VCL

# TMS Metro Controls Pack
DEVELOPERS GUIDE

**June 2020**

## Index

## Availability

TMS Metro Controls Pack is a VCL component set for Win32 & Win64 application development* and is available for Embarcadero™ Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10.0 Seattle, 10.1 Tokyo, 10.2 Berlin, 10.3 Rio, 10.4 Sydney & Embarcadero™ C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10.0 Seattle, 10.1 Berlin, 10.2 Tokyo, 10.3 Rio, 10.4 Sydney.

*Win64 development requires RAD Studio XE2 or newer versions.

## Online references

TMS software website:
https://www.tmssoftware.com

TMS Metro Controls Pack page:
https://www.tmssoftware.com/site/advmetro.asp

## Purchase a license

TMS Metro Controls are available in the following bundles:

- TMS Labels & buttons Pack : https://www.tmssoftware.com/site/labelsbuttons.asp
- TMS Component Pack: https://www.tmssoftware.com/site/tmspack.asp
- TMS Component Studio : https://www.tmssoftware.com/site/studio.asp
- TMS VCL Subscription : https://www.tmssoftware.com/site/vdsub.asp

## List of included components

- TAdvMetroForm
- TAdvMetroTaskDialog
- TAdvInputMetroTaskDialog
- TAdvMetroButton
- TAdvMetroToolButton
- TAdvMetroHint
- TAdvMetroProgressBar
- TAdvMetroScrollBox

## Introduction

The TMS Metro Controls Pack offers the form, dialog & base visual control classes to create applications with a look & feel that is inspired by the Microsoft Metro design language. Some key Microsoft Metro style design language guidelines comprise following:

- Focus on typography
- Simple symbol language
- Minimalistic color usage

A good starting point to read about the Microsoft Metro design language can be found at:
https://en.wikipedia.org/wiki/Metro_(design_language)

TMS Metro Controls Pack includes forms and dialogs that have this minimalistic flat look & feel. These are direct replacements for the standard VCL forms and dialogs. In addition, it includes TAdvMetroButton, TAdvMetroScrollBox, TAdvMetroHint, TAdvMetroProgressBar with a style that fit in Metro style forms & dialogs.

## Architecture

All TMS Metro enabled visual controls implement the ITMSTones interface. Via this interface, the colors of the visual control are setup. The ITMSTones is a simple interface with a single method: SetColorTones(TColorTones). TColorTones is a record structure holding the colors that specify how the Metro style looks. By calling the ITMSTones.SetColorTones() method for all visual controls in an application, the entire appearance of a Metro style VCL application can be changed. The TColorTone record structure is:

```
TColorTone = record
  BrushColor: TColor;
  BorderColor: TColor;
  TextColor: TColor;
end;


TColorTones = record
  Background: TColorTone;
  Foreground: TColorTone;
  Selected: TColorTone; // =Down
  Hover: TColorTone;
  Disabled: TColorTone;
end;
```

The TColorTone structure is defined in the unit ADVSTYLEIF.PAS. TColorTone holds the brush color, border color and text color of a visual element. TColorTones is a record structure that holds these settings for the different states of visual elements. In normal state of a visual element, two settings are possible: Foreground and Background. Other than the normal state, the color settings for visual elements in selected or down state are controlled by Selected: TColorTone, the color settings for the state where the mouse is over a visual element are controlled by Hover: TColorTone and finally, the color settings for disabled visual elements

It is the responsibility of the component to handle the SetColorTones() call and set the colors of all parts or elements of the control in a way that fits the Metro design language. Typically, in the Metro design language, there is a single accent color on either a white or black background. The accent color would be set with the Foreground  TColorTone record, the white or black background is set with the Background TColorTone record.

The unit ADVSTYLEIF.PAS offers 4 helper functions:

```
function DefaultMetroTones: TColorTones;
```

This returns the default white background, lightblue accent color Metro colors

```
function CreateMetroTones(Light: boolean; Color, TextColor: TColor):
TColorTones;
```

This function creates a TColorTones record with Color,TextColor the accent color on either a light (white) or dark (black) background color.

```
function ClearTones: TColorTones;
```

This function returns a null TColorTones record.

```
function IsClearTones(ATones: TColorTones): boolean;
```

This function returns true when the TColorTones record is a null record. A null TColorTones record means a record where all colors are clNone and is treated as a non-initialized record.

A helper component to setup all visual controls on a form, including the form itself, to the same Metro colors is TAdvFormStyler / TAdvAppStyler. The TAdvFormStyler has properties MetroColor, MetroTextColor that set the accent color and the property MetroStyle [msLight, msDark]. When the accent color is changed, or TAdvFormStyler.Metro is set to true, it will scan the entire form for all visual controls implementing the ITMSTones interface and call its SetColorTones() method with the TColorTones record that resulted from CreateMetroTones(). This way, an entire form Metro appearance can be changed with setting a single property. By adding the ITMSTones interface to a custom control, it can be made consistent with the other visual controls on the form supporting the Metro style.

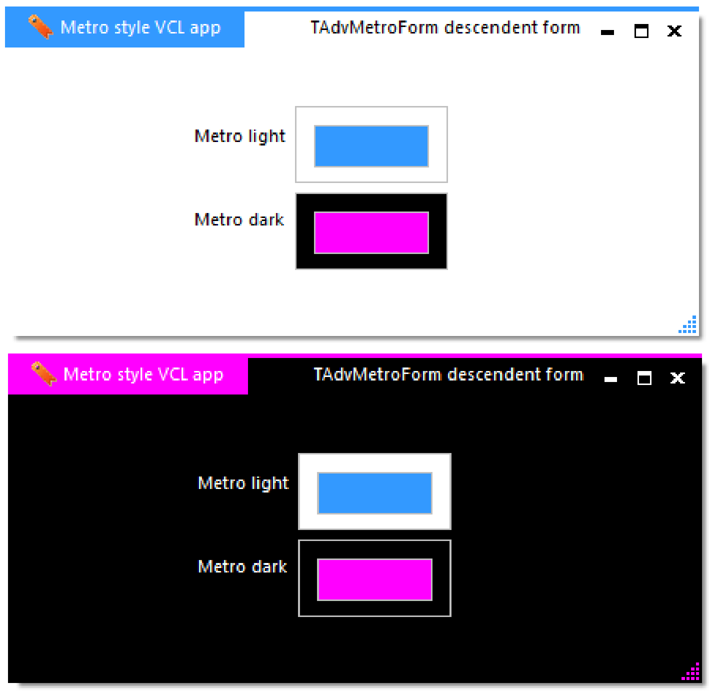
Example:

```
  TMyControl = class(TCustomControl, ITMSTones)
  private
    FAColor: TColor;
  public
    procedure SetColorTones(ATones: TColorTones);
  published
    property AColor: TColor read FAColor write FAColor;
  end;


{ TMyControl }

procedure TMyControl.SetColorTones(ATones: TColorTones);
begin
  // map all TColorTones here to the appropriate color settings
  // in the control

  FAColor := ATones.Background.BrushColor;


  // etc … for other control color properties

end;
```

An additional article with information about what TAdvFormStyler and TAdvAppStyler can do in your applications, now also with Metro style applications can be found here:
http://www.tmssoftware.com/site/atbdev3.asp

## TAdvMetroForm

To start creating forms adhering the Metro design-language, descend your forms from TAdvMetroForm instead of TForm. TAdvMetroForm has built-in support for a flat, minimalistic design and implements the ITMSTones interface:



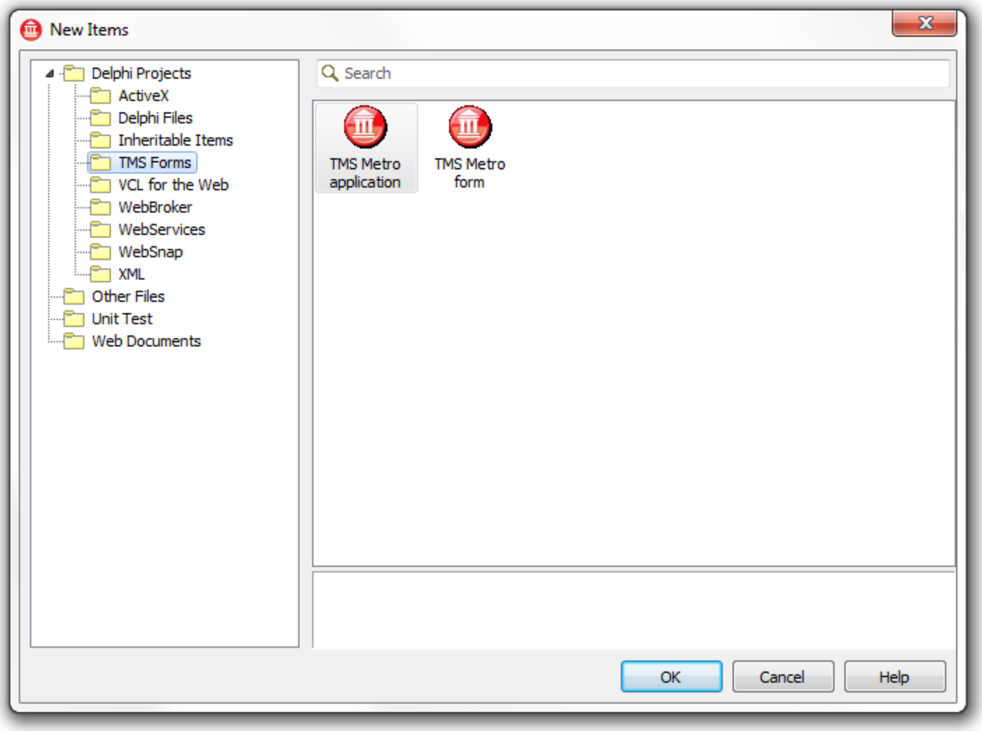
An existing VCL form can be turned into a TAdvMetroForm by simply adding the AdvMetroForm unit reference to the user list and changing its base class, i.e. replace:

```
TMyForm = class(TForm)
```

by

```
TMyForm = class(TAdvMetroForm)
```

A new Metro style form or Metro style application can also be created from the Delphi repository:



The additional properties that TAdvMetroForm exposes compared to TForm are:

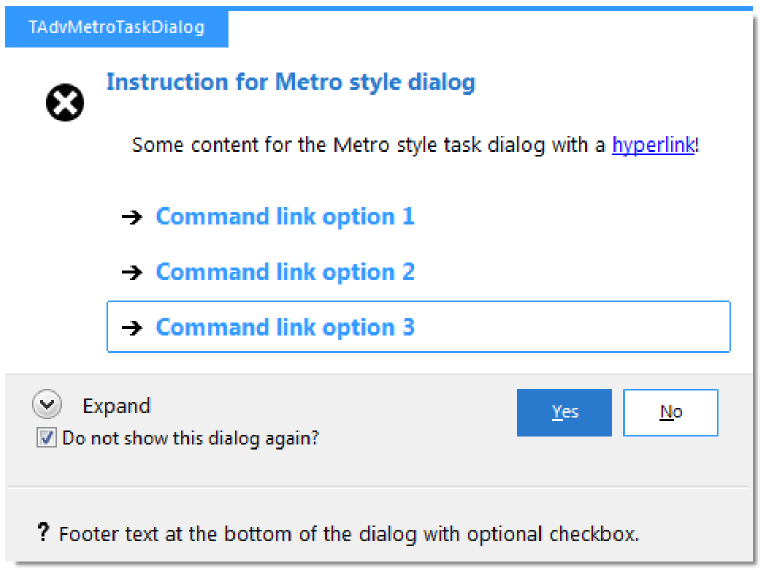
Appearance: TFormAppearance

SystemIconColor: TColor : sets the normal state color of system buttons on the caption.
SystemIconColorHot: TColor : sets the hover state color of system buttons on the caption.
SystemIconColorDown: TColor : sets the down state color of system buttons on the caption.
SystemIconColorDisabled: TColor : sets the disable state color of system buttons on the caption.
Color: TColor : sets the top area background color.
SizeGripColor: TColor : sets the color of the sizegrip in the bottom right corner of the form.
CaptionFont: TFont : sets the font for the caption in the top left corner of the form.
CaptionColor: TColor : sets the background color of the left caption part.
CaptionStyle: TCaptionStyle : when set to csMetro, the caption is divided in a left part in a different color than the full caption. When set to csPlain, the caption is drawn over the entire width of the form.
CaptionActiveColor: TColor : sets the color of the caption when the form is active.
Font: TFont : sets the font of the main caption part.
TextAlign: TAlignment : sets the alignment of Text on the caption.
TextColor: TColor : sets the color of Text.
ProgressColor: TColor : sets the color of the progress indication drawn as line at the top of the form.

ImageIndex: integer : sets the index of the image in the Image list to display in the caption.
Images: TCustomImageList: sets the Image list from where an image for the caption can be selected.
NoDropShadow: Boolean : when true, no shadow is shown around the form.
SizeGrip: Boolean  : when true a sizegrip is shown in the bottom right corner of the form.
ShowProgress: Boolean : when true, an animation is shown on the top line of the form to indicate something is being processed in the application.
Text: string : sets the text in the top right corner of the form caption.

## TAdvMetroTaskDialog, TAdvInputMetroTaskDialog

TAdvMetroTaskDialog and TAdvInputMetroTaskDialog are Metro style equivalents of the new task dialog that Microsoft introduced in Windows Vista:


TAdvMetroTaskDialog

Instruction for Metro style dialog

Some content for the Metro style task dialog with a hyperlink!

Command link option 1

Command link option 2

Command link option 3

Expand

Do not show this dialog again?

Yes No

Footer text at the bottom of the dialog with optional checkbox.

A task dialog is a dialog that is more complex than a regular MessageDlg() dialog, can show more information and get a response from the user in different ways.

The task dialog provides:

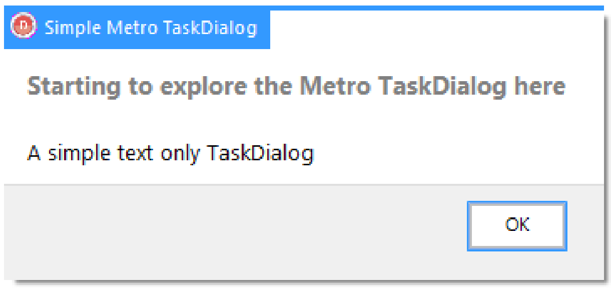
1. A caption
2. An icon representing the action, set by AdvMetroTaskDialog.Icon.
3. An main instruction text in bold, set by AdvMetroTaskDialog.Instruction.
4. The main content under the instruction, set by AdvMetroTaskDialog.Content.
5. Optionally multiple choices, either represented by radiobuttons or normal buttons (command links). The options are set with the AdvMetroTaskDialog.RadioButtons TStringList property
6. Extra content that is can be hidden or shown with the expand/collaps button. The extra content is

set by the property  AdvMetroTaskDialog.ExpandedText.
7. An optional checkbox that typically controls whether the user wants to see the dialog again when a similar action happens or not. The checkbox is displayed when text is set for AdvMetroTaskDialog.VerificationText
8. An optional footer text at the bottom of the dialog set with AdvMetroTaskDialog.Footer and an optional icon set with AdvMetroTaskDialog.FooterIcon.

Examples:

This first sample uses the TAdvMetroTaskDialog as a simple dialog box with just an instruction and content:

```
  AdvMetroTaskDialog1.Title := 'Simple Metro TaskDialog';
  AdvMetroTaskDialog1.Instruction := 'Starting to explore the Metro
TaskDialog here';
  AdvMetroTaskDialog1.Content := 'A simple text only TaskDialog';
  AdvMetroTaskDialog1.CommonButtons := [cbOK];
  AdvMetroTaskDialog1.Execute;
```

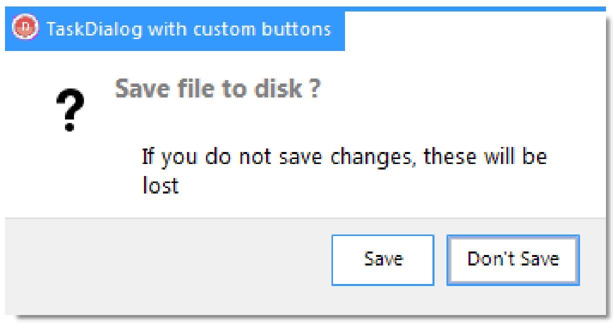


Starting to explore the Metro TaskDialog here

A simple text only TaskDialog

In the first example, a common OK button was choosen. With the property CustomButtons, it is possible to specify your own text for the button. Where for common buttons, AdvMetroTaskDialog.Execute returns the common Windows values for Ok, Cancel, Yes, No, ... the first custom button returns 100, the second 101, etc...

```
  AdvMetroTaskDialog1.Title := 'TaskDialog with custom buttons';
  AdvMetroTaskDialog1.Icon := tiQuestion;
  AdvMetroTaskDialog1.CustomButtons.Clear;
  AdvMetroTaskDialog1.CustomButtons.Add('Save');
  AdvMetroTaskDialog1.CustomButtons.Add('Don''t Save');
  AdvMetroTaskDialog1.DefaultButton := 101;
  AdvMetroTaskDialog1.Instruction := 'Save file to disk ?';
  AdvMetroTaskDialog1.Content := 'If you do not save changes, these
will be lost';
  ShowMessage(inttostr(AdvMetroTaskDialog1.Execute));
```

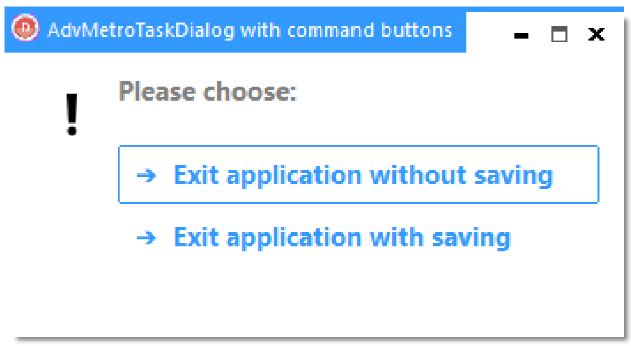

TaskDialog with custom buttons

Save file to disk ?

If you do not save changes, these will be lost

To make the possible actions standout, the buttons can be turned into CommandButtons like in the screenshot below. The code is very similar to the previous sample, just the setting doCommandLinks was added in TAdvMetroTaskDialog.Options:

```
  AdvMetroTaskDialog1.Title := 'AdvMetroTaskDialog with command
buttons';
  AdvMetroTaskDialog1.Icon := tiWarning;
  AdvMetroTaskDialog1.CustomButtons.Clear;
  AdvMetroTaskDialog1.CustomButtons.Add('Exit application without
saving');
  AdvMetroTaskDialog1.CustomButtons.Add('Exit application with
saving');
  AdvMetroTaskDialog1.DefaultButton := 100;
  AdvMetroTaskDialog1.Options := [doCommandLinks];
  AdvMetroTaskDialog1.Execute;
```

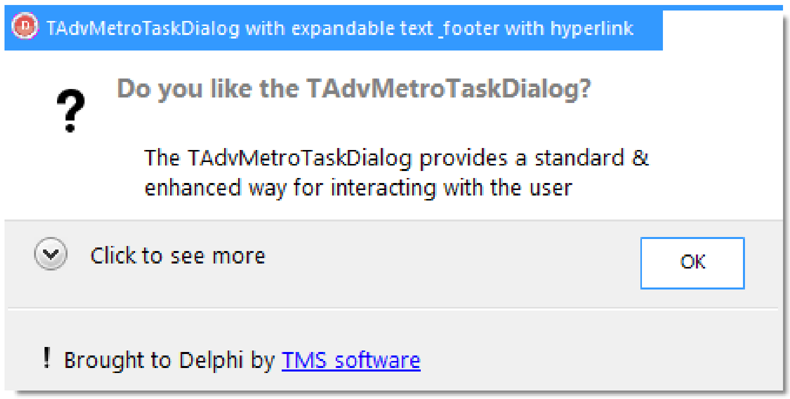

AdvMetroTaskDialog with command buttons

Please choose:

!
→ Exit application without saving
→ Exit application with saving

To make dialogs more clear & concise, it is possible to optionally hide detail text that expert users might want to see. The detail text is set with the ExpandedText property. As soon as this contains a non-empty text, it will be shown in the dialog after clicking on the arrow expand button. It is possible to override the default text for the expand/collaps button as well with the properties ExpandControlText, CollapsControlText. To activate the user of hyperlinks in TAdvMetroTaskDialog text, it is required to set doHyperlinks = true in TAdvMetroTaskDialog.Options. When the hyperlink is clicked, the event OnDialogHyperlinkClick is triggered.

```
  AdvMetroTaskDialog1.Options := [doHyperlinks];
  AdvMetroTaskDialog1.Title := 'TAdvMetroTaskDialog with expandable
text & footer with hyperlink';
  AdvMetroTaskDialog1.Instruction := 'Do you like the
TAdvMetroTaskDialog?';
  AdvMetroTaskDialog1.Icon := tiQuestion;
  AdvMetroTaskDialog1.Content := 'The TAdvMetroTaskDialog provides a
standard & enhanced way for interacting with the user';
  AdvMetroTaskDialog1.ExpandedText := 'Many new options make the
TAdvMetroTaskDialog very different from the old Windows MessageBox';
  AdvMetroTaskDialog1.ExpandControlText := 'Click to hide';
  AdvMetroTaskDialog1.CollapsControlText := 'Click to see more';
  AdvMetroTaskDialog1.Footer := 'Brought to Delphi by <A
href="http://www.tmssoftware.com">TMS software</A>';
  AdvMetroTaskDialog1.FooterIcon := tfiWarning;
  AdvMetroTaskDialog1.Execute;
```


TAdvMetroTaskDialog with expandable text _footer with hyperlink

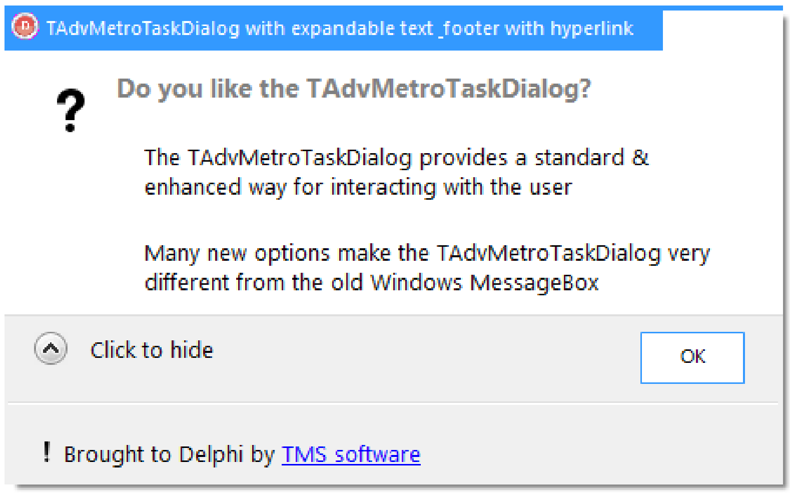
Do you like the TAdvMetroTaskDialog?

The TAdvMetroTaskDialog provides a standard & enhanced way for interacting with the user

Click to see more

OK

! Brought to Delphi by TMS software


TAdvMetroTaskDialog with expandable text _footer with hyperlink

Do you like the TAdvMetroTaskDialog?

The TAdvMetroTaskDialog provides a standard & enhanced way for interacting with the user

Many new options make the TAdvMetroTaskDialog very different from the old Windows MessageBox

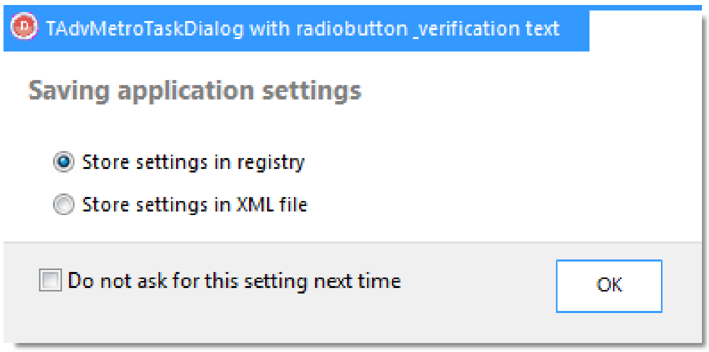
Click to hide

OK

! Brought to Delphi by TMS software

A TAdvMetroTaskDialog can also contain a series of radiobuttons to allow the user to make a choice. In addition, the typical checkbox can be added that allows the user to decide whether this dialog should be displayed in the future or not. The choice of the radiobuttons is returned by TAdvMetroTaskDialog.RadioResult. The result for the first radiobutton is 200, the 2nd radiobutton 201, etc... The selection of the verification checkbox is returned by TAdvMetroTaskDialog.VerifyResult.

```
  AdvMetroTaskDialog1.Title := 'TAdvMetroTaskDialog with radiobutton
& verification text';

  AdvMetroTaskDialog1.RadioButtons.Clear;

  AdvMetroTaskDialog1.RadioButtons.Add('Store settings in
registry');

  AdvMetroTaskDialog1.RadioButtons.Add('Store settings in XML
file');

  AdvMetroTaskDialog1.VerificationText := 'Do not ask for this
setting next time';

  AdvMetroTaskDialog1.Instruction := 'Saving application settings';

  AdvMetroTaskDialog1.Execute;

  case  AdvMetroTaskDialog1.RadioButtonResult of

  200: ShowMessage('Store in registry');

  201: ShowMessage('Store in XML');

  end;


  if AdvMetroTaskDialog1.VerifyResult then

    ShowMessage('Do not ask for this setting next time');
```


TAdvMetroTaskDialog with radiobutton _verification text

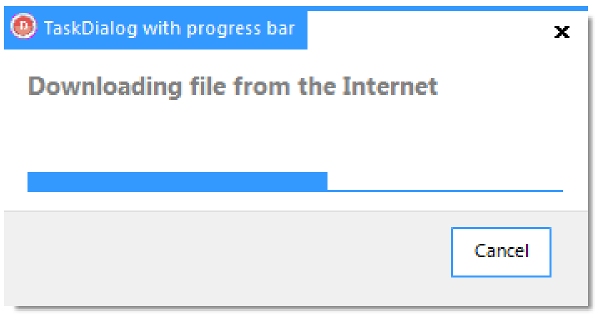
Saving application settings

● Store settings in registry
○ Store settings in XML file

☐ Do not ask for this setting next time

Finally, it is equally possible to use the TAdvMetroTaskDialog for progress indication. To enable a progressbar on the TAdvMetroTaskDialog, set doProgressBar = true in TAdvMetroTaskDialog.Options. From the event TAdvMetroTaskDialog.OnDialogProgress, the position of the progressbar is queried. By default, the TAdvMetroTaskDialog progress position is between 0 and 100 but can be set to other values with ProgressBarMin & ProgressBarMax properties. The sample code snippet shows how to setup the TAdvMetroTaskDialog with progressbar and a simple event that updates the progress. In addition, by invoking the TAdvMetroTaskDialog.ClickButton() when progress is 100%, the code makes the dialog automatically disappear when the process is complete.

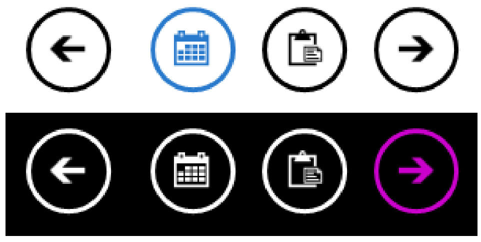
```
TMyForm = class(TAdvMetroForm)
public
    procedure DoDialogProgress(Sender: TObject; var Pos: Integer;
var State: TTaskDialogProgressState);
end;

  progresspos := 0;
  AdvMetroTaskDialog1.Title := 'TaskDialog with progress bar';
  AdvMetroTaskDialog1.Instruction := 'Downloading file from the
Internet';
  AdvMetroTaskDialog1.Options := [doProgressBar];
  AdvMetroTaskDialog1.CommonButtons := [cbCancel];
  AdvMetroTaskDialog1.OnDialogProgress := DoDialogProgress;
  AdvMetroTaskDialog1.Execute;

procedure TTMSForm4.DoDialogProgress(Sender: TObject; var Pos:
Integer;
  var State: TTaskDialogProgressState);

begin
  if (progresspos < 100) then
    inc(progresspos,2)   // increment in steps of 2
  else
    (Sender as TAdvMetroTaskDialog).ClickButton(1);
  // return progressbar position
  Pos := progresspos;
  State := psNormal;
end;
```



Downloading file from the Internet

## TAdvMetroButton, TAdvMetroToolButton

As introduced in the first paragraph, one of the elements in the Microsoft Metro design language is the use of symbols. Often, user interface actions are represented by simple minimalistic symbols. When clicked, a certain action is performed. To make it easy, the TAdvMetroButton and TAdvMetroToolButton are two button control that will take a bicolor symbol image file and automatically render it in the Metro colors for the various states of the button. The TAdvMetroButton and TAdvMetroToolButton are by design transparent, show the image centered and can optionally display a circular around the symbol:



This makes it very simple to use a single bi-color graphic (preferably a PNG image for better anti-aliasing) and have TAdvMetroButton automatically adapt its colors depending on the choosen Metro color for each state of the button. Therefore, the TAdvMetroButton will implement the ITMSTones interface. To have the automatic coloring of the symbol in the image work correct, use images with a black symbol on a white background.

Additional properties that TAdvMetroButton, TAdvMetroToolButton add to a TButton:

Appearance.PictureColor: this is the color of the symbol when the button is in normal state.

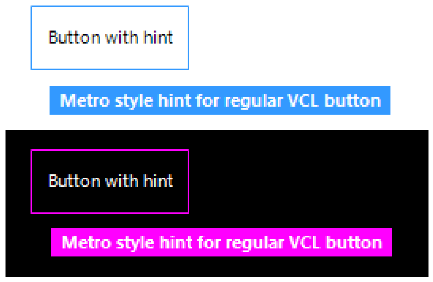
Appearance.PictureColorDown: this is the color of the symbol when the button is in down state.

Appearance.PictureColorHover: this is the color of the symbol when the button is in hovered state.

Picture: TPicture : this sets the symbol image. It is preferred to use a PNG image for its anti-aliasing capabilities (supported in Delphi 2010 and higher)

## TAdvMetroHint

In the Microsoft Metro design language, also hints have a minimalistic design. To enable such Metro styled hints, drop a single TAdvMetroHint instance on a form and all hints of controls will be shown with the Metro style. TAdvMetroHint implements the ITMSTones interface and will therefore also automatically let the hint color adapt with the chosen Metro color set.


Button with hint
Metro style hint for regular VCL button
Button with hint
Metro style hint for regular VCL button

The settings for TAdvMetroHint are:

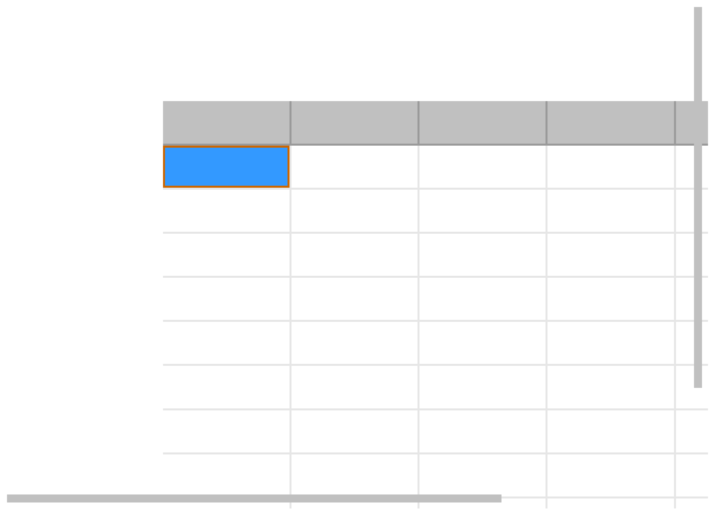
HintColor: TColor : background color of the hint.
HintFont: TFont : font used on the hint.
HinStyle: THintStyle : hsRectangle for a rectangular hint and hsRounded for a rounded rectangular hint.
LineColor: TColor : when different from clNone, a line is shown at the top of the hint in the LineColor.
Shadow: Boolean : when true, a shadow is shown.

## TAdvMetroProgressBar

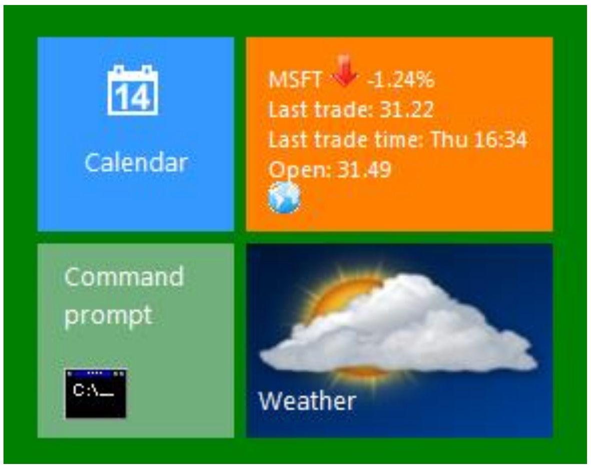
The TAdvMetroProgressBar is a simple progress bar control that respects the Metro design language. It is simple & minimalistic. At the bottom of the control, a fine line is drawn while the position of the progressbar is drawn over the full client height of the control. The TAdvMetroProgressbar has a single Color property, is transparent and implements the ITMSTones interface that will set its color property via the SetColorTones interface method.

## TAdvMetroScrollBox

TAdvMetroScrollBox is a component equivalent to TScrollBox but uses simple minimalistic unobtrusive scrollbars. A scroll can be performed by clicking the scroller rectangular elements and dragging. The TAdvMetroScrollBox implements the ITMSTones interface and its background color will automatically adapt to the selected Metro style color. The size of the scrollers automatically adapts to the size & position of the child controls in the scrollbox.



## TAdvMetroTile



TAdvMetroTile is a sophisticated button with a functionality similar to the tiles that can be found in the Windows 8 Metro tile menu. The TAdvMetroTile implements the ITMSTones interface and its background color will automatically adapt to the selected Metro style color.  This means the TAdvMetroTile.Appearance property that controls the color, bordercolor, textcolor of the tile in its states can be automatically updated via the TAdvFormStyler.

The TAdvMetroTile consists of a picture and text and the text can have HTML formatting. This means that the HTML formatted text can also include images (via the IMG tag). For a full specification of the supported HTML tags, please refer to: http://www.tmssoftware.com/site/minihtml.asp.

The picture and the text position on the tile are controlled by the Layout property.

Layout:
plPicLeft : picture is on left side of the tile
plPicRight: picture is on right side of the tile
plPicTop: picture is on top side of the tile
plPicBottom: picture is on bottom side of the tile
plPicBackground: picture is used as background of the tile, text is on top of the image

The picture and the text are positioned with a margin from the border in the tile. The margins are set for the picture with AdvMetroTile.PictureMarginHorz, AdvMetroTile.PictureMarginVert and for the text with AdvMetroTile.TextMarginHorz, AdvMetroTile.TextMarginVert.

With respect to displaying the picture in the tile, several extra capabilities are offered:

PictureAutoBrighten: when this property is set to true, when the mouse is hovered over the tile, the picture brightness is automatically increased. When the mouse is down in the tile, the brightness of the picture is automatically decreased.

PictureAutoColor: when used with a monochrome black PNG image and this property is set to true, the picture will displayed with the same color as the text color. This means that the black color pixels in the image will automatically be rendered in the text color, i.e. different text colors for the 4 different states of the button: normal, hover, down, disabled.

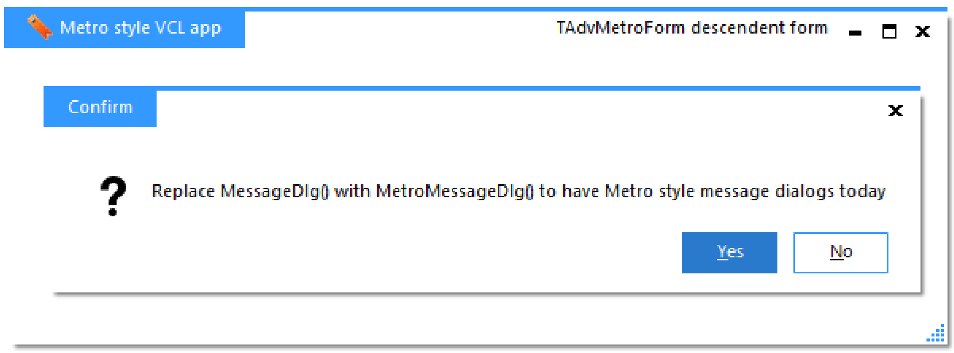
PictureAutoSize: when true, the size of the picture will be automatically adapted to fill the tile, otherwise, the picture is displayed in its default size.

ZoomOnHover: when this value is > 0, this means the tile will automatically increase its size in the number of pixels defined by ZoomOnHover when the mouse is over the tile.

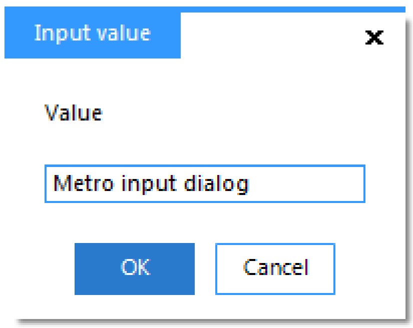
## Metro dialog functions

The units ADVMETRODLGS.PAS and ADVMETROTASKDIALOG.PAS provide lots of direct equivalent dialog functions that show the dialogs in Metro style:

In ADVMETRODLGS.PAS:



Replace MessageDlg() with MetroMessageDlg() to have Metro style message dialogs today



Direct replacements for the VCL ShowMessage() procedure:

```
procedure MetroShowMessage(Text: string); overload;

procedure MetroShowMessage(Text, Caption: string); overload;

procedure MetroShowMessage(Text, Caption, CaptionText: string);
overload;
```

Direct replacement for the VCL ShowMessageFmt() procedure:

```
procedure MetroShowMessageFmt(const Instruction: string; Parameters:
array of const);
```

Direct replacement for the VCL MessageDlg() function:

```
function MetroMessageDlg(const Instruction: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint): Integer; overload;



function MetroMessageDlg(const Instruction: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; DefaultButton:
TMsgDlgBtn): Integer; overload;
```

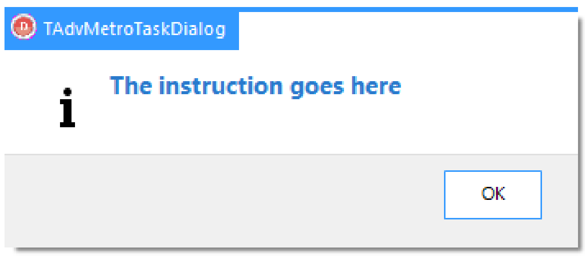
Direct replacement for the Windows MessageBox() function:

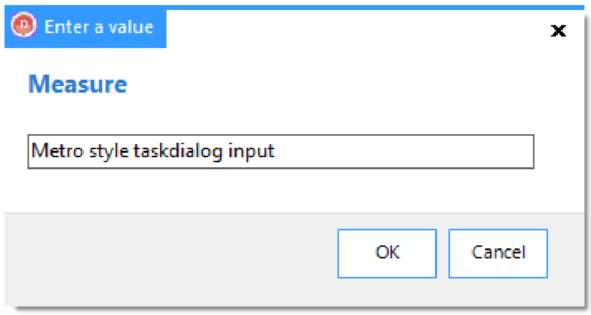
```
function MetroMessageBox(hWnd: HWND; lpInstruction, lpTitle: PChar;
flags: UINT): Integer;
```

Direct replacement for the VCL InputQuery() function:

```
function MetroInputQueryDlg(ACaption, APrompt: string; var Value:
string): boolean;
```

In ADVMETROTASKDIALOG.PAS:





Task dialog based direct equivalents for the VCL ShowMessage() function:

```
function MetroTaskShowMessage(const Instruction: string): boolean;
overload;

function MetroTaskShowMessage(const Title, Instruction: string):
boolean; overload;

function MetroTaskShowmessage(const Title, Instruction: string;
tiIcon: TTaskDialogIcon): boolean; overload;

function MetroTaskShowMessage(const Title, Instruction, content,
verify: string;

  tiIcon: TTaskDialogIcon): boolean; overload;
```

Task dialog based direct equivalents for the VCL ShowMessageFmt() function:

```
function MetroTaskShowMessageFmt(const Instruction: string;
Parameters: array of const): boolean;
```

Task dialog based direct equivalents for the Window MessageBox() function:

```
function MetroTaskMessageBox(hWnd: HWND; lpInstruction, lpTitle:
PChar; flags: UINT): Integer;
```

Task dialog based direct equivalents for the VCL MessageDlg() function:

```
function MetroTaskMessageDlg(const Instruction: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint): Integer; overload;
```

```
function MetroTaskMessageDlg(const Instruction: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; DefaultButton:
TMsgDlgBtn): Integer; overload;


function MetroTaskMessageDlg(const Title, Msg: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint): Integer; overload;


function MetroTaskMessageDlg(const Title, Msg: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; DefaultButton:
TMsgDlgBtn): Integer; overload;


function MetroTaskMessageDlgPos(const Title, Msg: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer):
Integer; overload;


function MetroTaskMessageDlgPos(const Title, Msg: string; DlgType:
TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer;

  DefaultButton: TMsgDlgBtn): Integer; overload;


function MetroTaskMessageDlgPosHelp(const Title, Msg: string;
DlgType: TMsgDlgType;

  Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer;

  const HelpFileName: string): Integer; overload;


function MetroTaskMessageDlgPosHelp(const Title, Msg: string;
DlgType: TMsgDlgType;
```

```
  Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer;

  const HelpFileName: string; DefaultButton: TMsgDlgBtn): Integer;
overload;
```

Task dialog based direct equivalents for the VCL InputQuery() function:

```
function MetroInputQueryDlg(ACaption, APrompt: string; var Value:
string): boolean;
```