Screen

The `MinScreenSize_Percent` and `MaxScreenSize_Percent` properties allow you to keep a form within a required range of screen “real-estate.” To set these properties, guess at the percentage of Screen **width** the form is taking up and put that percentage in the appropriate property as a whole number between 1 and 100. If the number in `MaxScreenSize_Percent` is 30 and the number in `MinScreenSize_Percent` is 10, the form will not allow resizing by the user or the program beyond those ranges. Either property can be used with or without the other. If the number in either property is zero, it will be ignored.

It is fairly common to have scaling discrepancies when the form is scaled to an extremely small size and back, or to a size that is significantly smaller than the original form’s designed size. The scaling is done on the last width the form occupied. When scaled to a very small size, an integer is inadequate for differences and some precision is lost. When the form is then resized to a larger portion of the screen, this effect can sometimes be seen in the form of smaller or larger fonts. The effects of this can be minimized by making sure there’s room in labels and textboxes for larger or smaller font size. If you leave some room for error, it can be sized up and down indefinitely with no problems. You can also limit the minimum size to prevent scaling discrepancies. You can ignore this completely for the most part if the form will occupy the whole screen throughout the instance of your application.

Use TrueType fonts for device-independent applications. Non true-type fonts work, but scaling is not as smooth and there’s a good risk of blocky text. You may not have this option if your forms and captions need to be extremely small; TrueType doesn’t seem to look good on the screen at extremely small sizes. For toolbar captions I normally use the `SmallFont` fonts. It’s your discretion as to whether or not toolbars should be device-independent. I would say you’re better off leaving them as they were at design time.

Incompatibilities

Delphi's dialog controls don't seem to work too well with the TDevice_Independence control in some cases, especially when a tiled background is used. Take care when creating resolution independent forms with these controls on them. If it looks right on your program when running on your machine, chances are it'll work on other machines as well but test it thoroughly to make sure.

The TStringGrid and TDrawGrid components now work with the TDevice_Independence control. There are some behaviors that you should be aware of, however. When the form is resized, TDevice_Independence checks to see if the ScrollBars property is set to ssNone. If not, it re-scales the cells anyway but allows the width of the Grid control to increase beyond its borders. This is necessary to prevent the scrollbars from covering part of the control at all times when you didn't intend them to.

If the Scrollbars property *is* set to ssNone, the border automatically shrinks to fit the size of the control. It still exhibits a behavior which may be unwelcome; when the bottom row or rightmost column is selected it may show a border the width of the column/height of the row. This is a behavior of the TDrawGrid and TStringGrid controls and could not be easily altered under current timeframe restrictions.

Currently the TDevice_Independence component doesn't support MDI child forms completely. If you allow the form to resize, you should set an upper limit on it. It has a tendency to allow the form to grow more than the spacing between controls and the controls themselves. This allows the right and bottom form borders to steadily increase the distance from the controls as they are increased in size. You'll get better results if you leave a fair distance between borders and controls, especially the right and bottom form borders--to prevent the border from covering controls when the form's size is reduced by the user. Also with MDI Child forms (recommended with any other forms as well) AutoScroll should be False.

Do not use any other device independence product in the same application you're using this one for. Extremely strange results will occur, and you won't get exactly what you were expecting.

TClientMove Control

Properties

Enabled: Boolean. With this property enabled, the user can hold the left mouse button down on the form and drag the form to a new location. With it disabled, the control doesn't do anything.

Creating Toolbars

To create toolbars, the easiest thing to do is to load the files included on your installation diskettes (TBar2.Pas, Tool2.Pas), save them to a new file and modify them as needed. Your distribution diskettes should have been received with the write-protect tab missing. If not, you should take the time to knock them out. It would be too easy to overwrite the files and leave yourself unprotected from an error you didn't intend to cause. There are a couple of procedure calls made from the form's Tpanel component. These enable form movement by allowing the form to be dragged by the form's 'caption'. These procedures are a part of this form's source code and can be cut and pasted elsewhere if desired. Required form settings for making the border and actual form caption disappear are:

1. Select the form. Then you need to double-click on the + at the left of the word BorderIcons in the Object Inspector. This drops down a list of border icons. You then need to make all of these icons **False**.
2. The Form's Caption property must be empty. *Note: A rare behavior in form captions is to be visible after all this has been done. If this happens, most likely there's a space or two in the Caption property box. A guaranteed fix is to position your cursor in this box, hit CTRL-HOME, hold down the shift key and press END. Then press the DELETE key. That will solve the problem by removing the offending space(s). This is not a Delphi or Screen Dream problem, but simply a behavior of spaces: they're invisible.*
3. All non-client items (buttons, etc.) need to be children of a TPanel or other Container component, or have their own window handle. Otherwise, they need to be drawn on the panel. If not, the mouse message that would click these elements gets captured and used to move the form instead.
4. Artificial/Custom borders can be created using the TTileBgnd control (see TTileBgnd Control for more information).

TTileBgnd Control

Properties

- BorderOuterWidth:** Byte. This is the width of the outermost border. If set to zero, no outer border will be drawn on the form. Allowed range is 0 to 5. Default: 0.
- BorderInnerWidth:** Byte. This is the width of the border line drawn next to the outermost border. If set to zero, no inner border will be drawn on the form. Allowed range is 0 to 5. Default: 0.
- BorderInnermostWidth:** Byte. This is the width of the border closest to the inside of the form. If set to zero, no innermost border will be drawn on the form. Allowed range is 0 to 5. Default: 0.
- LowCornerOuterColor:** TColor. This is the color of the section of outer border on the right side and bottom. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.
- LowCornerInnerColor:** TColor. This is the color of the section of inner border on the right side and bottom. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.
- LowCornerInnermostColor:** TColor. This is the color of the section of innermost border on the right side and bottom. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.
- HighCornerOuterColor:** TColor. This is the color of the section of outer border on the left side and top. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.

TTileBgnd Control (Continued)

HighCornerInnerColor: TColor. This is the color of the section of inner border on the left side and top. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.

HighCornerInnermostColor: TColor. This is the color of the section of innermost border on the left side and top. This can be set at design time by double-clicking on the property. At run-time it can be set by assigning the value of a TColor variable to it.

Enabled: Boolean. This property when true, causes the TTileBgnd control to tile the background each time a paint message is received by the form. When false, it ignores paint messages.

Selecting a Bitmap

To select a bitmap, simply double-click on the control. A load dialog will pop up. Select a suitable bitmap for the background and click OK. The control will resize to the bitmap's size and everything is ready to go. The smaller the bitmap, the less run-time space and load time will be required but the longer redraws will take (measured in milliseconds).

For most cases it is important that your tiled bitmap looks like a full-sized bitmap on your runtime screen, with no seams. You can accomplish this to varying degrees of success using a good paint program. Recommended are texture fills and special-effects fills that don't vary a great deal in color selection. To get the edges to match, use an effect such as Seamless Welder in Corel Photopaint. That's how the sample bitmaps were created. There are many good bitmaps in the samples (and a few not so good). Try those out to get a feel for them.

Selecting a Bitmap (Continued)

Keep in mind also, that different systems have different color palettes. Running the Screen Dream demo is a good example; even at 256 colors, certain parts of the program look less than satisfactory. At 16 colors, forget it! I believe 256 colors is an acceptable standard. Ensure if you resample your tile images to 256 colors (from 24 or 32 bits), that the palette used is general enough to work on any 256 color system or you'll notice some strange effects. Having the image at 256 colors is no guarantee that it will look good on a system with a 256 color palette unless the image palette is compatible with the system palette. Most of the sample images are 16 color but some are 256 and even real-color.

Coding Concerns

When you are using a tiled background control, all controls on your forms must have their own clipping regions. Otherwise those controls won't show. Since the Tlabel control doesn't have its own clipping region, any Tlabel controls you use must be on a container control such as the Tpanel control. The Timage control must be placed on a Tpanel as well.

Incompatibilities

The TTileBgnd component does not work with MDI forms. It does, however, work with MDI child forms. It was not designed or tested for this use, however, but for using with standard non-mdi forms.

Technical Support

For questions about this product, problems or concerns, you can call (505) 836-1384 or (505) 264-0708 at any generally accepted working hour. If I'm not available, leave a message and I'll resolve it as soon as possible and get back to you. You can also leave me email at jreynd@rt66.com. This is probably a better alternative. When sending email, please leave me your phone number so I can call you if I need to clarify anything. *Unregistered users are not authorized technical support. If I have the time, I'll help you. I can't return your calls. Your best bet is to leave me e-mail.*

For bugs or incompatibilities in the library, I'll try to have them resolved within 48 to 60 hours and send you a fixed replacement via an on-line service such as the Internet or CompuServe. If you have found what you believe is a bug, please write down any information about the bug that could even laughably be called pertinent. Items such as operating system and version, processor, bios, time, date, video board and version, video driver and version, bus type, mouse, mouse driver, keyboard, and anything else you can think of. This will help me to solve the problem more quickly, and help you and my other customers in the process.

If you would like to know about future offerings, please send me email or mark the appropriate space on your registration sheet.. I won't send you any demos unless you ask. But I would like to send you mail asking if you want to receive a demo, bugfix or free upgrade if available. With your permission I'll do that. Without it, I won't.