This script was originally intended for experimentation and to demonstrate what a Presets dialog might look like, but perhaps it has become a prototype for Koda. Something like this could replace the Styles and exStyles tabs in Object Inspector.

Here's what the Help might say:

This dialog has several principal parts:

- Presets, which are combinations of styles which are known to work
- Styles groups, where Koda shows which styles are included in a Preset, and where you can choose your own styles
- Autoit parameters lists, that show which Windows constants will appear in generated code
- A list of Windows constants that AutoIt uses when it calls the Windows Create function, including those AutoIt forces
- Lists of the differences between what Windows constants AutoIt requests, and what Windows actually provided.
- A status indicator, which shows "Preset" and a number, "Custom", or "Failed".
- An optional picture of what the form will actually look like.

There are three ways of specifying styles for a form:

- Click on a Preset radio button: Presets are known to work; they cover most common requirements. The status indicator shows "Preset nn".
- Click on a Preset radio button, then check or uncheck styles: the status indicator usually shows "Custom", but may show "Failed".
- Click on the "None" radio button to uncheck all styles, then check styles as you wish: the status may be "Custom" or "Failed".

The style checkboxes you check are *requests* which Windows may fulfil (or may not): such are the ways of Windows!

Whether or not a form has a parent does affect the outcome for some styles. For example, "Simulate MDI" will cause a "Failed" status unless the form has a parent, as specified in the ParentForm property.

While the "AutoIt or Windows ignored" list is an indicator that your form may not have the styles that you request, the Test form feature shows exactly how your form will appear when the generated code is run. To turn this feature on, check "Show test form". Initially the test form will appear for 2 seconds. You can show it for longer by changing the value in the spinner control.

If you hover the mouse over a Preset radio button, styles it provides are displayed in a hint (tip).

If you hover the mouse over a style check box, the Windows constant it provides is displayed.

You will notice that one style, Close box, is always disabled. This is because there is no one Windows constant that provides a Close box. Several of the Presets do provide a Close box.

Be aware that the presence of a K icon in the Title bar does not in all cases mean that there is a System menu.

Styles that include `$WS_CAPTION` need to be explained. MSDN defines `$WS_CAPTION` as

BitOR($WS_BORDER, $WS_DLGFRAME). $WS_BORDER provides a thin-line border; $WS_DLGFRAME a double-line border. But Windows cannot provide both at the same time! It gives priority to $WS_DLGFRAME.

The AutoIt help states that calling GuiCreate() with a style parameter of -1 is equivalent to calling it with BitOr($WS_MINIMIZEBOX, $WS_CAPTION, $WS_POPUP, $WS_SYSMENU). This is Preset 1 in Koda. This Preset does provide what GuiCreate does with -1. But to reduce confusion, it checks Double-line border but not Thin-line border, even though Caption is checked (and Caption provides $WS_BORDER).

So if you modify Preset 1, you may get a surprise: checking "Thin-line border" does nothing! To explain, let's return to what a Preset is, and what checking style checkboxes does:

- When you choose a Preset, Koda shows you what styles you will get. (Preset 1 does not provide a thin-line border, $WS_BORDER.)
- By checking style checkboxes, you make requests; they may or may not be granted. (In this case, checking "Thin-line border" does nothing, because it was needed in the Preset to get a Caption, but unchecking "Double-line border" does change the appearance of the Test form.)

To make matters worse, Windows fails to report that $WS_BORDER was ignored, as you can see in the "AutoIt or Windows ignored" list.

"Layered effects" needs some help to work in your script. This dialog runs special code to display the Test Form when "Layered effects" is checked. After creating $testForm with the WS_EX_LAYERED extended style, Koda runs the Delphi equivalent of:

```
GUICtrlCreateLabel("Press Esc key to close", 8, 8, 164, 17,-1,$GUI_WS_EX_PARENTDRAG)
_WinAPI_SetLayeredWindowAttributes($testForm, 0x010101)
```

The Presets are based on Analysis of Form Styles.wb3 and .pdf. They can be refined, and more added. Perhaps in the future users would be able to add their own.

The Styles and ExStyles that are here are all that I have found in MSDN and that can be expected to work. $WS_EX_PALETTEWINDOW, a BitOR of WS_EX_WINDOWEDGE and WS_EX_TOPMOST, is omitted because it is not in an AutoIt include file.

If GuiCreate() works, the script compares:

- what the GuiCreate call to the Windows Create function requested (according to AutoIt help), and
- what the GetWindowLong{} function says it got.

The "Autoit or Windows ignored" and "AutoIt orWindows added" lists show the differences.

The script does not show the $GUI_SS_DEFAULT_GUI constant because I feel that leaving it out makes the results more easily comprehensible to the user: he doesn't need to refer to a help file to find out what it includes. Actually, the AutoIt help on GuiCreate does not mention it, nor does the help on Gui Styles state what it includes.

Some of the higher-numbered Presets should not be part of the Delphi code. They are demonstrations.

This script exceeds what GuiCreate() will do: the Test Form can have a Minimize Box and/or a

Maximize Box without a System Menu. I suggest that Koda consider doing the same. The script's code after `GuiSetState(@SW_SHOW)` is:

```
If BitAND($gParamStyles,$WS_SYSMENU)=0 Then
     Local $n = _C_GetWindowLong($testForm,$GWL_STYLE)
     $n = BitAND($n,BitNOT($WS_SYSMENU))
     _C_SetWindowLong($testForm,$GWL_STYLE,$n)
EndIf
```

where `_C_GetWindowLong` and `_C_SetWindowLong` are copies of the WinAPI functions.