Programming with Delphi

Debugging

Debugging techniques in Delphi & bug prevention

Stefan Cruysberghs
www.scip.be
June 2003

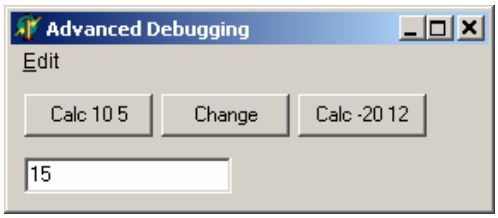
Contents

Debugging

Even if you did write a program in disciplined, well-structured, careful manner, you probably still need to debug it to find some bugs. Delphi offers great debugging tools and when you know how to use them, you will surely save a lot of time finding the exact reason of the problem.

In this small tutorial I will explain some nice features in Delphi which will help you to debug your programs. Because it is better to avoid bugs, I will also give some tips for making better programs.

Try it with an example




```
unit DebugTechn;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    MainMenu1: TMainMenu;
    Edit2: TMenuItem;
    MenuCalculate1: TMenuItem;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure MenuCalculate1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    procedure Calculate(const Int1, Int2 : Integer);
  public
  end;

var
  Form1: TForm1;
  IntNumber1, IntNumber2 : Integer;
```

```
implementation

{$R *.DFM}

procedure TForm1.Calculate(const Int1, Int2 : Integer);
var
  IntSum : Integer;
  BlnPositive : Boolean;
begin
  IntNumber1 := Int1;
  IntNumber2 := Int2;
  IntSum := IntNumber1 + IntNumber2;
  BlnPositive := (IntSum >= 0);

  OutputDebugString(PChar('Sum = '+IntToStr(IntSum)));

  Edit1.Text := IntToStr(IntSum);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Calculate(10,5);
  OutputDebugString(PChar('Calculate, 10 and 5'));
end;

procedure TForm1.MenuCalculate1Click(Sender: TObject);
begin
  Button1Click(Sender);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  IntNumber1 := 12;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Calculate(-20,12);
end;
```

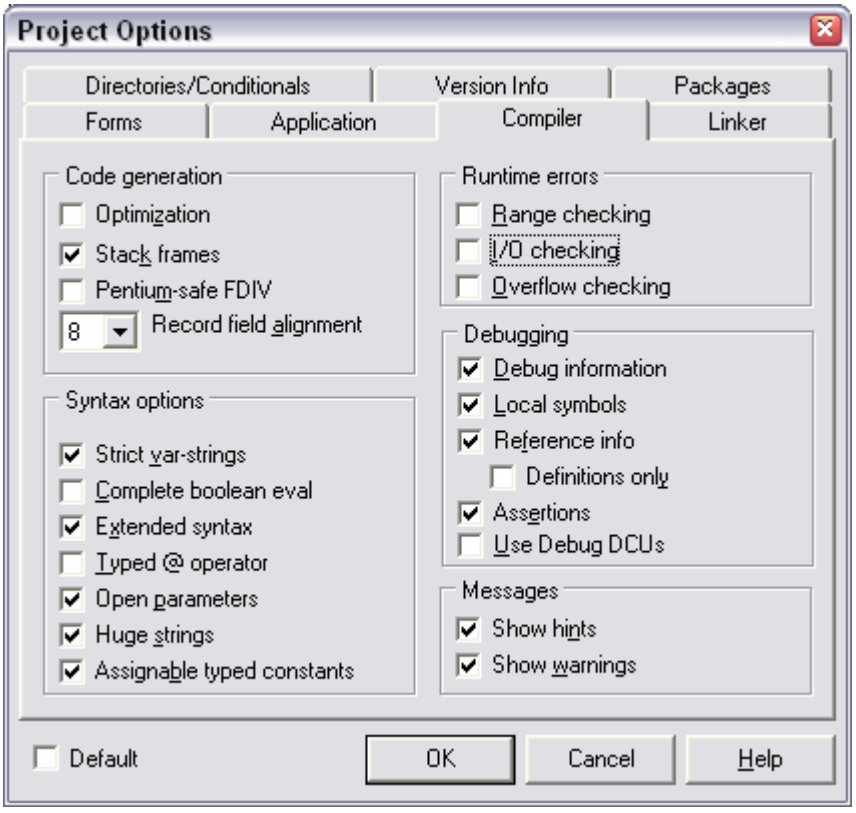
Options

Before you can start using the Delphi debugger tools, you have to make sure all necessary settings are set.

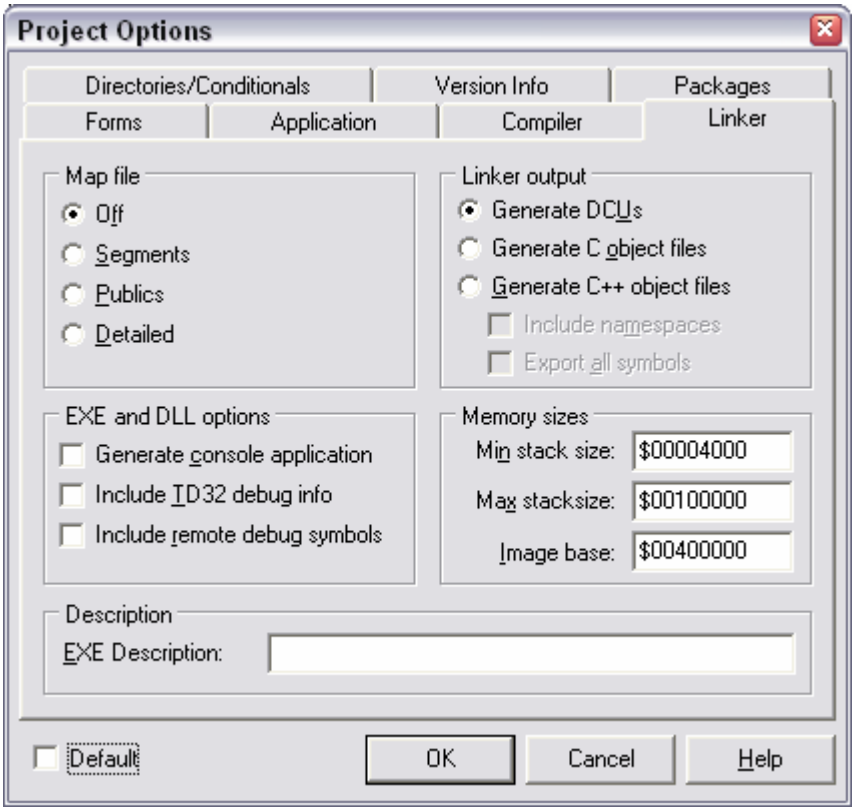
Project options

Go to the tab **Compiler** in the project options via the menu **Project >> Options**. Uncheck **Optimization**, check **Stack frames** and check some of the **Debugging** options. Without these options you will not be able to use all the debugging tools.

When creating an official build of your application it is better to uncheck all **Debugging** options and to check **Optimization.** Your final EXE will be a little bit smaller without this debug information.

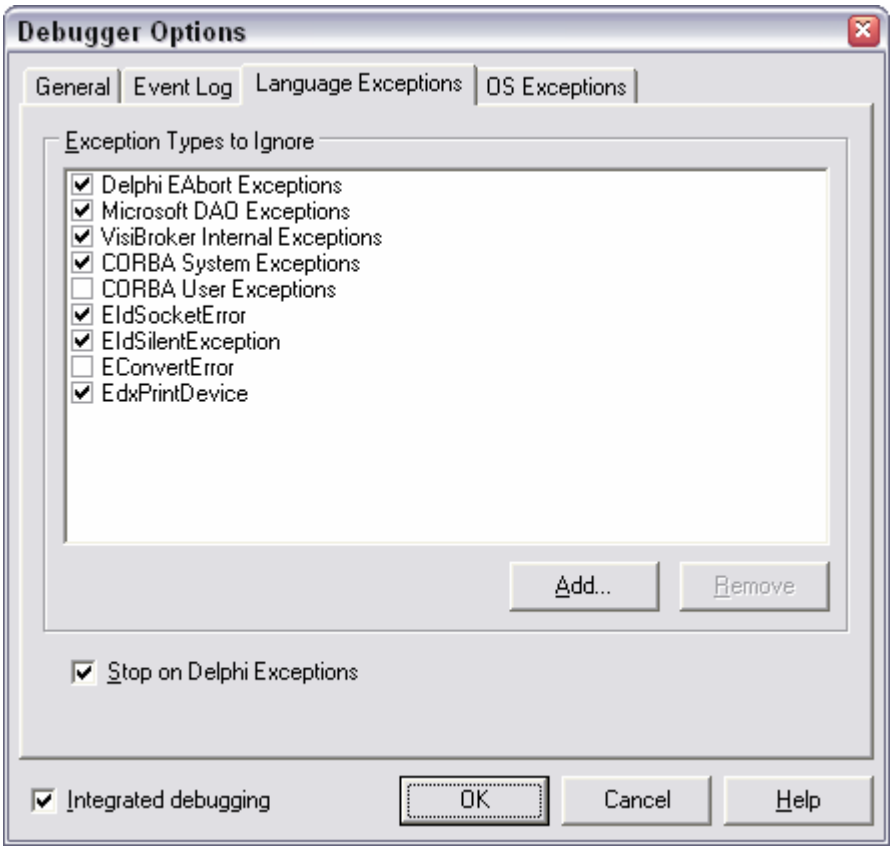


Just use the default **Linker** options. When using external debuggers like MemProof you sometimes have to set the **Map file** to **Detailed** and check **Include TD32 debug info**. In all other cases don't check the **EXE and DLL options** and don't use the **Map file**.



Debugger options

To use the debugger tools you have the check if the option **Integrated debugging** (**Tools >> Debugger options**) is checked.



Breakpoints

When pressing the F5 button or clicking on the left bar in your editor you can add a red line to your source. This line of source will have a breakpoint. When running the program, the execution will stop when it passes the source line. Now you can trace into your source by using some function keys.

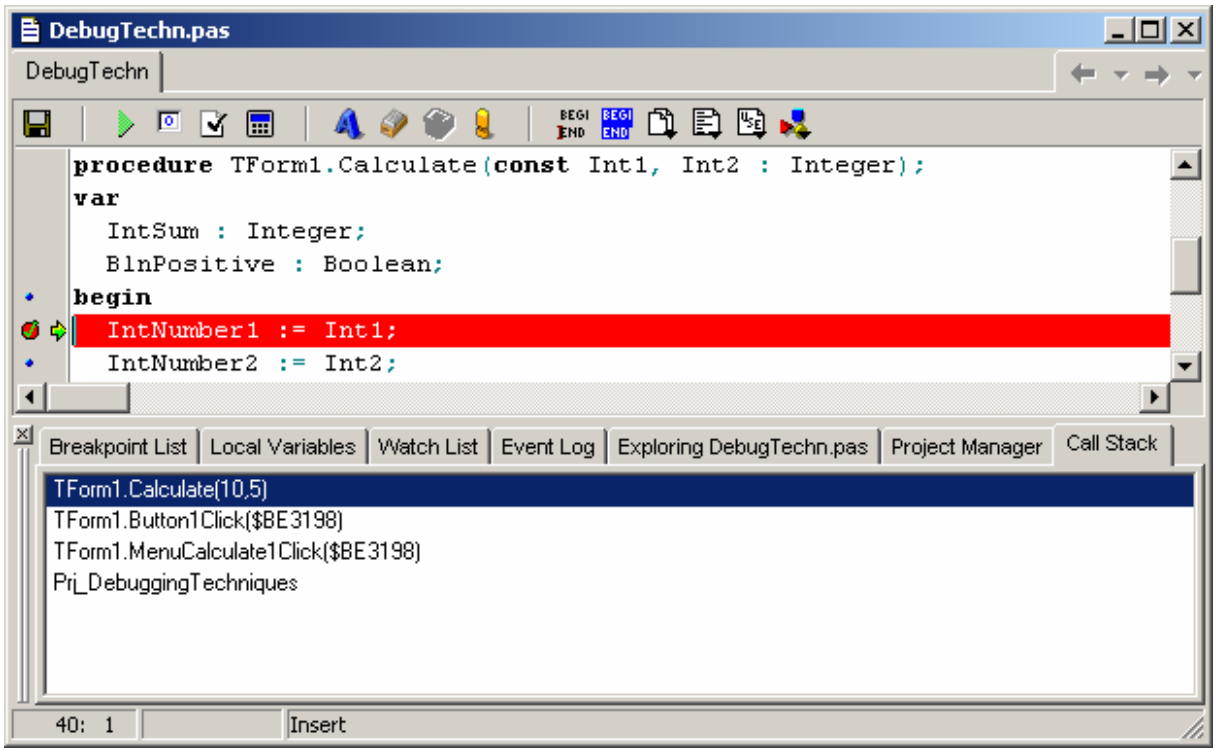
F9 Run/Start
F8 Step over to next source line
F7 Trace into (source of function/procedure)
F4 Run to cursor (and skip the other source lines)
F5 Add/remove break

Call stack

The **Call Stack** window displays the function calls (=LIFO list) that brought you to your current program location and the arguments passed to each function call.

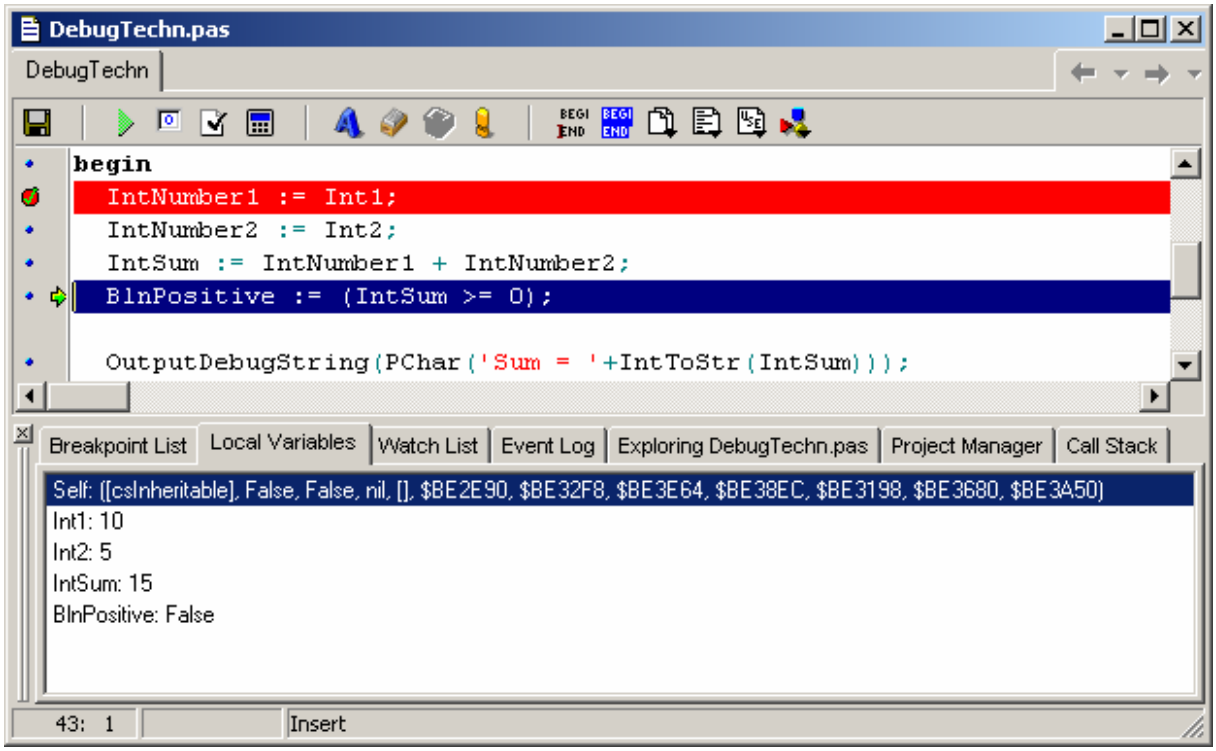
The top of the Call Stack window lists the last function called by your program. Below this is the listing for the previously called function. The listing continues, with the first function called in your program located at the bottom of the list. If debug information is available for a function listed in the window, it is followed by the arguments that were passed when the call was made.

The Call Stack window also shows the names of member functions (or methods). When right clicking you can jump to the selected procedure/function/…



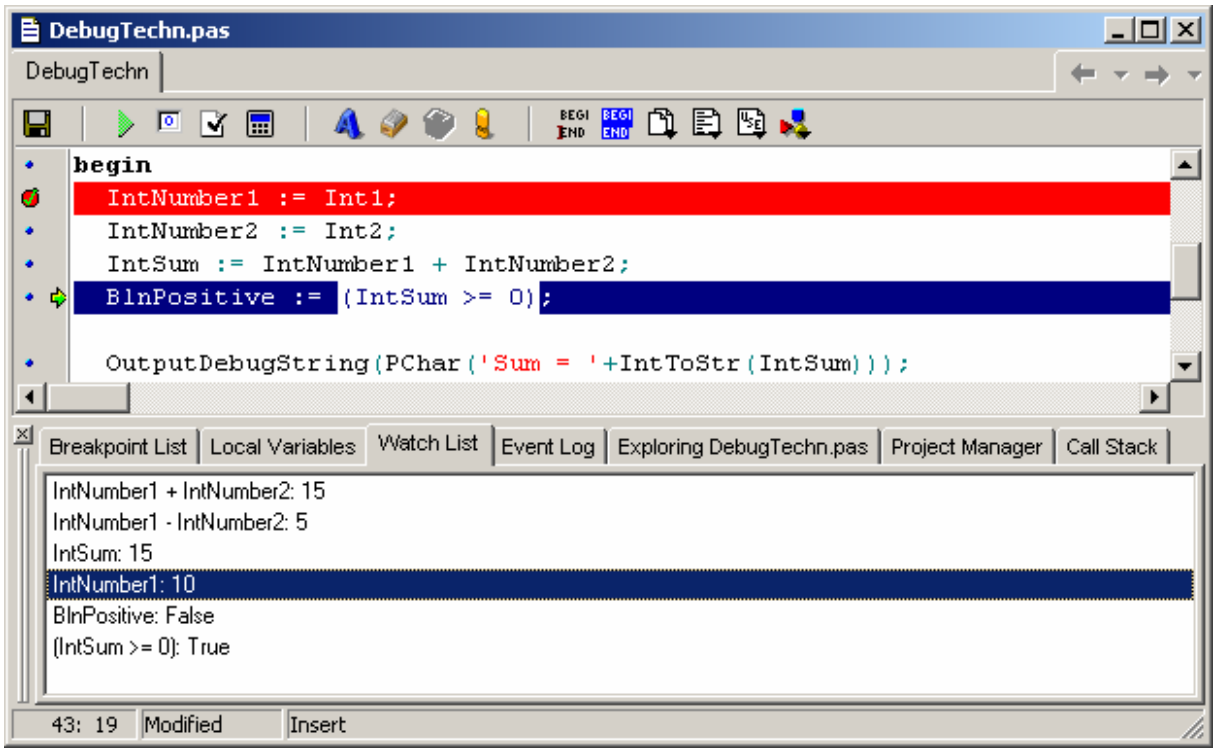
Local variables

This window will show all local variables and their current value in the current function or procedure.



Watches

Right click and choose **Add watch** or drag an expression from the editor to the **Watch List** window.



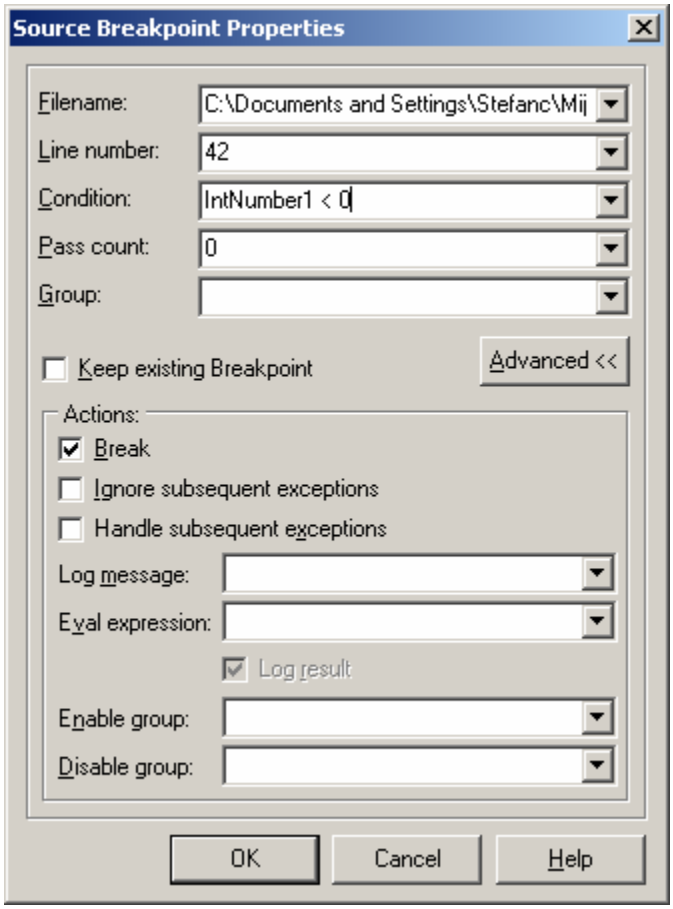
Options

- **Expression** : Specifies the expression to watch. Enter or edit the expression you want to watch. Use the drop-down button to choose from a history of previously selected expressions.
- **Repeat count** : Specifies the repeat count when the watch expression represents a data element, or specifies the number of elements in an array when the watch expression represents an array.
- **Digits** : Specifies the number of significant digits in a watch value that is a floating-point expression. Enter the number of digits.

Breakpoints

Source breakpoints

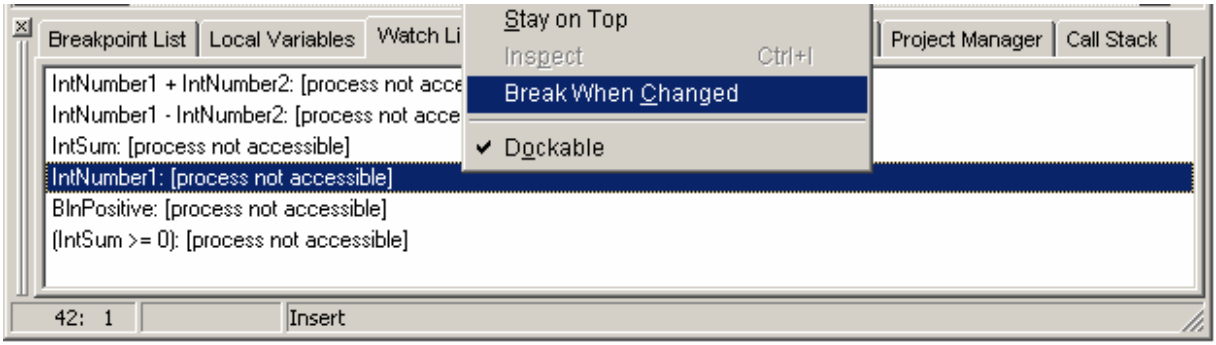
Conditions, Pass count, Log message



- **Condition :** Specifies a conditional expression that is evaluated each time the breakpoint is encountered. Program execution stops when the expression evaluates to True. Enter a conditional expression to stop program execution. You can enter any valid language expression. However, all symbols in the expression must be accessible from the breakpoint's location.
- **Pass count :** Stops program execution at a certain line number after a specified number of passes. Enter the number of passes. When you use pass counts with conditions, program execution pauses the nth time that the conditional expression is true. the debugger decrements the pass count only when the conditional expression is true.
- **Log message** : Writes the specified message in the event log.
- **Eval expression** : Evaluates the specified expression and because Log result is checked by default writes the result of the evaluation to the event log. Uncheck Log result to evaluate without logging.

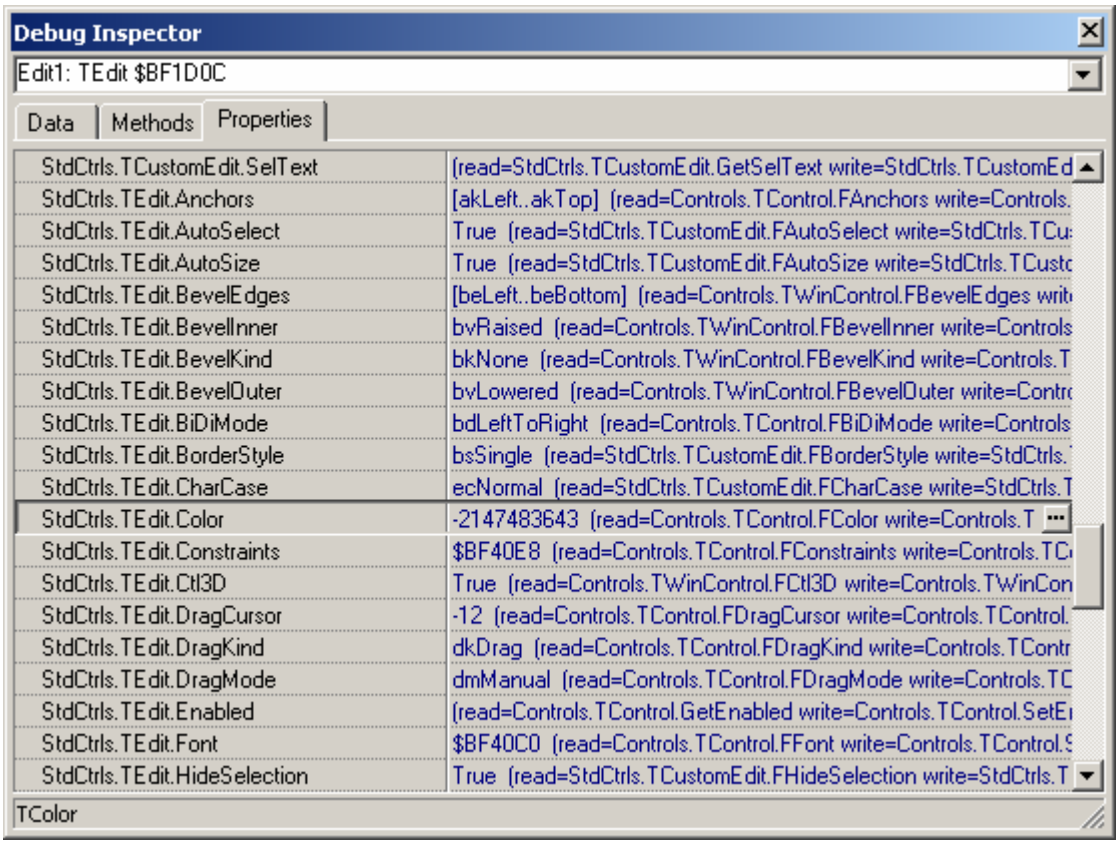
Data breakpoints

Right click on a **watch** in run-time (or add a new watch) and choose the menu **Break When Changed.** Every time the contents of a watch changes, the execution process will be stop on the source line were on of the elements of the expression has changed.



Debug inspector

- CTRL + click on object
- Inspect in popupmenu of watch



Event log (OutputDebugString)

The Event log window shows all information about

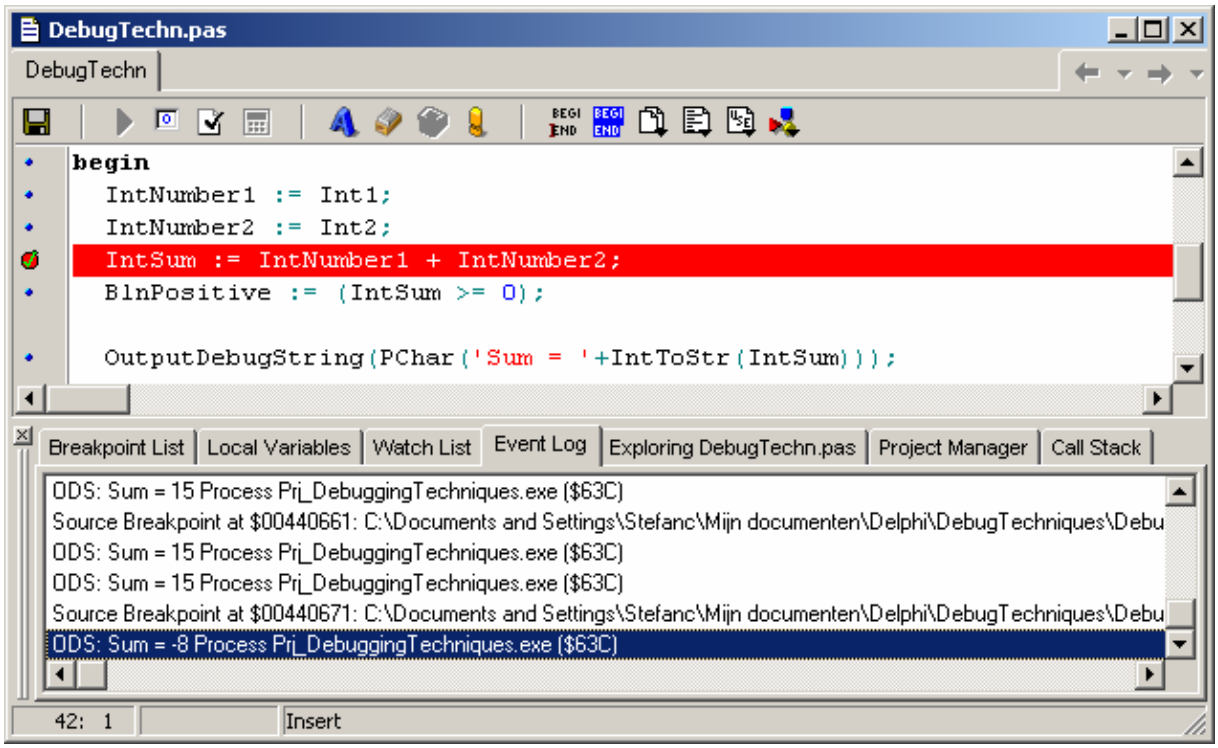
- Loaded modules
- Processed breakpoints
- Breakpoint messages
- Developer messages (OutputDebugString, displayed as ODS in log)

The popupmenu can be used to clear the log or save it to file.

The API function **OutputDebugString** sends a string to the debugger for the current application.

```
VOID OutputDebugString(
    LPCTSTR lpOutputString    // pointer to string to be displayed
   );
```

The parameter lpOutputString points to the null-terminated string to be displayed.



Ideas for creating your own debug features

- Make a debug menu in your mainform which can be activated by pressing a key combination. In this menu you can add all kinds of submenus which will show special debug forms.
- Make a form which shows the version of the application, the available system memory, the windows version, the current user name, … You can only start to reproduce a problem when you have all necessary system information. The most important information can also be shown in an aboutbox so that every user can see this information.
- Make a function ShowDataset which will open a form with a grid, which shows some field information (name, datatype, …), the SQL statement of a query, … This function can be used to show the contents of a dataset in situations when you are not sure the data is correct.

Bug prevention

Solve warnings and hints

Solve all hints and warnings. The warnings marked by Delphi can really cause some errors in run-time.

Items of lists

Never access items of a list without beeing sure that the item really exists.

```
if objText.Count > 10 then
  strText := objText.Strings[10];
```

Pointers to objects

Use the Assign function to check if an object is still assigned to a pointer. When freeing the object, also make sure the pointer is set to nil.

```
if Assigned(objText) then
  intCount := objText.Count;

FreeAndNil(objText);
```

Try-Finally

Always use a try-finally structure where you have to be sure some actions are executed to finish what has been started.

- **DisableControls & EnableControls** : Use the DisableControls of a Dataset to prevent flickering and to make dataset actions a lot faster.
- **Create & Free**
- **Screen.Cursor:=crHourglass & Screen.Cursor:=crDefault**
- **GetBookmark & FreeBookmark**

```
Screen.Cursor := crHourGlass;
try
  …
finally
  Screen.Cursor := crDefault;
end;
```

```
objStrings := TStringList.Create;
try
  …
finally
  objStrings.Free;
end;
```

```
ptrBookmark := GetBookmark;
try
  …
  GotoBookmark(ptrBookmark);
finally
  FreeBookmark(ptrBookmark);
end;
```

Try-Except

Make sure the option **Stop on Delphi exceptions** in the **Language exceptions** of the **Debugger options** is checked.

Use try-except structure for all actions which can go wrong. Make sure the exception is only called for exceptional cases. An exception is very slow so it is a lot better to add some if conditions if the exception does occur frequently.

Use the StrToIntDef and StrToDateDef functions to give a default in case the conversion goes wrong.

Use the raise statement when nesting try-except structures.

Examples with there durations

Try-Except, 100 000 repeats ➔ 28 seconds, so 100 000 000 ➔ more then 7 hours

```
procedure TForm1.adtBitBtn1Click(Sender: TObject);
var
  i : integer;
  t : tdatetime;
  a : string;
  b : integer;
begin
  t := now();
  a := 'A';
  for i:=0 to 100000 do
  begin
    try
      b := strtoint(a);
    except
    end;
  end;
  adttextedit1.Text := TimeToStr(Now()-t);
end;
```

Try-Except, 100 000 000 repeats ➔ 1 second

```
procedure TForm1.adtBitBtn2Click(Sender: TObject);
var
  i : integer;
  t : tdatetime;
  a : string;
  b : integer;
begin
  t := now();
  a := 'A';
  for i:=0 to 100000000 do
  begin
    if a <> 'A' then
    begin
      try
        b := strtoint(a);
      except
      end;
    end
    else
      b := 0;
  end;
  adttextedit2.Text := TimeToStr(Now()-t);
end;
```

Try-Except, 100 000 000 repeats ➔ 4 seconds

```
procedure TForm1.adtBitBtn3Click(Sender: TObject);
var
  i : integer;
  t : tdatetime;
  a : string;
  b : integer;
begin
  t := now();
  a := 'A';
  for i:=0 to 100000000 do
  begin
    b := strtointdef(a,0);
  end;
  adttextedit3.Text := TimeToStr(Now()-t);
end;
```