



# DelphiDay

italian conference

## The Art of Debugging

EXTENDED  
EDITION

Do you really know how to debug?



# Primož Gabrijelčič



[thedelphigeek.com](https://thedelphigeek.com)



[gabr42@gmail.com](mailto:gabr42@gmail.com)



[gabr42](#)



[github.com/gabr42](https://github.com/gabr42)



[linkedin.com/in/gabr42](https://linkedin.com/in/gabr42)



**DelphiDay**  
italian conference

19-20 Giugno 2025  
Piacenza



**wintech**  
italia



# OPEN-SOURCE PROJECTS

[github.com/account](https://github.com/gabr42/account)

## OmniThreadLibrary

[github.com/gabr42/OmniThreadLibrary](https://github.com/gabr42/OmniThreadLibrary)

## GpDelphiUnits

[github.com/gabr42/GpDelphiUnits](https://github.com/gabr42/GpDelphiUnits)

## Chatterbox

[github.com/gabr42/Chatterbox](https://github.com/gabr42/Chatterbox)



19-20 Giugno 2025  
Piacenza



# The Delphi Geek

random ramblings on Delphi, programming, Delphi programming, and all the rest

Friday, April 04, 2025

## Run your Delphi programs in a browser

Next Wednesday I'll be talking about Delphi and TMS Web Core in Ljubljana. As the presentation will be in Slovenian language, so is the rest of this post.

[Read more »](#)

Posted by [gabr42](#) at [09:02](#) 1 comment

Labels: [Delphi](#), [presentations](#), [web](#)

Sunday, February 02, 2025

## Delphi and AI [7]: How good are local DeepSeek models (for Delphi)

Due to potential privacy concerns with DeepSeek servers (we're unsure if the data sent over the paid API is kept private), I looked into some smaller DeepSeek models available on the Ollama.com site. These models use less complex AI with fewer parameters than the online version, but they might still be good enough for an average Delphi programmer. We'll see.

For testing, I used a powerful RTX 4090 card with 24 GB of memory. If your graphics card has less memory, your selection of useful models will be more limited.

I asked all models the same two questions: one on a general programming topic and another specific to the FireMonkey platform. The first question was:

"I have a multiline string containing newline ASCII characters (TMemo.Text). I want to change it to a single-line string with only printable ASCII characters. I could do that with BASE64 encoding, for example. I would, however, like to keep the text as much readable as possible by "encoding" only non-printable characters. Is there a simple way to do that?"

You can check Codellama's response in an older post (Codellama being the only local model I had tested so far): [Delphi and AI \[5\]: Encoding Multi-line Strings](#).

Last week, I asked the same question to the online DeepSeek-Reasoning model. Check the answers in this post: [Delphi and AI \[6\]: DeepSeek-Reasoning Model](#).

The second question was:

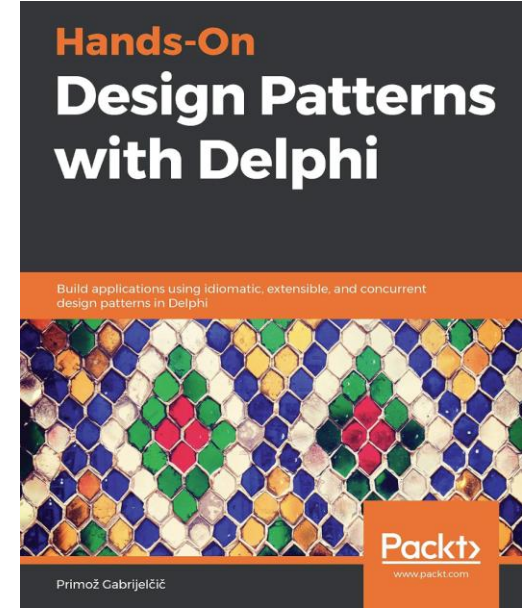
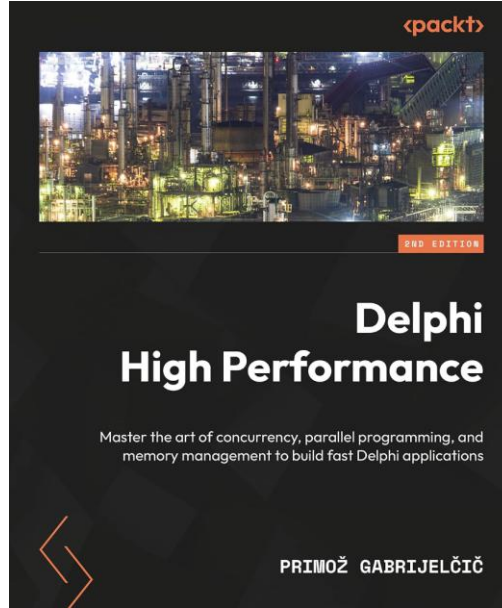
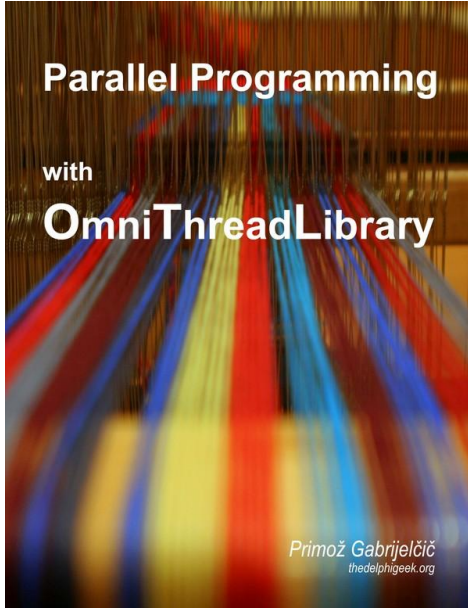
"How can I copy text to clipboard in a Delphi Firemonkey application?"

Embarcadero  
MV

Pages

[Presentations](#)





<https://delphi-books.com/>



# AGENDA

---

1. Strategies and pitfalls
2. Tooling
3. Tips & tricks



# Strategies

1



---

“Debugging is a **methodical** process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.”

--unknown



**Be methodical!**



# Debugging loop

---

1. Make sure you can repeat the problem!
2. Establish a hypothesis
3. Gather data
4. Implement potential fix
5. Test
6. If not fixed, go to 2



# Debugging loop

---

## 1. Make sure you can repeat the problem!

- If a problem appeared elsewhere, you may need a correct configuration
- Or a correct version of the code from the repository
- Or it could be a random problem that is hard to repeat

## 2. Establish a hypothesis

## 3. Gather data

## 4. Implement potential fix

## 5. Test

## 6. If not fixed, go to 2



# Repeat the problem!

---

- Bugs can depend on software configuration ...
- ... Or on a specific input data
- ... Or on specific version of your code
  - If you're lucky, the bug was already fixed
- ... Or on an OS version
- ... Or on a specific hardware
- ... Or, especially in multithreaded code, on random chance



# Repeat the problem (there's more)

---

- If the problem is hard to reproduce, try
  - Stress-testing
    - Slow CPU, multiple clients
    - Randomly pause program in debugger and resume after short time
  - Reproduce exact input data from the customer's side
    - Create special version that records all the data there
    - Reproduce on testing computer
  - Test on multiple computers
  - Debug directly on problematic installation (Remote Debugger)





# Debugging loop

---

1. Make sure you can repeat the problem!

**2. Establish a hypothesis**

- You can start with “I have absolute no idea what is going on”
- Or with “I know what the problem is!”

3. Gather data

4. Implement potential fix

5. Test

6. If not fixed, go to 2



# Establish a hypothesis

---

- If you have a rough idea on what could be going wrong, try that first 😊
- Don't focus on a single idea too much as it may be completely wrong!
  - Frequently it **will** be completely wrong
- Test multiple hypotheses in parallel
  - Try to **confirm** them
  - Try to **reject** them



# Debugging loop

---

1. Make sure you can repeat the problem!
2. Establish a hypothesis
- 3. Gather data**
  - Step through code, use breakpoints, inspect data
  - Logging
4. Implement potential fix
5. Test
6. If not fixed, go to 2



# Gather data

---

- Pause the program (Breakpoints)
- Inspect variables (Evaluate, Watch)
- Single step through the problematic code
  - Again, you may not be looking at the real reason for the problem!
- Repeat



# Automate!

---

- Use logging
  - Especially important when debugging multithreaded problems
  - Pause + step + evaluate may change program behaviour
- `OutputDebugString`, `GpStuff.OutputDebugString`
- Console output, `GpConsole`
  - Fast!
- Any logging suite
  - LoggerPro, CodeSite, Log4Delphi, Logify
- Capture exception stack on problematic computer
  - MadExcept, EurekaLog, JclDebug





# Debugging loop

---

1. Make sure you can repeat the problem!
2. Establish a hypothesis
3. Gather data
- 4. Implement potential fix**
  - You cannot **guarantee** it will work, you can only assume
5. Test
6. If not fixed, go to 2



# Debugging loop

---

1. Make sure you can repeat the problem!
2. Establish a hypothesis
3. Gather data
4. Implement potential fix
- 5. Test**
  - See item 1
6. If not fixed, go to 2



# Test

---

- Testing = failure to repeat a problem
  - Testing  $\nleftrightarrow$  guarantee that the code works
- This is a problem especially in multithreaded code
  - Where problem may just “hide away” because of the fix
- You'll maybe have to revisit the problem at the later stage
- When documenting (commit log) always state what you **believe** was the cause of the problem



# Debugging loop

---

1. Make sure you can repeat the problem!
2. Establish a hypothesis
3. Gather data
4. Implement potential fix
5. Test
- 6. If not fixed, go to 2**



# Tooling

# 2





# Basic shortcuts - code

---

- Ctrl+F2     Program reset
- F4            Run to cursor
- F5            Toggle breakpoint
- F7            Execute next line, step into functions
- F8            Execute next line, step over functions
- Shift-F8     Execute until return from function
- F9            Run with debugging



# Basic shortcuts - data

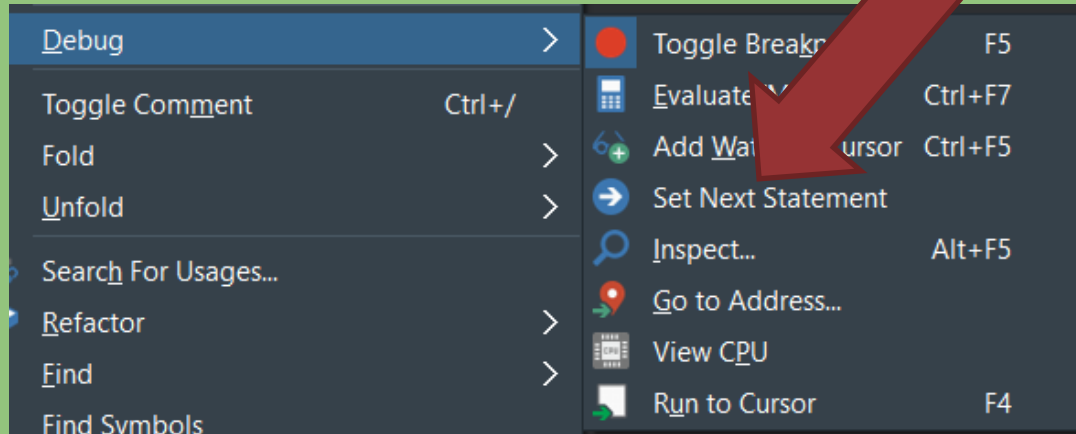
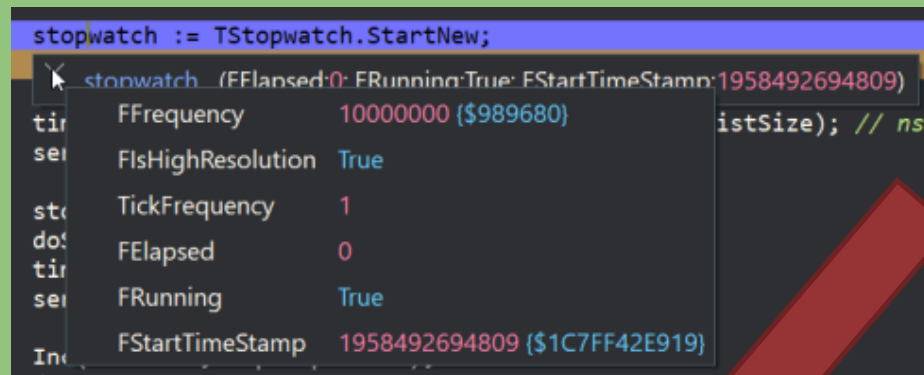
---

- Ctrl+F5    Add watch
- Alt+F5    Inspect
- Ctrl+F7    Evaluate/Modify



# Debugging with mouse

- Ctrl-Shift-click Inspect
- Mouse hoover Evaluate
- <Right-click>, Debug





# Project and Linker options



## Code generation

- > Code inlining control On
- > Code page 0
- > Emit runtime type information ☐ false
- > Minimum enum size Byte
- > Optimization ☐ false
- > Record field alignment Quad word
- > Stack frames ☒ true

## Debugging

- > Assertions ☒ true
- > Debug information Debug information
- > Local symbols ☒ true
- > Symbol reference info Reference info
- > Use debug .dcus ☒ true
- > Use imported data references ☒ true

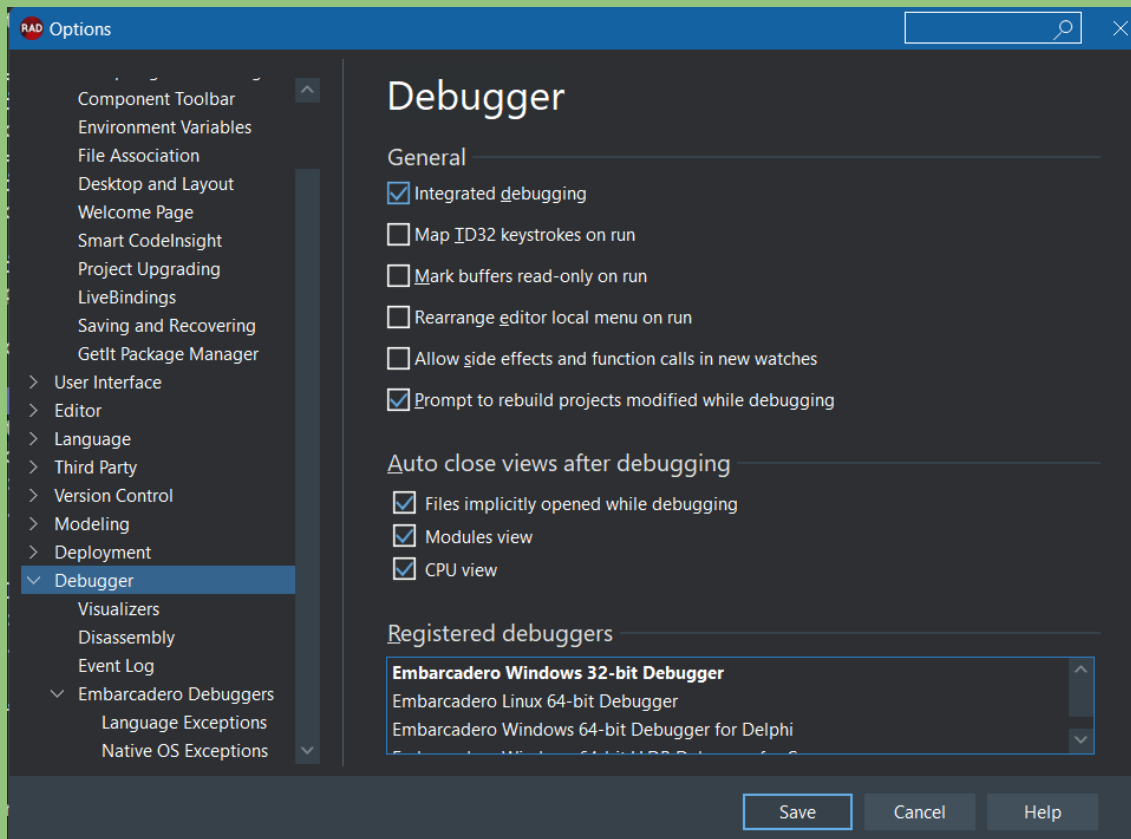


- > Data Execution Prevention compat ☒ true
- > Debug information ☒ true
- > Enable large addresses ☐ false
- > EXE Description
- > Generate console application ☐ false
- > Image Base 400000
- > Include remote debug symbols ☐ false
- > Map file Off





# Debugger options





# Tips & tricks

# 3



# Conditional breakpoints

---

- Set conditional breakpoints
  - Breakpoint properties
    - Condition
    - Pass count
    - Thread
    - Group
    - Enable group, Disable group
  - Slow!



# Conditional breakpoints in code

---

- Conditional breakpoints in code

```
if (condition) then
    sleep(0); // put breakpoint here
```
- Or use GpStuff

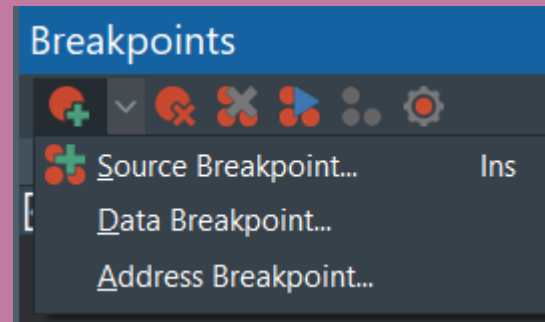
```
DebugBreak(condition);
```





# Hardware breakpoints

- Implemented in the CPU
- Address breakpoint
  - When code at address is executed
- Data breakpoint
  - When data at address is modified

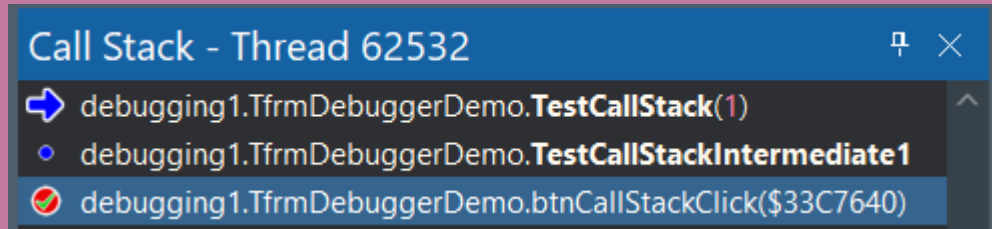




# Breakpoint tricks

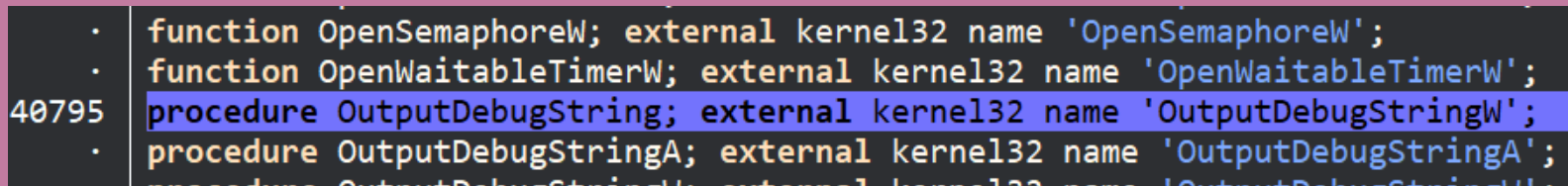
- Breakpoints can be set on the call stack

- Triggers when the code returns to that point

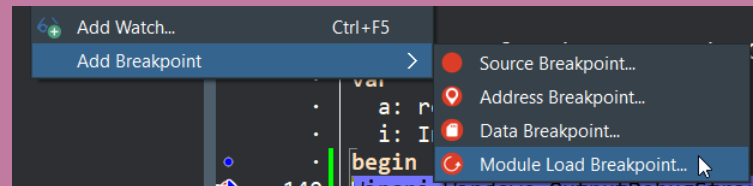


- Breakpoint can be set on a function import

- Requires “Use debug .dcus”



- Breakpoint can be set on module load





# Bad 'initialization' section

---

- Open System unit
- Find procedure InitUnits
- Use “High pass count” trick to find problematic initialization section



# Use breakpoints for logging

---

- Breakpoint properties
  - Break (disable)
  - Evaluate expression
  - Log message
  - Log call stack
- Combine with other breakpoint properties
  - Conditional evaluation etc



# Logging

---

- Make output more visible in debug log

```
OutputDebugString(#13#10'test'#13#10);
```

- Conditional logging

```
if (condition) then  
    Log(some_stuff);
```

- On demand

```
var b := false;  
if b then  
    Log(some_stuff);
```



# Log to console

---

uses GpConsole;

```
Console.WriteLine('xxx');
```

```
Console.Write(['The value is ', value]);
```

- Fast
- Thread-safe
- Can output simple data types (incl. Pointers and Booleans)
- VCL, FireMonkey, and Console applications



# GpConsole

---

- Disable in one line

```
Console.Disabled := true;
```

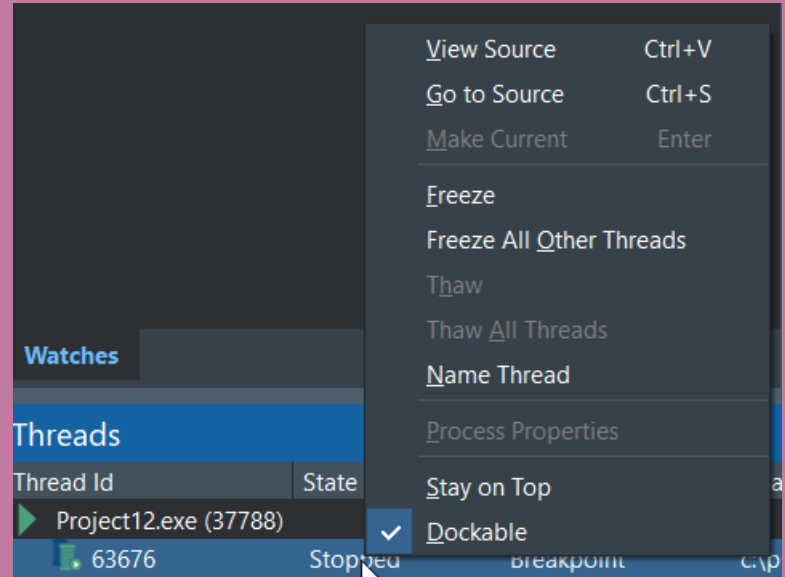
- Use colors to enhance readability

```
Console.WriteLine(['And the answer is {bright red on bright  
yellow}', 42, '{}!']);
```



# Multithreaded programs

- Threads window
- Right-click
  - Freeze
  - Freeze All Other Threads
  - Thaw
  - Thaw All Threads







# Debugging multiple programs

---

- Open project group
  - **Build all**
- Select first program
  - **Run**
- Select second program
  - **Run**
- Debugger is attached to all running instances



# Switching between programs

---

- Run, Detach From Program
- Run, Attach To Process
- Sometimes unstable (can crash)



# Ignoring exceptions

- Tools, Options, Debugger, Embarcadero Debuggers, Language Exceptions
- Or wrap code in two breakpoints
  - First disables exceptions
  - Second re-enables exceptions

## Actions

- ☒ Break
- ☐ Ignore subsequent exceptions
- ☐ Handle subsequent exceptions

## Language Exceptions

- ☒ Notify on language exceptions

### Exception types to ignore

- ☒ EAbort Exceptions
- ☒ Indy Silent Exceptions
- ☒ Microsoft DAO Exceptions
- ☒ System.Threading.SynchronizationLockException
- ☒ System.Threading.ThreadAbortException
- ☒ EPyStopIteration



# Ignoring exceptions – in code

---

```
uses GpStuff;  
var ignore := ExceptionsInDebugger.Ignore;  
try  
...  
finally ignore := nil; end;
```



THANK YOU