Message Box 2.0

This is a replacement for the standard VDF message boxes: Info_Box, Stop_Box, YesNo_Box, and YesNoCancel_Box. In most if not all cases, the only change required should be to remove the underscore from existing code.

Background: The original replacement class was developed to address a few issues with the underlying Windows API used by the message boxes, specifically:

- Prior to Vista, Windows would respect the line breaks ("\n") that the programmer supplied for message boxes. Starting with Vista, it seems Microsoft decided there would be maximum width for message boxes, and started insert line breaks and kept the line breaks in the message box. The results did not look good.
- The Windows message box interface allows for a "Help" button to provide the user more information about the message, but using it requires callbacks that are not possible in VDF.
- A way was needed to add a checkbox to a message box (e.g. "Do not show this message again").

This replacement class needed to be updated both to remove font settings (we used "Microsoft Sans Serif" font instead of "MS SANS SERIF" font) that conflicted with new 17.x font handling and provide some additional capabilities. The result maintains the old interface (using the message box procedures and functions without the underscore) and provides a new interface to tap the new capabilities.

Some enhancement may be implemented in the future, including:

- Setting a maximum width for the message box, and letting the message box handle wrapping multiline text automatically.
- Specifying a monitor that the message box should appear on (right now the CENTER_ON_SCREEN locate mode is used which centers it on the primary monitor)
- Add other predefined bitmaps (not sure which ones would be useful…)
- Add capability of accepting input in forms (dates/strings/numbers)
- Add multilingual support for predefined buttons

If you have the time, feel free to implement any of these ideas and send me the code. ☺

Disclaimer: This code is provided "as-is". It seems pretty solid, but I only wrote it yesterday. I did give it a pretty good workout though. If you do find any bugs, please let me know: matthewd@datatechag.com.

Usage

Include the messagebox.dg at the desktop level. For “legacy” compatible usage, simply remove the underscores from your existing code:

```
Send Info_Box "This feature not implemented yet."
```

Is changed to:

```
Send InfoBox "This feature not implemented yet."
```

The following extensions have been made to the “legacy mode” interface:

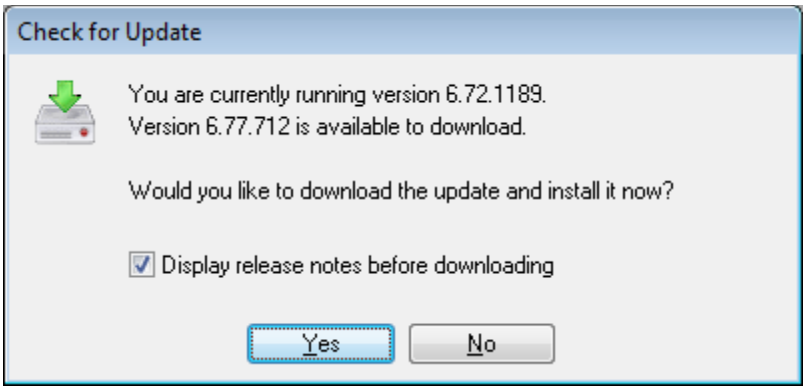
New messages are available including procedure WarningBox (displays the caution/warning symbol with an Ok button), function WarningCancelBox (warning with Ok and Cancel buttons), and function AbortRetryIgnoreBox (does exactly what it says on the tin).

An optional parameter is available on all messages to specify the Help_ID that should be displayed when the user clicks on the help button on the message box. If this parameter is not supplied or it is zero, then the message box will not have a help button.

Two additional parameters are available on the YesNoBox, YesNoCancelBox and AbortRetyIgnoreBox to provide the label for a checkbox and the default checked state of the checkbox.

```
Get YesNoBox ("You are currently running version"*sCurrentVersion+;
              ".\nVersion"*sAvailableVersion*"is available to download.\n\n"+;
              "Would you like to download the update and install it now?") "Check for Update" ;
              (MB_DEFBUTTON1+MB_ICONDOWNLOAD) 0 "Display release notes before downloading" ;
              True to iRetVal
```

The code above produces the following message:



The return value will indicate the button clicked and also has a bit (MBR_CHECKED) set that indicates the checked_state of the checkbox. Use iAnd to evaluate the return result when there is a checkbox, e.g.:

```
                If (iRetVal iand MBR_No=MBR_No) Procedure_Return
                If (iRetVal iand MBR_CHECKED) Begin
```

(MB_ICONDOWNLOAD was added as a predefined icon to display on messages boxes)

Extended Usage

Internally, each of the replacement message box interfaces compose a struct with all of the needed information to display the message, and pass that struct to a single interface that does the job of building and displaying the message box. You can use this interface to create more elaborate message boxes that include:

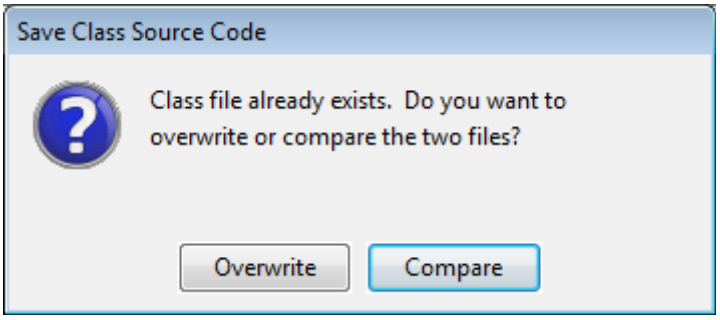
- Multiple sections of text that may be formatted independently
- Format options include font size/bold/italic/underline
- Additional 16x16 bitmaps that appears to the left of the text.
- Multiple checkboxes may be added to the message box instead of a single one
- Buttons other than the preprogrammed Ok/Cancel/Yes/No/Abort/Retry/Fail may be used

Example usage:

```
tmbMessageBox mb

Move MB_ICONQUESTION to mb.iBitmap
Move "Save Class Source Code" to mb.sTitle
Move "Class file already exists.  Do you want to\noverwrite or compare the two files?" to ;
     mb.Message[0].sMessage
Move "Overwrite" to mb.Button[0].sLabel
Move 10 to mb.Button[0].iType
Move "Compare" to mb.Button[1].sLabel
Move 11 to mb.Button[1].iType
Move True to mb.Button[1].bDefault
Send ShowMessageBox (&mb)
If (mb.iRetVal=11) Move (sFilename+".new") to sFilename   // check return value to see which
                                                          // button was clicked
                                                          //  10 = Overwrite
                                                          //  11 = Compare
```

This code produces the following message dialog:



At a minimum you must provide a title, a message, and at least button. Messages and buttons are defined by their own structures. The tmbMessageBox struct includes an array of messages and an array of buttons.

The struct is passed by reference, and the message box interface will modify the struct to include the button clicked (in the .iRetVal member) as well as the state of any checkboxes that are included in the message (in the .Message[x].bChecked members).

Bitmaps

Specify either a predefined bitmap (MB_ICON*) in the .iBitmap member or the name of a bitmap file in the .sBitmap member. Include the standard VDF transparency flags in the filename. The main bitmap for the message dialog should be 48x48 pixels. It can be smaller, but the positioning is based on a 48x48 image.

Help Button

Provide a help id in the .iHelp member. When the button is clicked, the help message is sent.

Messages

For each element in the "Message" array member, you must at a minimum specify the message text (.sMessage). Additionally, you can set the following members of the Message elements:

.sBitmap: A path to a 16x16 icon that will appear to the left of this message text

.iFontPointHeight/.bBold/.bItalic/.bUnderline: Font control settings, self explanatory I hope!

.bCheckbox/.bChecked: If bCheckbox is true, a checkbox control will be created instead of a text object. Multiline text is support for checkbox elements (additional lines will be indented to look nice). bChecked indicates the intial state of the checkbox and will contain the checked_state of the checkbox when the message box was closed.

When multiple messages are provided, a bit of extra spacing is provided between each message.

Buttons

For standard button, provide a predefined button type (MBR_Yes, MBR_No, etc) in the .iType member. It is not necessary to provide a label, as this will be assigned based on .iType.

For custom buttons, you may provide a label (including an "&" before the shortcut key for that button) and a value in the .iType member. It is important for custom buttons to **not** use any of the predefined types for standard buttons (1-7) or the type that is used internally for the help button (-1); if you do, then your label setting will not take effect. In the sample code, the values of 10 and 11 are used for the .iType values. Whatever values you supply for custom buttons are what you need to check for when the message box returns.

One button may be designated as the default button by setting the .bDefault member to true, this button will take the focus when the message box is displayed so that pressing space or [Enter] will select that button.

In addition to setting the values of the button structs explicity, there are some "helper functions" that return tmbButton structs for standard buttons: mbOkButton, mbYesButton, etc.

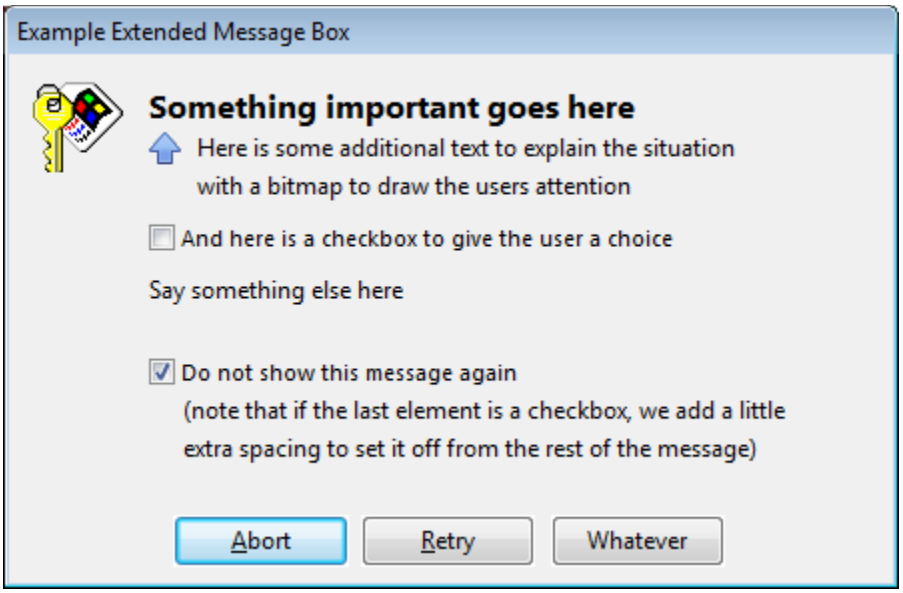
Final Example

Here is a final example to demonstrate most if not all of these capabilities:

```
tmbMessageBox mb

Move "Example Extended Message Box" to mb.sTitle
Move "login256.bmp/3d" to mb.sBitmap
Move "Something important goes here" to mb.Message[0].smessage
Move 12 to mb.Message[0].iFontPointHeight
Move True to mb.Message[0].bBold
Move ("Here is some additional text to explain the situation\n"+;
      "with a bitmap to draw the users attention") to mb.Message[1].sMessage
Move "moveup.bmp/3d" to mb.Message[1].sbitmap
Move "And here is a checkbox to give the user a choice" to mb.Message[2].sMessage
Move True to mb.Message[2].bCheckbox
Move "Say something else here" to mb.Message[3].sMessage
Move ("Do not show this message again\n"+;
      "(note that if the last element is a checkbox, we add a little\n"+;
      "extra spacing to set it off from the rest of the message)") to mb.Message[4].sMessage
Move True to mb.Message[4].bCheckbox
Move True to mb.Message[4].bChecked
Get mbAbortButton of (omessagebox(Self)) to mb.Button[0]
Get mbRetryButton of (omessagebox(Self)) to mb.Button[1]
Move "Whatever" to mb.Button[2].sLabel
Move 42 to mb.Button[2].iType
Send ShowMessageBox (&mb)
```

Results in the following message dialog:



The bitmaps folder contains some sample bitmaps for use with the message boxes that were generated using IconWorkshop by Axialis. If, like me you are not a graphic designer, IconWorkshop can help you generate decent looking icons quickly. They also have some nice collections of Icons for sale (that can of course be modified in IconWorkshop) at reasonable prices (especially if you get them during one of their 50% off sales).