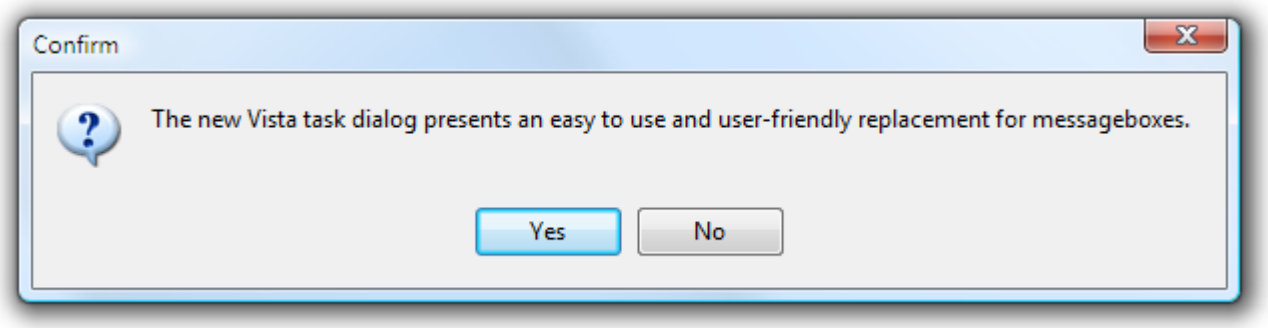
Taking the new Windows Vista TaskDialog one step further

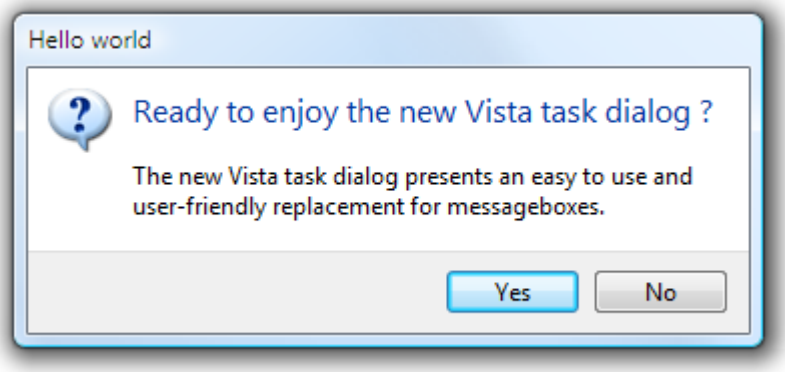
Introduction

Windows Vista introduces a new dialog to communicate with the user, called TaskDialog. In its most simple form, a TaskDialog is just a nicer looking MessageBox equivalent with options to customize the title and to add a description and content.

Pre Windows Vista dialog boxes



Windows Vista basic TaskDialog equivalent:



Other than the basic task dialog, Windows Vista offers many more capabilities to extend the dialog with selectors for multiple possibilities, verify checkbox, optionally hidden extra information, progressbars etc... These extra capabilities will be discussed in a next article.

Using the basic TaskDialog from Delphi

Using the TaskDialog from Delphi is simple. Windows Vista makes the TaskDialog available as a Win32 API call in the new COMCTL32.DLL v6. It is important to note that Windows Vista still ships with COMCTL32.DLL v5 as well and this is the default library an application uses from Delphi. To use COMCTL32.DLL v6 and thus also the TaskDialog API, make sure the manifest TXPManifest or TMS TWinXP component in your app.
The TaskDialog API function is declared in the Windows Vista SDK as:

```
HRESULT TaskDialog(HWND hWndParent,
    HINSTANCE hInstance,
    PCWSTR pszWindowTitle,
    PCWSTR pszMainInstruction,
    PCWSTR pszContent,
```

```
    TASKDIALOG_COMMON_BUTTON_FLAGS dwCommonButtons,
    PCWSTR pszIcon,
    int *pnButton
);
```

this translates in Delphi to a function:

```
TaskDialog: function(HWND: THandle; hInstance:
THandle; cTitle, cDescription, cContent: pwidechar; Buttons: Integer; Icon:
integer; ResButton: pinteger): integer;
```

A disadvantage of the new TaskDialog is that this will only work on Windows Vista. Earlier versions of the Windows operating system simply do not support it. We have therefore created a function to use the basic TaskDialog on Windows Vista that will fallback to the standard MessageDlg for older operating systems. This way, your application will have the latest user-interface appearance on Windows Vista and simultaneously continue to work on older versions of Windows. The full code of this new TaskDialog method is included here below. An extra TaskMessage function is provided for replacing the Delphi ShowMessage as well.

```
const
TD_ICON_BLANK = 100;
TD_ICON_WARNING = 101;
TD_ICON_QUESTION = 102;
TD_ICON_ERROR = 103;
TD_ICON_INFORMATION = 104;
TD_ICON_BLANK_AGAIN = 105;
TD_ICON_SHIELD = 106;

TD_OK = 1;
TD_YES = 2;
TD_NO = 4;
TD_CANCEL = 8;
TD_RETRY = 16;
TD_CLOSE = 32;

DLGRES_OK = 1;
DLGRES_CANCEL = 2;
DLGRES_RETRY = 4;
DLGRES_YES = 6;
DLGRES_NO = 7;
DLGRES_CLOSE = 8;

function TaskDialog(AForm: TCustomForm; ATitle,
ADescription, AContent: string; Buttons,Icon: integer): integer;
var
  VerInfo: TOSVersioninfo;
  DLLHandle:
THandle;
  res: integer;
  wTitle,wDescription,wContent:
array[0..1024] of widechar;
  Btns:
TMsgDlgButtons;
  DlgType:
TMsgDlgType;
  TaskDialogProc:
function(HWND: THandle; hInstance: THandle; cTitle, cDescription, cContent:
pwidechar; Buttons: Integer; Icon: integer;
       ResButton: pinteger): integer; cdecl
stdcall;
```

```
begin
  Result := 0;

  VerInfo.dwOSVersionInfoSize := SizeOf(TOSVersionInfo);
  GetVersionEx(verinfo);

  if (verinfo.dwMajorVersion >= 6) then
  begin
    DLLHandle := LoadLibrary('comctl32.dll');
    if DLLHandle >= 32 then
    begin
      @TaskDialogProc := GetProcAddress(DLLHandle,'TaskDialog');

      if Assigned(TaskDialogProc) then
      begin
        StringToWideChar(ATitle, wTitle, sizeof(wTitle));
        StringToWideChar(ADescription, wDescription, sizeof(wDescription));
        StringToWideChar(AContent, wContent, sizeof(wContent));
        TaskDialogProc(AForm.Handle, 0, wTitle, wDescription, wContent,
Buttons,Icon,@res);

        Result := mrOK;

        case res of
        DLGRES_CANCEL : Result := mrCancel;
        DLGRES_RETRY : Result := mrRetry;
        DLGRES_YES : Result := mrYes;
        DLGRES_NO : Result := mrNo;
        DLGRES_CLOSE : Result := mrAbort;
        end;
      end;
      FreeLibrary(DLLHandle);
    end;
  end
  else
  begin
    Btns := [];
    if Buttons and TD_OK = TD_OK then
      Btns := Btns + [MBOK];

    if Buttons and TD_YES = TD_YES then
      Btns := Btns + [MBYES];

    if Buttons and TD_NO = TD_NO then
      Btns := Btns + [MBNO];

    if Buttons and TD_CANCEL = TD_CANCEL then
      Btns := Btns + [MBCANCEL];

    if Buttons and TD_RETRY = TD_RETRY then
      Btns := Btns + [MBRETRY];

    if Buttons and TD_CLOSE = TD_CLOSE then
      Btns := Btns + [MBABORT];

    DlgType := mtCustom;

    case Icon of
    TD_ICON_WARNING : DlgType := mtWarning;
    TD_ICON_QUESTION : DlgType := mtConfirmation;
```

```
    TD_ICON_ERROR : DlgType := mtError;
    TD_ICON_INFORMATION: DlgType := mtInformation;
    end;

    Result := MessageDlg(AContent, DlgType, Btns, 0);
  end;
end;

procedure TaskMessage(AForm: TCustomForm; AMessage: string);
begin
  TaskDialog(AForm, '', '', AMessage, TD_OK, 0);
end;
```

This sample code snippet uses the new TaskDialog and TaskMessage functions:

```
 if TaskDialog(self, 'Hello world','Ready to enjoy the new Vista task dialog ?',
   'The new Vista task dialog presents an easy to use and user-friendly
replacement for messageboxes.',

   TD_YES + TD_NO, TD_ICON_QUESTION) = mrYes then
     TaskMessage(self,'yes');
```

With these basic functions, it is simple to make your Delphi applications already a little more Vista ready.

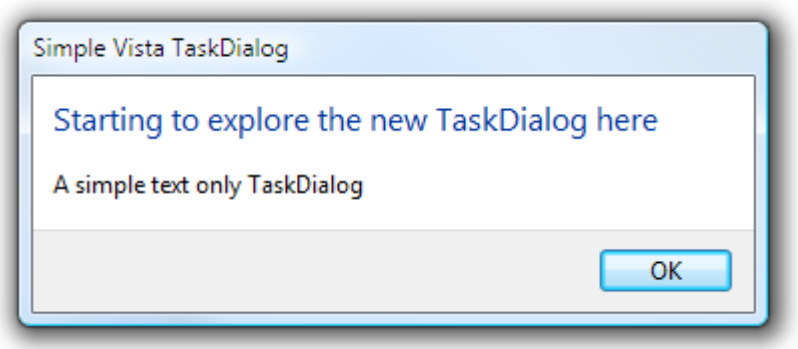
One step further

To get more out of the new TaskDialog, Vista exposes the API TaskDialogIndirect that offers a lot more capabilities than the simplified TaskDialog version.
Using TaskDialogIndirect is a little bit more complex but we have encapsulated all this in an easy to use Delphi component TTaskDialog with properties and events that expose the new functionality. The source code of the new TTaskDialog is available and those interested can study it. Source file and package have been built with and tested for Delphi 2006. This article will focus on explaining how the new features can be used with the component from Delphi.

The samples below can be used by dropping the TTaskDialog on the form, setting its properties and calling Execute.

Starting with the simple dialog This code snippet shows a dialog with title text, main instruction and content:

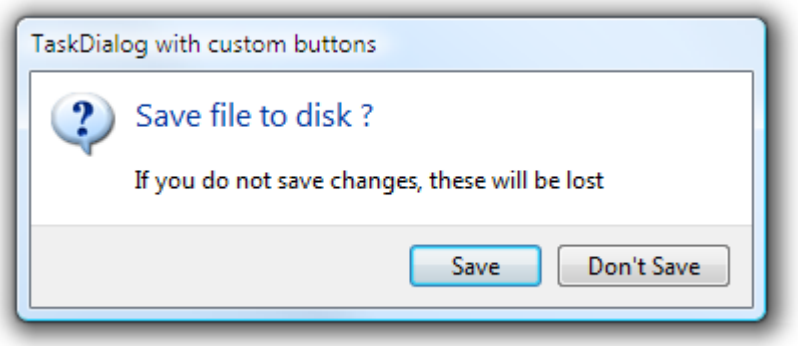
```
TaskDialog1.Title := 'Simple Vista TaskDialog';
TaskDialog1.Instruction := 'Starting to explore the new TaskDialog here';
TaskDialog1.Content := 'A simple text only TaskDialog';
TaskDialog1.CommonButtons := [cbOK];
TaskDialog1.Execute;
```



Dialog with custom buttons

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, TaskDialog.Execute returns the common Windows values for Ok, Cancel, Yes, No, ... the first custom button returns 100, the second 101, etc...

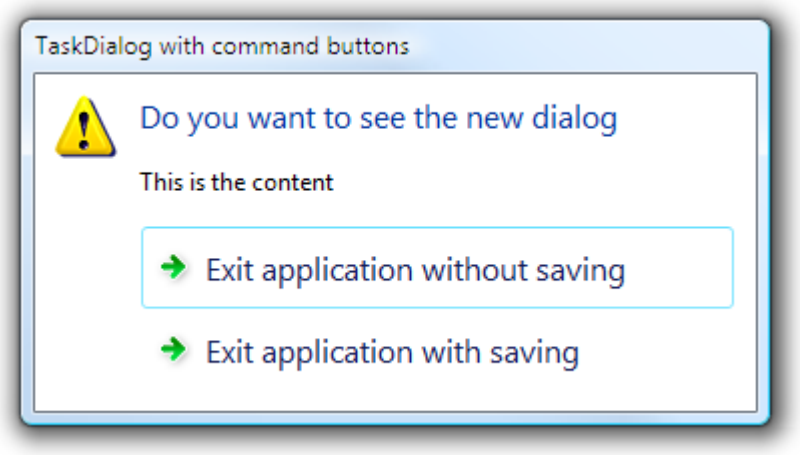
```
TaskDialog1.Title := 'TaskDialog
with custom buttons';
TaskDialog1.Icon := tiQuestion;
TaskDialog1.CustomButtons.Clear;
TaskDialog1.CustomButtons.Add('Save');
TaskDialog1.CustomButtons.Add('Don't Save');
TaskDialog1.DefaultButton := 101;
TaskDialog1.Instruction := 'Save file to disk ?';
TaskDialog1.Content := 'If you do not save changes, these will be lost';
ShowMessage(inttostr(TaskDialog1.Execute));
```



Dialog with CommandButtons

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 TaskDialog.Options:

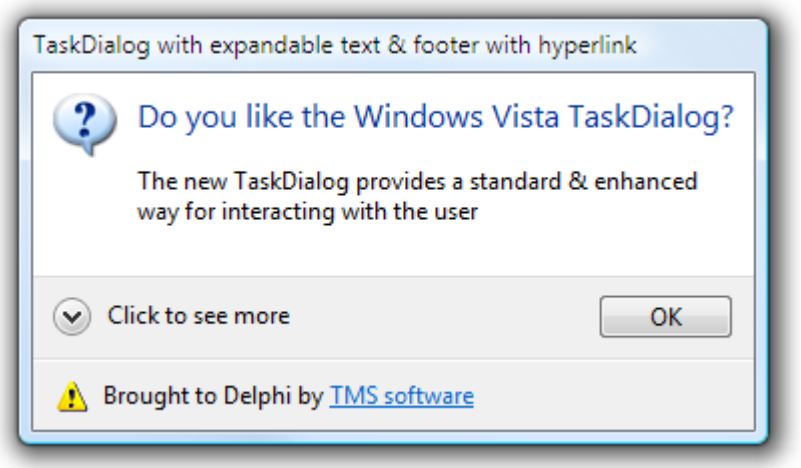
```
TaskDialog1.Title := 'TaskDialog with command buttons';
TaskDialog1.Icon := tiWarning;
TaskDialog1.CustomButtons.Clear;
TaskDialog1.CustomButtons.Add('Exit application without saving');
TaskDialog1.CustomButtons.Add('Exit application with saving');
TaskDialog1.DefaultButton := 100;
TaskDialog1.Options := [doCommandLinks];
TaskDialog1.Execute;
```

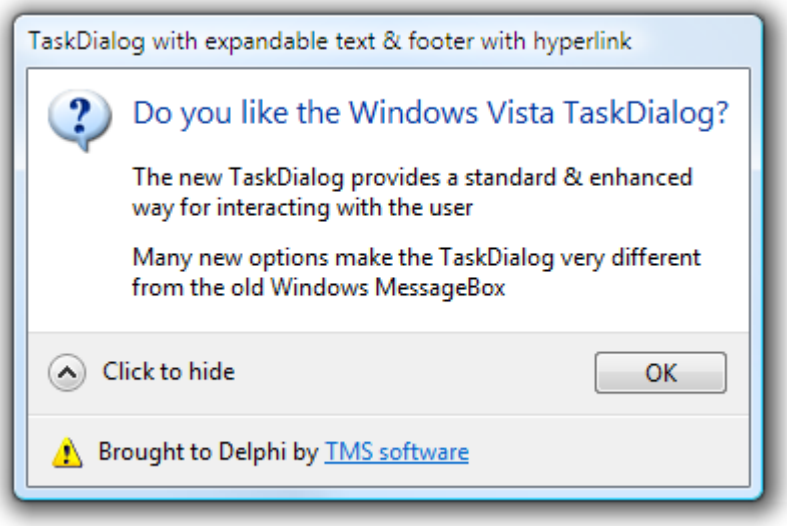


TaskDialog with expandable region, footer text and hyperlink

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 TaskDialog text, it is required to set doHyperlinks = true in TaskDialog.Options. When the hyperlink is clicked, the event OnDialogHyperlinkClick is triggered.

```
TaskDialog1.Options := [doHyperlinks];
TaskDialog1.Title := 'TaskDialog with expandable text & footer with hyperlink';
TaskDialog1.Instruction := 'Do you like the Windows Vista TaskDialog?';
TaskDialog1.Icon := tiQuestion;
TaskDialog1.Content := 'The new TaskDialog provides a standard & enhanced way
for interacting with the user';
TaskDialog1.ExpandedText := 'Many new options make the TaskDialog very different
from the old Windows MessageBox';
TaskDialog1.ExpandControlText := 'Click to hide';
TaskDialog1.CollapsControlText := 'Click to see more';
TaskDialog1.Footer := 'Brought to Delphi by <A
href="http://www.tmssoftware.com">TMS
software</A>';
TaskDialog1.FooterIcon := tfiWarning;
TaskDialog1.Execute;
```



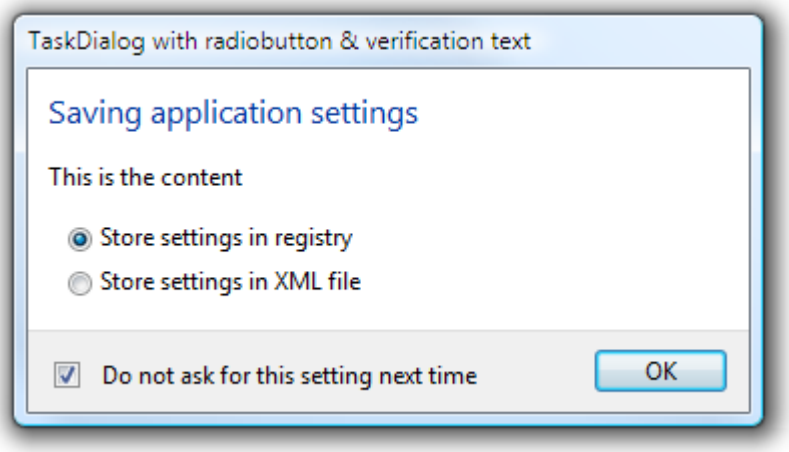


TaskDialog with RadioButtons and Verify checkbox

A TaskDialog 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 TaskDialog.RadioResult. The result for the first radiobutton is 200, the 2nd radiobutton 201, etc... The selection of the verification checkbox is returned by TaskDialog.VerifyResult.

```
TaskDialog1.Title := 'TaskDialog
with radiobutton & verification text';
TaskDialog1.RadioButtons.Clear;
TaskDialog1.RadioButtons.Add('Store settings in registry');
TaskDialog1.RadioButtons.Add('Store settings in XML file');
TaskDialog1.VerificationText := 'Do not ask for this setting next time';
TaskDialog1.Instruction := 'Saving application settings';
TaskDialog1.Execute;
case TaskDialog1.RadioButtonResult of
200: ShowMessage('Store in registry');
201: ShowMessage('Store in XML');
end;

if TaskDialog1.VerifyResult then
  ShowMessage('Do not ask for this setting next time');
```



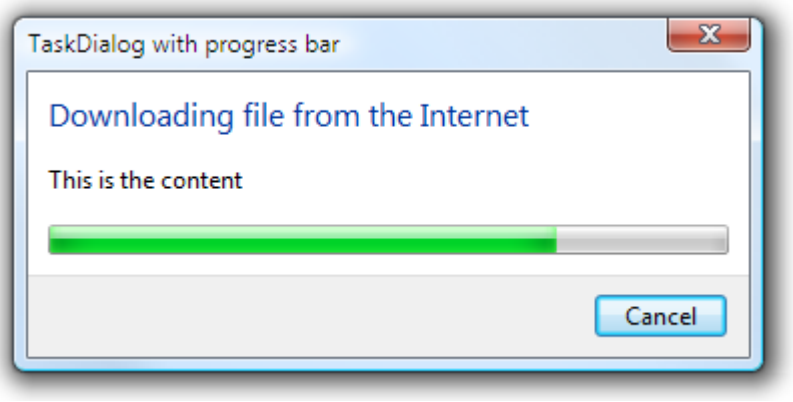
TaskDialog with ProgressBar

Finally, it is equally possible to use the new TaskDialog for progress indication. To enable a progressbar on the TaskDialog, set doProgressBar = true in TaskDialog.Options. From the event TaskDialog.OnDialogProgress, the position of the progressbar is queried. By default, the TaskDialog 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 TaskDialog with progressbar and a simple event that updates the progress. In addition, by invoking the TaskDialog.ClickButton() when progress is 100%, the code makes the dialog automatically disappear when the process is complete.

```
progresspos := 0;

TaskDialog1.Title := 'TaskDialog with progress bar';
TaskDialog1.Instruction := 'Downloading file from the Internet';
TaskDialog1.Options := [doProgressBar];
TaskDialog1.CommonButtons := [cbCancel];
TaskDialog1.OnDialogProgress := DoDialogProgress;
TaskDialog1.Execute;

procedure
TForm1.DoDialogProgress(Sender: TObject; var Pos: Integer; var State:
TTaskDialogProgressState);
begin
  if (progresspos < 100) then
  begin
    (Sender as TTaskDialog).EnableButton(1,false);
    inc(progresspos,2);
  end
  else
  begin
    (Sender as TTaskDialog).EnableButton(1,true);
    (Sender as TTaskDialog).ClickButton(1);
  end;
  Pos := progresspos;
  State := psNormal;
end;
```



Download the source code for TTaskDialog.