Contents

Information

xTools

xTools is part of my upcoming collection of smart components for Delphi. Each component is compiled and comes with a DCU, DCR and INT file. The component can be freely used in your application, but in the Delphi IDE I look for a valid registration code. The registration code must be entered in the DELPHI.INI under the [Stefc.xTool] section. If no valid registration exists, the component displays a small reminder.

An example of DELPHI.INI :

```
....
[Stefc.xTool]
TStopWatch=Mickey Mouse|1234
....
```

To register, please contact me via e-mail or write to me. If you are interested in the full source code for a component, please ask. I plan to make it avaiable for a small fee. The registration can be handled via CompuServe's Software Registration (GO SWREG Id=5763) or send a check to the following address. I've deceide to register each component seperatly because of the wide useness of them. No developer need all of the components together and so don't want to pay for stuff he not need. So I get the registration fee very small. Isn't it ?

Stefan Böther
Methfesselstr. 38
20257 Hamburg
= Germany =

CompuServe : 100023,275
InterNet : 100023.275@CompuServe.Com

Files

The complete xTool collection is distribute over various networks in one file called xTool.Zip. The prefered source for get the newest xTools is the Delphi forum on CompuServe. If you get xTool over an other network please contact me how actual is it. The version number can be seen at the first page of this documentation.

All the files needed for one component are dedicated to a single ZIP file within the xTOOL.ZIP file, so it is possible to split the single components. If someone do so please include all files in the main file into the component.

Each component contains a DCU (Delphi Compiled Unit), a DCR (Delphi Component Resource), a INT (Unit Interface Part) and sometimes a DFM (Delphi Form). Together with the component you'll find in some cases an small sample application consist of a xxx_PRJ.DPR, xxx_UNI.PAS and xxx_UNI.DFM. The sample isn't compiled because to hold the size of xTool.Zip very smart.

Here is are the list of all components (I hope to extend it fast because I have a lot of good ideas). If someone has ideas for me, or enhancements of existing components, please feel free to write to me.

TStopWatch	$10	$30
TMemTable	$10	$30
TDBRemember	$10	$50
TDBStatus	$10	$50
TDBFilter	$10	$50
TDBNextNo	$10	$50
TScreenSaver	$10	$50
TJoystick	$10	$40
TRemember	$10	$50
TTipTrickDialog	$10	$40
TFileCopy	$10	$30
TInstalment	$10	$30
TStatusBar	$10	$30

At this place I want to place some thanks to the following persons, who help me especially :

- Steve Garland For his help and sugestions at the BDE specifc topics.
- Andrew M. Stewart For his look on my English documentation and correct.

Components

TStopWatch component StpWatch

The TStopWatch component for Delphi is designed to help the developer measure elapsed time for a task (downloading, processing,etc.). You can control the type of measuring in various ways.

properties

`Display : Boolean`

If true, the text in "DisplayText" is shown automatically if you stop the TStopWatch component.

`DisplayText : String`

This is the text that would be shown if the TStopwatch is stopped and "Display" is True. It can contain a formating sequence to display the ellapsed time. For example, the strings "Elappsed time : %d minutes" and "Time gone %s" are valid. The formatting characters depend on the resolution. If DisplayText is empty, then a default formatting is used.

```
Started : Boolean
```

With this set to True you can start the stopwatch. The watch runs until you set this property back to False or directly call the Stop Method.

```
Paused : Boolean
```

If the stopwatch is started, you can pause it by setting this property to True. With False you can resume the watch.

```
Resolution: TResolution=(resMSeconds,resSeconds,resMinutes, resHours,
                  resTime);
```

With this property you can specify which resolution should be displayed in the default show method. The resolution has no effect on the measure itself.
For example :

```
Resolution := resTime;
DisplayText := 'Time elapsed %s';
```

displayed the elapsed time.

```
MSeconds: Longint
```

Returns number of milliseconds.

```
Seconds: Longint
```

Returns number of seconds

```
Minutes: Longint
```

Return number of minutes

```
Hours: Longint
```

Return number of hours

```
Time: TDateTime
```

Return elapsed time in TDateTime format

```
AsString : String
```

Return the actual ellapsed time as formatted string.

Methods

```
procedure Start;
```

Starts the stopwatch. Started:=True does the same.

```
procedure Stop;
```

Stops the stopwatch. Started:=False does the same.

```
procedure Pause;
```

Paused the stopwatch. Paused:=True does the same.

```
procedure Resume;
```

Resumes the stopwatch. Paused:=False does the same.

```
procedure Show;
```

Displays the result of the stopwatch. If Display is True this method is called on every Stop action.

Events

```
OnStart: TNotifyEvent
```

This event will be triggered when the watch starts

```
OnStop: TNotifyEvent
```

This event will be triggered when the watch stops

```
OnDisplay: TNotifyEvent
```

This event will be called when the result of the stopping should be displayed. If you do not handle it yourself, a default message dialog appears.

TMemTable MemTable

The TMemTable component is a descendant of TTable. With it you can create and use tables in memory. It does this task with a function of the BDE. The component itself adds no further properties or methods.

You must do the following tasks to use it in your programs.

- Drop it to your form from the component palette
- Assign a TableName to it
- Define some fields with the DataSet-Designer
  - Set Active to True

After this you have an empty, open table into which you can insert your data. You can connect a TDataSource to TMemTable and then use the data-aware components. Be sure you only use it for temporary storage because after closing TMemTable, all of its data is lost! To avoid that you should copy the data to a real TTable object.
The big advantage of TMemTable is speed because all of the tasks are done in memory, not in files. Also you don't need temporary files on your disk any longer. Use TMemTable and after you are done throw it away.
I don't know how much data you can put into a TMemTable at this time. Also some of the TTable functions do not work properly on memory tables. Indexing and Querying of TTable data don't work.
If you have ideas of how I can enhance this component, please
send me suggestions via e-mail.

TDbRemember DbRememb

The TDbRemember component for Delphi will save the width and position of a dataset component. In most cases you'll need this to save attributes of a TDbGrid persistently in a Windows ini-file. After restarting the program, it remembers things like width and the index of a field. All you must do is drop a TDbRemember component on to the form you want to store.

properties

```
DataSet : TDataSet
```

The dataset component which you want to save.

```
IniFile : String
```

Here you put in the name of the ini file where the information should be stored.

```
Section : String
```

This is the section in the inifile where the information will be stored. The string here is combined with additional information of the information type. For the item name inside the section, the name of the component is used.

```
Options: TDbRememberOptions= set of TDbRememberOption =
                        (roWidth,roIndex);
```

With this set you specify which information from the dataset you want to save into the ini-file.

- roIndex - stores the index of the corresponding fields
- roWidth - stores the display width of the fields

Methods

```
procedure Write; virtual;
procedure Read; virtual;
```

If you want to descend from TDbRemember, these methods may help you. You can plug in you own attributes that you simply write/read into the given IniFile. But don't forget to call the inherited method! You can access the IniFile with the FStorage object, which is active during the both methods.

Events

`OnWrite: TNotifyEvent`

After all the writing is done this event is called.

`OnRead: TNotifyEvent`

After all reading is done, this event is called by the component.

Tips & Tricks

Don't use a path in you inifile. Without path the *.INI is stored in the Windows directory. From this location your settings are unique to the station of the user. This works well in network environments.

TDbStatus DbStatus

DbStatus is a small data-aware control which visually informs the user of the current state of a datasource. This is done with a small bitmap on a panel, which changes whenever the linked datasource changes its state. It can also be linked to a TDbNavigator and control the visible buttons of the navigator depending on the datasource-state.

properties

`DataSource : TDataSource`

Here you set the datasource which you want to connect to. After setting the datasource informs the TDbStatus control of each state change, so the control can act.

It currently displays the following bitmaps :

- dsSetKey a zoom glass
- dsBrowse a book
- dsEdit pencil on a filled paper
- dsInsert pencil on a empty paper

`Navigator : TDbNavigator`

Here you can set a Navigator that you also want to control the visual appearance for when the state changes. For example, I do not need to see the disabled Post and Cancel Buttons in the Browse state. Here are the buttons which you can see in each state :

- dsSetKey first, next, prior, last
- dsBrowse first, next, prior, last, edit, delete, insert
- dsEdit,
  dsInsert post, cancel

It is not necessary to set this property.

Tips & Tricks

Please send me your own glyphs for the status. If I have enough together I plan to enhance this component to support various bitmaps.

TDbFilter DbFilter

DbFilter is a non-visual component that lets you filter a dataset. This is done with BDE filters.

Properties

`Handle : hDBIFilter`

This read-only property returns the handle of the BDE filter.

`DataSource : TDataSource`

Here you set the datasource you want to connect to. Every time you change the activation of the datasource the filter is reassigned.

`Priority : Word`

This is the priority that the filter has. The value varies from 1..n where 1 is the highest priority. The value is given directly to the BDE at the time the filter is created.

Events

`OnFilter : TFilterEvent`

This event is called each time the filter needs to know if the current records fits or not. You must return TRUE or FALSE as function results to indicate what should happen.

A sample event-handler looks like :

```
function TForm1.DbFilter1Filter(Sender: TObject; Dataset: TDataset):
Boolean;
begin
  Result := (fldCustType.Value = 'A') and
            (fldCustCredit.Value >= 10000.0);
end;
```

Don't do a lot of functions in the call because it can decrease your browsing performance. If you filter an area in a table, you are better off using a SetRange call. At this time related tables and calculated fields not supported. Sorry.

If you change the filter-function, always do a TTable.Refresh to ensure all data is re-filtered!

TDbNextNo DbNextNo

This component you can be used to increase number fields in datasources. For example you want to insert a new customer into your database. What customber-no he should get ? TDBNextNo connects to two datasources, one is the destination of the number the other is the source where to count the numbers. I myself often use a table called NEXTNUM which all numbers for the application contains. It contains the a key (for example "CustNo") and the last given number for that key. In multiuser environments it's also neccessary to control the number counting to grant unique numbers even if hundred of users work simultan.

Types

```
TNNIncType= (IncOnInsert,IncOnPost);

TNNSearchType = (stSearch,stNone);
```

Properties

`DataSource : TDataSource`

Here you connect to the destination datasource that the fields contains which you want to increase. (e.g. dsCustomer)

```
DataField : String
```

Here you specify the field in the DataSource that should contains the increased number. (e.g. fldCustomerNo).

... Information to come !!!! ....

TScreenSaver ScrSav

The TScreenSaver component helps you build Screen-Savers with Delphi. Although Delphi's VCL implants a large footprint on the resulting screen-saver, Delphi is ideal for easily building Screen-Savers.

How to build a Screen-Saver with this component:

- Create an main form and place the component in it
- Create the logic of the screen-saver into the form (in most cases a TTimer component which draws something onto the TForm.Canvas).
- Add {$D SCRNSAVE : DELPHI <name of saver> } into the
- delphi-project source.
- Compile the program to an *.exe, rename it to *.scr and copy it to the Windows directory.
  - Test it in the control-panel

properties

```
Desktop : TSaverDesktop= (sdBlack,sdTransparent)
```

Here you define how your the screensaver should act.

sdBlack	Maximize the form, give it a black color, and no border. Use it if you want to program a screensaver on a black ground.
sdTransparent	Use this if you want to program a screen-saver which acts directly on the desktop. On activation it automatically hides the containig form.

```
Saving : Boolean
```

With this read-only property you can check if the screen-saver is already started. It is set to true at beginning of screen-saving and is set to False shortly before closing.

```
Title : String
```

This is the title of the screen-saver. This string is used for the section in the ini-file and as title for the password dialog.

```
Password : String
```

With this property you set the password for the screen-saver. By reading this property you get the crypted password.

```
CtrlIni : TIniFile
```

With this IniFile object you can store your own screen-saver settings. Internally it is assigned to the control.ini file in the Windows directory.

```
CtrlSection: String
```

The complete name of the section where the settings of the screen-saver should be stored.

Methods

```
function CheckPassword(Value:String):Boolean;
```

The only method you can use so far is the method to check a password against the actual one.

Events

```
OnSetup: TNotifyEvent
```

This event occurs if the setup-dialog is called from the control-panel. The setup-form should not be auto-created. An event-handler might look like the following :

```
procedure TfrmSaver.ScreenSaver1Setup(Sender: Tobject);
begin
  Application.CreateForm(TfrmSetup, frmSetup);
end;
```

After execution of this dialog the program terminates automatically.

```
OnSave: TNotifyEvent
```

You can use this method to do something that must be done at the startup of the saver. I use this to create a TTimer descendant that acts on the TForm or Desktop-canvas.

```
OnPassword: TPasswordEvent
```

This property is triggered when a password is assigned to your screen-saver by setup. If you don't assigned it yourself the standard InputQuery is used.
The handler can look like this :

```
function TfrmSaver.ScreenSaver1Password(Sender: Tobject;
  var aPass: OpenString): Boolean;
begin
  Result := InputQuery(ScreenSaver1.Title,'Password:',aPass);
end;
```

Tips & Tricks

Use the following as the first line of the projet source to ensure that the screen-saver only one instance runs at a time.

```
if HPrevInst <> 0 then Exit;
```

Don't assign an event-handler to FormMouseMove,FormMouseDown or FormKeyDown, because they would be hooked by the screen-saver !

TJoystick JoyStick

The TJoystick component for Delphi will allow developers to access the joystick service of the Windows Multimedia System.

types

```
TJoystickId= JOYSTICKID1..JOYSTICKID2;
TJoystickButton= (jbNone, jbFirst, jbSecond, jbThird, jbFourth);
TJoystickButtons= set of Tjoystickbutton;
TJoystickButtonEvent= procedure(Sender: TObject; Button:
TJoystickButton) of object;
TJoystickMoveEvent= procedure(Sender:TObject; X,Y: Word) of object;
```

properties

```
Id : TJoystickId
```

The first joystick has Id 0; the second is Id 1.

```
Interval : Word
```

Here you define the interval of time the the joystick service should wait to call move and buttonpush events. It is measured in milliseconds.

```
Changed : Boolean
```

If this property is set to true, events are only produced if the value changed its state. For example, a move event is only triggered if you move the stick.

```
Enabled : Boolean
```

Enable/Disable the joystick.

```
Threshold : Word
```

The threshold of the joystick position. Only if a position change is over the threshold value an event is produced.

```
X,Y,Z : Word
```

Reads the current value of the joystick position. The range can be from 0 to 65000. The Z-position is not available for two-axis joysticks !

```
Buttons: TJoystickButtons
```

This set contains all actual pressed buttons.

Events

```
OnMove: TJoystickMoveEvent
```

This event occurs at each movement of the stick. The new X and Y positions is also updated.

```
OnButtonDown: TJoystickButtonEvent
```

This event occurs if a button is pressed. The button parameter shows which buttons was pressed.

```
OnButtonUp: TJoystickButtonEvent
```

This event occurs if some button is released if it previously was pressed. The button parameter shows which buttons was released. Note that auto-fire produces a lot of ButtonDown and ButtonUp events.

TRemember Remember

The TRemember component for Delphi saves various attributes of a form in a Windows ini-file. After restarting the program, it remembers things like positions, sizes, colors, and fonts. All you must do is drop a TRemember component on to the form you want to store.

properties

```
IniFile : String
```

Here you put in the name of the ini file where the information should be stored.

```
Section : String
```

This is the section in the ini-file where the information is stored. The string here is combined with additional information. For example, the position and size are stored in the section [Browser;Placement;640x480] where "Browser" is the value of the section property. The resolution of the screen is stored to let the user work with various screen-environments. For the item name inside the section, the name of the component is used.

```
Options : TRememberOptions= Set of TRememberOption =
        (roPosition,roSize,roColor,roFont);
```

With this set you specify which information from the form you want to save into the inifile.

- roPosition stores the position of the form
- roSize stores the size of the form
- roColor stores the color of the form
- roFont stores the font of the form

Methods

```
procedure Write; virtual;
procedure Read; virtual;
```

If you want to descend from TRemember these methods may help you. You can plug in you own attributes which you simply read/write into the IniFile. But don't forget to call the inherited method! You can access the IniFile with the FStorage object, which is active during the both methods.

Events

```
OnWrite: TNotifyEvent
```

After all the writing is done this event is called.

```
OnRead: TNotifyEvent
```

After all read is done this event is called by the component.

Tips & Tricks

The roColor and roFont styles make sense if you use the ParentFont and ParentColor properties in your controls. You can inherit the form color and font within the form.

In most cases you want to store font & color common to all your forms in the application. To do this independent of size and position of the form, place two TRemember components into your forms. One for the placement (with different names) and one for the color and fonts (with the same names).

Don't use a path in you inifile. Without a path the *.INI is stored in the windows directory. From this location, your settings are unique to the station of the user. This works well in a network environments.

TTipTrickDialog TipTrick

This component is a dialog that shows a Tips&Tricks dialog at your applications startup. Each time a new hint is shown in the dialog. Together with the tips&tricks you can enable buttons to start a tour, a registration task or a news-helpfile. I've look at the Windows 95 Welcome Dialog for this.

properties

```
Tips : TStrings
```

This is the stringlist where all tips are stored. Each line in the list is one tip. Each tip is limited to 255 characters.

```
Title : String
```

Here you say which title should appear at the top of the dialog.

```
Options : TTipTrickOptions = set of TTipTrickOption =
(ttTour,ttNews,ttRegister);
```

With this set you specify which buttons should be visible at runtime in you dialog. Be sure that you only display the buttons that you've coded.

- ttTour If you features a guided tour you should enable this option
- ttNews If you want to display some news regards your application use this option
- ttRegister If you want some type of online registration enable this option

```
AlwaysShow : Boolean
```

This property is identically to the checkbox in the dialog. You must save and load this value yourself after and before the dialogs should startup. Probably this is done with an item in a ini-file.

Methods

```
function Execute: Boolean;
```

With this method you start the dialog. You should use it in your FormCreate method of the mainform but only if the user wants to see it each time. After calling it you should store the AlwaysShow property somewhere. The result of this function have no meaning.

Events

```
OnTour: TNotifyEvent
```

This is called if the user click the Tour-Button if aviable.

```
OnRegister: TNotifyEvent
```

This is called if the user click the Register-Button if aviable.

```
OnNews: TNotifyEvent
```

This is called if the user click the News-Button if aviable.

Tips & Tricks

If you need the component for a language other than English you must open the TIPTRICK.DFM in Delphi alone and edit it in ASCII Mode. You'll see where to place the language specific strings. This woulld be the form-caption, the buttons and the labels. If you customize this form specific four your country it would be nice if you can send it to me back. Then I can integrate it into the xTool-Collection.

Display the dialog also in the help-menu of your application mainmenu as WinWord does it.

TInstalment Instalment

In a single project I need methods for calculate instalments for various time frames. The idea to bring this into a component was born. This component isn't very big but do what it should do.

Types

```
TInstalmentFrame=(ifWeek,ifTwoWeek,ifMonth,ifTwoMonth,
        ifQuart,ifHalfYear,ifYear);
```

This type specify the instalment type. If you need more than I please write me which you need and how it should be named (ifXXXX) and I'll include it in a future version of Tinstalment.

properties

```
Input,Output: Extended;
```

One of this propertys you'll fill with a value, in most cases you should use the Input property but Output works too. After put one value in you can read the result from the other property. This is all you need for do the convert.

```
InpFrame,OutFrame : TInstalmentFrame
```

Here you say which type of instalment correspond with the Input & Output values.

Sample

The following source shows how to do the convert for a yearly ammount into a monthly instalment.

```
var
     aInstal: Tinstalment;
begin
  aInstal:=TInstalment.Create(nil);
  with aInstal do begin
    InpFrame:=ifYear;
    OutFrame:=ifMonth;
    Input:=1000.00;
    writeln('Monthly ammount :',Output:10:2);
    Free;
  end;
end;
```

TFileCopy FileCopy

This small component have the purpose to helps you to easy copy files from one source to a target. Also it can informs the program about the stage of the copy process with an event.

properties

```
SourceFile, TargetFile: String;
```

Here you place in the filename of the source file and the one of the target file.

```
Percent : Integer;
```

If you connect the component to a gauge or similar display you can use this readonly property to update the control. During the filecopy process this value is updated.

Events

```
OnNotify: TNotifyEvent
```

This event can be used to visualize the value of Percent during the copy process.

Methods

```
function Execute: Boolean;
```

With this method you start the copy process. It returns True if the copy is ok.

Sample

There is a small sample project included that use this component to copy the EXE of itself to a *.OLD one. The percentage is shown by a TLabel.

TStatusBar StatBar

This descendant from TPanel shows the hint of the focused control of the form. All you need is to place it into your form and you get a status line Bottomaligned and with the right border bevels. The component at this time have no own properts or methods and inheriteds the rest from TPanel.