Trice User Manual

```
+ Speed of Light `printf` Comfort Within Interrupts And Everywhere +
-   (TL;DR)   ->  Too Long; Don't Read - use it as reference only ❗
```

(go to bottom)

Table of Contents

▶ (click to expand)

- 1. Abstract
- 2. A brief history of Trice
- 3. How it works - the main idea
- 4. Trice Features (Overview)
  - 4.1. Open source
  - 4.2. Easy-to-use
  - 4.3. Small size - using Trice frees FLASH memory
  - 4.4. Execution speed
  - 4.5. Robustness
  - 4.6. Minimal Transfer Bytes Amount
  - 4.7. More comfort than printf-like functions but small differences
  - 4.8. Tags, Color and Log Levels
  - 4.9. Compile Time Enable/Disable Trice Macros on File or Project Level
  - 4.10. Target and host timestamps
  - 4.11. Target source code location
  - 4.12. Several target devices in one log output
  - 4.13. Any byte capable 1-wire connection usable
  - 4.14. Scalability
  - 4.15. Portability and Modularity
  - 4.16. Optional Trice messages encryption
  - 4.17. Trice Protection
  - 4.18. Trice Diagnostics
  - 4.19. Trice Cache
  - 4.20. Avoiding False-Positive Editor Warnings
  - 4.21. Trice Generator
  - 4.22. Versions and Variants Trice Stability
  - 4.23. Legacy Project Code Integration
- 5. Start with Trice
  - 5.1. Get it
  - 5.2. Install It
  - 5.3. Try it
  - 5.4. Use It
  - 5.5. Fork It (get a contributor)
    - 5.5.1. ☑ What “forking” means
    - 5.5.2. How to Fork (GitHub)
  - 5.6. Clone It
  - 5.7. Build It
  - 5.8. Modify It
  - 5.9. Port it
    - 5.9.1. Target Macros
    - 5.9.2. Target Trice Stamps
    - 5.9.3. Trice Checks
    - 5.9.4. Communication Ports
    - 5.9.5. Target Code Overview

        - 5.9.6. User Code Adaption
        - 5.9.7. Limitations
        - 5.9.8. Trice (Time) Stamps
        - 5.9.9. Trice Parameter Bit Widths
    - 5.10. Avoid it
        - 5.10.1. Parser Limitation
        - 5.10.2. Trice macros in header files
        - 5.10.3. Trice macros inside other macros
        - 5.10.4. Upper case only TRICE macros should be written with id(0), Id(0) or ID(0)
- 6. Trice Trouble Shooting Hints
    - 6.1. Initial Data Transfer Setup Hints
    - 6.2. Short Trouble Shooting Hints
- 7. Trice Cache for Compilation Speed
    - 7.1. Trice Cache Idea
    - 7.2. Trice Cache Logic
    - 7.3. Trice Cache Remarks
    - 7.4. Trice Cache Tests
    - 7.5. Possible Trice Cache Editor-Issues And How To Get Around
    - 7.6. Activating the Trice Cache
- 8. Embedded system code configuration
- 9. Trice tool in logging action
- 10. Encryption
- 11. Trice Command Line Interface & Examples
    - 11.1. Common information
    - 11.2. Further examples
        - 11.2.1. Automated pre-build insert command example
        - 11.2.2. Some Log examples
        - 11.2.3. Logging over a display server
        - 11.2.4. Binary Logfile
        - 11.2.5. TCP output
        - 11.2.6. TCP4 input
        - 11.2.7. UDP4 input (accepted pull request 529)
        - 11.2.8. Stimulate target with a user command over UART
        - 11.2.9. Explpore and modify tags and their colors
        - 11.2.10. Location Information
- 12. Limitations
    - 12.1. Permanent Limitations
        - 12.1.1. Limitation TRICE in TRICE not possible
    - 12.2. Current Limitations
        - 12.2.1. String Concatenation Within TRICE Macros Not Possible
        - 12.2.2. Limited Trice Parser Capabilities
        - 12.2.3. Special Care Demands
- 13. Additional hints
    - 13.1. Pre-built executables are available
    - 13.2. Configuration file triceConfig.h
    - 13.3. Setting up the very first connection
    - 13.4. Avoid buffer overruns
    - 13.5. Buffer Macros
    - 13.6. Logfile viewing
    - 13.7. Using the Trice tool with 3rd party tools
    - 13.8. Several targets at the same time
    - 13.9. TRICE_STACK_BUFFER could cause stack overflow with -o0 optimization
    - 13.10. Cycle Counter
- 14. Switching Trice ON and OFF
    - 14.1. Target side compile-time Trice On-Off
    - 14.2. Host side Trice On-Off
- 15. Framing
- 16. Optional XTEA Encryption
- 17. Endianness

- 18. Trice (Time)Stamps
  - 18.1. Target (Time)Stamps Formatting
- 19. Binary Encoding
  - 19.1. Symbols
  - 19.2. Package Format
    - 19.2.1. typeX0 Trices
    - 19.2.2. Framing - NONE or with COBS or TCOBS encoding
- 20. Trice Decoding
  - 20.1. Trice ID list til.json
  - 20.2. Trice location information file li.json
- 21. Trice ID Numbers
  - 21.1. ID number selection
    - 21.1.1. Trice tool internal Method to get fast a random ID
  - 21.2. ID number usage and stability
  - 21.3. Trice ID 0
- 22. Trice ID management
  - 22.1. Trice inside source code
    - 22.1.1. Trice in source code comments
    - 22.1.2. Different IDs for same Trices
    - 22.1.3. Same IDs for different Trices
    - 22.1.4. ID Routing
    - 22.1.5. Possibility to create new tags without modifying trice tool source
- 23. Trice version 1.0 Log-level Control
  - 23.1. Trice version 1.0 Compile-time Log-level Control
  - 23.2. Trice version 1.0 Run-time Log-level Control
  - 23.3. Trice Version 1.0 Compile-time - Run-time Log-level Control
- 24. ID reference list til.json
  - 24.1. til.json Version control
  - 24.2. Long Time Availability
- 25. The Trice Insert Algorithm
  - 25.1. Starting Conditions
  - 25.2. Aims
  - 25.3. Method
    - 25.3.1. Trice Insert Initialization
  - 25.4. User Code Patching (trice insert)
  - 25.5. User Code Patching Examples
  - 25.6. Exclude folders & files from being parsed (pull request 529)
  - 25.7. ID Usage Options
  - 25.8. General ID Management Information
    - 25.8.1. Option Cleaning in a Post-build process
    - 25.8.2. Option Let the inserted Trice ID be a Part of the User Code
    - 25.8.3. Option Cleaning on Repository Check-In
- 26. Trice Speed
  - 26.1. Target Implementation Options
    - 26.1.1. Trice Use Cases TRICE_STATIC_BUFFER and TRICE_STACK_BUFFER - direct mode only
    - 26.1.2. Trice Use Case TRICE_DOUBLE_BUFFER - deferred mode, fastest Trice execution, more RAM needed
    - 26.1.3. Trice Use Case TRICE_RING_BUFFER - deferred mode, balanced Trice execution time and needed RAM
  - 26.2. A configuration for maximum Trice execution speed with the L432_inst example
  - 26.3. A configuration for normal Trice execution speed with the G0B1_inst example
- 27. Trice memory needs
  - 27.1. F030_bare Size
  - 27.2. F030_inst Size with TRICE_OFF=1
  - 27.3. F030_inst with ring buffer
  - 27.4. F030_inst with ring buffer
  - 27.5. A developer setting, only enabling SEGGER_RTT
  - 27.6. A developer setting, only enabling SEGGER_RTT and without deferred output gives after running `./build.sh TRICE_DIAGNOSTICS=0 TRICE_PROTECT=0`:
  - 27.7. Settings Conclusion
  - 27.8. Legacy Trice Space Example (Old Version)

    - 27.9. Memory Needs for Old Example 1
    - 27.10. Memory Needs for Old Example 2
- 28. Trice Project Image Size Optimization
    - 28.1. Code Optimization -o3 or -oz (if supported)
    - 28.2. Compiler Independent Setting (a bit outdated)
    - 28.3. Linker Option --split-sections (if supported)
    - 28.4. Linker Optimization -flto (if supported)
        - 28.4.1. ARMCC Compiler v5 Linker Feedback
        - 28.4.2. ARMCLANG Compiler v6 Link-Time Optimization
        - 28.4.3. GCC
        - 28.4.4. LLVM ARM Clang
        - 28.4.5. Other IDE´s and compilers
    - 28.5. Legacy STM32F030 Example Project - Different Build Sizes
        - 28.5.1. ARMCC compiler v5
- 29. Trice Tags and Color
    - 29.1. How to get
        - 29.1.1. Output options
        - 29.1.2. Check Alternatives
    - 29.2. Color issues under Windows
- 30. Trice without UART
- 31. Trice over RTT
    - 31.1. For the impatient (2 possibilities)
        - 31.1.1. Start JLink commander and connect over TCP
        - 31.1.2. Start using JLinkRTTLogger
        - 31.1.3. JLinkRTTLogger Issue
    - 31.2. Segger Real Time Transfer (RTT)
    - 31.3. J-Link option
        - 31.3.1. Convert Evaluation Board onboard ST-Link to J-Link
        - 31.3.2. Some SEGGER tools in short
        - 31.3.3. JLinkRTTClient.exe
        - 31.3.4. JLinkRTTViewer.exe
    - 31.4. Segger RTT
    - 31.5. Segger J-Link SDK (~800 EUR) Option
    - 31.6. Additional Notes (leftovers)
    - 31.7. Further development
    - 31.8. NUCLEO-F030R8 example
        - 31.8.1. RTT with original on-board ST-LINK firmware
        - 31.8.2. Change to J-LINK onboard firmware
        - 31.8.3. RTT with J-LINK firmware on-board
    - 31.9. Possible issues
    - 31.10. OpenOCD with Darwin (MacOS)
    - 31.11. SEGGER J-Link on Darwin (MacOS)
    - 31.12. Links
- 32. Writing the Trice logs into an SD-card (or a user specific output)
- 33. Trice Target Code Implementation
    - 33.1. TRICE Macro structure
        - 33.1.1. TRICE_ENTER
        - 33.1.2. TRICE_PUT
        - 33.1.3. TRICE_LEAVE
    - 33.2. TRICE_STACK_BUFFER
    - 33.3. TRICE_STATIC_BUFFER
    - 33.4. TRICE_DOUBLE_BUFFER
    - 33.5. TRICE_RING_BUFFER
    - 33.6. Deferred Out
        - 33.6.1. Double Buffer
        - 33.6.2. Ring Buffer
    - 33.7. Direct Transfer
    - 33.8. Possible Target Code Improvements
- 34. Trice Similarities and Differences to printf Usage

    - 34.1. Printf-like functions
    - 34.2. Trice IDs
    - 34.3. Trice values bit width
    - 34.4. Many value parameters
    - 34.5. Floating Point Values
    - 34.6. Runtime Generated 0-terminated Strings Transfer with triceS
    - 34.7. Runtime Generated counted Strings Transfer with triceN
    - 34.8. Runtime Generated Buffer Transfer with triceB
    - 34.9. Remote function call syntax support with triceF
    - 34.10. Extended format specifier possibilities
        - 34.10.1. Trice format specifier
        - 34.10.2. Overview Table
    - 34.11. UTF-8 Support
    - 34.12. Switch the language without changing a bit inside the target code
    - 34.13. Format tags prototype specifier examples
- 35. Development Environment Setup
    - 35.1. Common Information
    - 35.2. Important to know
    - 35.3. Animation
    - 35.4. Setup Linux PC - Example with Debian12 - KDE Desktop
        - 35.4.1. Basic setup
        - 35.4.2. Github
        - 35.4.3. vsCode
        - 35.4.4. Go
        - 35.4.5. Gitkraken (or other GUI for git)
        - 35.4.6. arm-none-eabi toolchain (or other target system compiler)
        - 35.4.7. J-Link (if needed)
        - 35.4.8. Beyond Compare (if no other diff tool)
    - 35.5. Setup Windows PC Example
        - 35.5.1. Setup Trice
        - 35.5.2. Setup ARM Environment Example
        - 35.5.3. Setup STM32
        - 35.5.4. Setup Onboard J-Link on NUCLEO (other ST evaluation boards too)
        - 35.5.5. Setup VS-Code
    - 35.6. Makefile with Clang too
    - 35.7. Download Locations
        - 35.7.1. Clang
        - 35.7.2. GCC
    - 35.8. Install Locations
    - 35.9. Environment Variables
    - 35.10. Build command
    - 35.11. Run & Debug
    - 35.12. Logging
    - 35.13. Setting up a new project
- 36. Example Projects without and with Trice Instrumentation
    - 36.1. Nucleo-F030R8 Examples
        - 36.1.1. F030_bare
        - 36.1.2. F030_inst
    - 36.2. Nucleo-G0B1 Examples
        - 36.2.1. G0B1_bare
        - 36.2.2. G0B1_inst
    - 36.3. Nucleo-L432KC Examples
        - 36.3.1. L432_bare
        - 36.3.2. L432_inst
- 37. Trice Generate
    - 37.1. Colors
    - 37.2. C-Code
    - 37.3. C#-Code
    - 37.4. Generating a RPC Function Pointer List

- 38. Testing the Trice Library C-Code for the Target
  - 38.1. General info
  - 38.2. How to run the tests
  - 38.3. Tests Details
  - 38.4. How to add new test cases
  - 38.5. Test Internals
  - 38.6. Test Results
  - 38.7. Special tests
  - 38.8. Test Cases
    - 38.8.1. Folder Naming Convention
- 39. Test Issues
- 40. Add-On Hints
  - 40.1. Trice on LibOpenCM3
    - 40.1.1. Prerequisites
    - 40.1.2. triceConfig.h
    - 40.1.3. main.c
    - 40.1.4. nucleo-f411re.ld
    - 40.1.5. Makefile
    - 40.1.6. Usage
  - 40.2. Get all project files containing Trice messages
  - 40.3. Building a trice library?
  - 40.4. Possible Compiler Issue when using Trice macros without parameters on old compiler or with strict-C settings
- 41. Trice And Legacy User Code
  - 41.1. Legacy User Code Option Separate Physical Output Channel
  - 41.2. Legacy User Code Option Trice Adaption Edits
  - 41.3. Legacy User Code Option Print Buffer Wrapping and Framing
  - 41.4. Legacy User Code Option Trice Aliases Adaption
    - 41.4.1. PR533 Doc
    - 41.4.2. PR533 Summary
    - 41.4.3. PR533 Motivation
    - 41.4.4. What This PR533 Adds
    - 41.4.5. PR533 Example
    - 41.4.6. PR536 Doc
    - 41.4.7. Alias Example Project
- 42. Future Development
  - 42.1. Trice Log-level Control Specification Draft
    - 42.1.1. What log levels exist in general, including exotic ones, and what is their exact weighting relative to each other?
    - 42.1.2. Compile-time Log-level Control
    - 42.1.3. Run-time Log-level Control
  - 42.2. Trice Structured Logging
    - 42.2.1. Trice Structured Logging Compile-time Information
    - 42.2.2. Trice Structured Logging Runtime Information
    - 42.2.3. Trice Structured Logging Limitations and Special Cases
    - 42.2.4. A Trice Structured Logging Example
    - 42.2.5. Trice Structured Logging CLI Switches and Variables
    - 42.2.6. Trice Structured Logging User Defined Values
    - 42.2.7. Trice Structured Logging CLI Switches Usage Options
    - 42.2.8. Trice Structured Logging Level Specific Configuration
    - 42.2.9. Trice Structured Logging Assert Macros (TODO)
  - 42.3. Improving the Trice Tool Internal Parser
    - 42.3.1. Trice Internal Log Code Short Description
  - 42.4. Using Trice on Servers
- 43. Working with the Trice Git Repository
  - 43.1. Install `opencommit` on MacOS
  - 43.2. Install `opencommit` on Windows
- 44. Trice Maintenance
  - 44.1. Trice Project structure (Files and Folders)
  - 44.2. The .github Folder — Purpose and Contents
    - 44.2.1. `.github` Root

        - 44.2.2. 🗁 `.github/workflows` — GitHub Actions Workflows
        - 44.2.3. GitHub Action clang-format.yml - Check C Code Formatting
        - 44.2.4. GitHub Action codeql.yml - Static Code Analysis
        - 44.2.5. GitHub Action coverage.yml - Test Coverage and Coveralls Integration
        - 44.2.6. GitHub Action go.yml - Building and Testing Go Code
        - 44.2.7. GitHub Action goreleaser.yml - Build & Pack Trice Distribution
        - 44.2.8. GitHub Action greetings.yml - Greeting Message
        - 44.2.9. GitHub Action label.yml - Automatic Labeling Rules
        - 44.2.10. GitHub Action learn-github-actions.yml - Instructional Workflow
        - 44.2.11. GitHub Action link-check.yml - Broken Links Check
        - 44.2.12. GitHub Action manual.ym - To Be Triggered Manually
        - 44.2.13. GitHub Action shellcheck.yml - Catching Common Bash Scripts Bugs
        - 44.2.14. GitHub Action shfmt.yml - Ensure Consistent Shell Scripts Formatting
        - 44.2.15. GitHub Action stale.yml - Automatic Stale Issue Handling
        - 44.2.16. GitHub Action superlinter.yml - Ensure Consistent YAML and Markdown Formatting
        - 44.2.17. Github Action pages.yml - Creates The Trice Github Pages
        - 44.2.18. Github Action test_goreleaser.yml - Checks If Goreleaser Would Succeed
    - 44.3. Trice User Manual Maintenance (or any `*.md` file)
    - 44.4. Cleaning the Sources
- 45. Build and Release the Trice Tool
    - 45.1. Build Trice tool from Go sources
    - 45.2. Prepare A Release
        - 45.2.1. Check a Goreleaser Release before Publishing
    - 45.3. Trigger a **real** Trice release via CI (with `git tag`)
        - 45.3.1. Make sure your workflow reacts to tags
        - 45.3.2. Final checks before tagging
        - 45.3.3. Choose a version and create a `git tag`
        - 45.3.4. Push the tag to GitHub (this triggers CI)
        - 45.3.5. Watch the CI release run on GitHub
        - 45.3.6. Check the GitHub Release
- 46. Scratch Pad

(back to top)

---

1. Abstract

If you develop software for an embedded system, you need some kind of system feedback. Debuggers are awesome tools, but when it comes to analyze dynamic behavior in the field, they are not usable.

Logging then, usually done with printf-like functions, gets quick a result after having i.e. `putchar()` implemented. This turns out to be an expensive way in terms of processor clocks and needed FLASH memory, when you regard the library code and all the strings needing FLASH memory space. For small micro-controllers that´s it.

Bigger micro-controllers are coming with embedded trace hardware. To use it, an expensive tool is needed. Useful for analyzing complex systems, but for in-field related issues at least unhandy.

Unhappy with this situation, the developer starts thinking of using digital pins or starts emitting some proprietary LED blinking codes or byte sequences, difficult to interpret.

The Trice technique tries to fill this gap, being minimal invasive for the target and as comfortable as possible. It is the result of a long-year dissatisfaction and several attempts to find a loophole to make embedded programming more fun and this way more effective.

Trice is an unusual software tracer-logger, using internally IDs instead of format strings to get maximum speed but provides the user with a printf-like comfort:

```
trice("Hello! 👋😀");

int a = -4;
float x = 3.14159265;
trice("info:π/%d is %f with the bit pattern %032b\n", a, aFloat(x/a), x );

string s = "world";
triceS("msg:A runtime generated string: %s", s);
```

Replacing a `printf` library, the Trice target source code occupies 1-4 KB Flash memory and less than 1 KB RAM in dependence of the configuration which is done with a user file named `triceConfig.h`:

```
#define TRICE_DEFERRED_OUTPUT 1
#define TRICE_BUFFER TRICE_DOUBLE_BUFFER
#define TRICE_DEFERRED_UARTA 1
#define TRICE_UARTA USART2
```

The open-source Trice PC tool is executable on all Go platforms, at least:

- [x] Linux
- [x] MacOS
- [x] Windows

In the future other ports are possible:

- C/C++ or Rust program to run on a separate controller board
- Go-mobile
- tinyGo
- Wasm
- Phython

(back to top)

2. A brief history of Trice

Developing firmware means to deal also with interrupts and often with timing. How do you check, if an interrupt occurred? Ok, increment a counter and display it in a background loop with some printf-like function. What about time measurement? Set a digital output to 1 and 0 and connect a measurement device. Once, developing software for a real-time image processing device, I had no clue where in detail the processing time exploded when the image quality got bad. A spare analog output with a video interrupt synced oscilloscope gave me the needed information, after I changed the analog output on several points in my algorithm. But, hey guys, I want to deal with my programming tasks and do not like all this hassle connecting wires and steer into instruments.

A `printf` is so cool on a PC, developing software there. But an embedded device often cannot use it for performance reasons. My very first attempt was writing the format string `.const` offset together with its values in a FIFO during a log statement and to do the `printf` it in the background. But that is compiler specific. Ok the full string address is better but needs buffer space. Zephyr for example does something like that calling it "deferred logging".

Then, one day I had the idea to compute short checksums for the format strings in a pre-compile step and to use them as ID in a list together with the format strings. That was a step forward but needed to write a supporting PC program. I did that in C++ in the assumption to get it better done that way. Finally, it worked, but I hated my PC code, as I dislike C++ now because of all its nuts and bolts to handle, accompanied by missing libraries on the next PC. The tool usability was also unhandy and therefore error prone and the need became clear for a full automatized solution. Also, what is, if 2 different format strings accidentally generate the same short checksum? There was a way around, but an ID based message filtering will never be possible that way.

The need became clear for controllable IDs and management options. And there was Go now, an as-fast-as-**C** language, easy to learn, promising high programming efficiency and portability. It would be interesting to try it out on a real PC project.

Trying to add tags in form of partial Trice macro names was blowing up the header code amount and was a too rigid design. Which are the right tags? One lucky day I came to the conclusion to handle tags just as format string parts like `"debug:Here we are!\n"` and getting rid of them in the target code this way also giving the user freedom to invent any tags.

An other point in the design was the question how to re-sync after data stream interruption, because that happens often during firmware development. Several encodings where tried out and a proprietary escape sequence format and an alternative flexible data format with more ID bits where working reliable but with COBS things got satisfying. A side result of that trials is the Trice tool option to add different decoders if needed. Now the default Trice message framing is TCOBSv1 which includes short message compression and this way allows very low transmit bandwidths and/or saves storage, when binary Trice data are stored in Flash memory.

There was a learning **not** to reduce the transmit byte count to an absolute minimum, but to focus more on Trice macro speed and universality. That led to a double buffer on the target side as an alternative to the ring buffer solution. The actual binary encoding, allowing

alongside user protocols, is result of the optional target timestamps and location info some users asked for, keeping the target code as light as possible. Float and double number support was implementable for free because this work is done mainly on the host side.

Trice grew, and as it got usable I decided to make it Open Source to say "Thank You" to the community this way.

Learning that Trice is also a baby girl name, our daughter Ida designed the little girl with the pen symbolizing the Trice macro for recording and the eyeglasses standing for the PC tool Trice visualizing the logs.

(back to top)

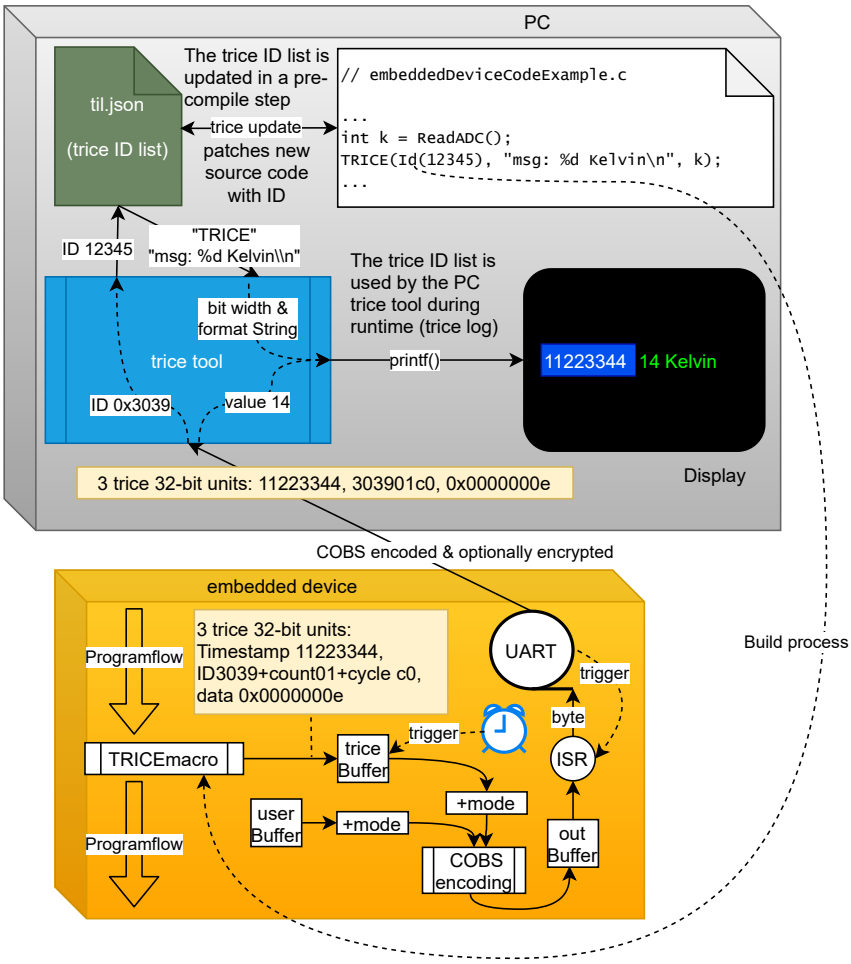
3. How it works - the main idea

Trice performs **no** costly printf-like functions on the target at all. The Trice macro, instead, just copies an ID together with the optional values to a buffer and is done. In the minimum case this can happen in 6(six!) processor clocks even with target timestamps included. When running on a 64 MHz clock, **light can travel about 30 meters in that time**.

To achieve that, a pre-compile step is needed, executing a `trice insert` command on the PC. This is fast enough not to disturb the build process. The Trice tool parses then the source tree for macros like `trice( "msg: %d Kelvin\n", k );` and patches them to `trice( iD(12345), "msg: %d Kelvin\n", k );`, where `12345` is a generated 14-bit identifier (ID) copied into a **T**rice **I**D **L**ist. During compilation than, the Trice macro is translated to the `12345` ID only, and the optional parameter values. The format string is ignored by the compiler.

The target code is project specific configurable. In **direct mode** the the stack or a static buffer is used as Trice buffer and the Trice macro execution includes optionally the quick COBS encoding and the data transfer. This more straightforward and slower architecture can be interesting for many cases because it is anyway much faster than printf-like functions calls. Especially when using Trice over RTT a single Trice is executable within ~100 processor clocks. See `TRICE_DIRECT_SEGGER_RTT_32BIT_WRITE` inside triceDefaultConfig.h and look into the examples folder. In **deferred mode** a service swaps the Trice double buffer or reads the Trice ring buffer periodically, the configured encoding, default is TCOBS, takes part and with the filled buffer the background transfer is triggered. Out buffer and Trice buffer share the same memory for efficiency.

During runtime the PC Trice tool receives all what happened in the last ~100ms as a package from the UART port. The `0x30 0x39` is the ID 12345 and a map lookup delivers the format string *"msg: %d Kelvin\n"* and also the bit width information. Now the Trice tool can write target timestamp, set msg color and execute `printf("%d Kelvin\n", 0x0000000e);`



The Trice tool is a background helper giving the developer focus on its programming task. The once generated ID is not changed anymore without need. If for example the format string gets changed into `"msg: %d Kelvin!\n"`, a new ID is inserted automatically and the reference list gets extended. Obsolete IDs are kept inside the **T**rice **I**D **L**ist for compatibility with older firmware versions. It could be possible, when merging code, an ID is used twice for different format strings. In that case, the ID inside the reference list wins and the additional source gets patched with a new ID. This maybe unwanted patching is avoidable with proper Trice ID management. The reference list should be kept under source code control.

Moreover, using `trice i -cache && make && trice c -cache` in a build script makes the IDs invisible to the developer reducing the data noise giving more space to focus on the development task. See build.sh as a working example and the Trice Cache chapter for details.

(back to top)

4. Trice Features (Overview)

4.1. Open source

Target code and PC tool are open source. The MIT license gives full usage freedom. Users are invited to support the further Trice development.

4.2. Easy-to-use

Making it facile for a user to use Trice was the driving point just to have

- one Trice tool
- one additional target code source folder
- a project specific simple to use triceConfig.h
- and to get away with the one macro `trice` for most situations.

Trice understands itself as a silent helper in the background to give the developer more focus on its real task. If, for example, `trice log` is running and you re-flash the target, there is ***no need to restart*** the Trice tool. When til.json was updated in an pre-build step, the Trice tool automatically reloads the new data during logging.

The Trice tool comes with many command line switches (`trice help -all`) for tailoring various needs, but mostly these are not needed. In file ../internal/args/tricehelpall_test.go the expected test output contains this information as well.

Normal Trice tool usage is:

- ./build.sh containing `trice insert -cache`, `make` and `trice clean -cache`
- `make log` containing `trice l -p COMn` for logging with default baud rate.

In this example, the user code gets **not** polluted with Trice IDs - they exists only during the compilation step and the Trice cache makes this invisible for the user and the build system.

4.3. Small size - using Trice frees FLASH memory

Compared to a printf-library code which occupies 1 to over 20 KB FLASH memory, the Trice code is normally smaller but provides full support.

4.4. Execution speed

Can it get faster than 6 clocks only? Only 3 runtime Assembler instructions per Trice needed in the minimum case! Optional target timestamp, critical sections, cycle counter, diagnostics and overflow protection can consume a few more processor clocks, if enabled, but a Trice is still incomparable fast.

4.5. Robustness

When a Trice data stream is interrupted, the optional COBS or TCOBS encoding allows an immediate re-sync with the next COBS/TCOBS package delimiter byte and a default Trice **cycle counter** gives a high chance to detect lost Trice messages. See also Versions and Variants Trice Stability.

4.6. Minimal Transfer Bytes Amount

A Trice message is 4 bytes long (2 ID bytes and 2 count bytes) plus optional time stamps and/or values. In conjunction with the compressing TCOBS framing the Trice data stream is as small as possible. Use the `-debug` switch to see the compressed and framed packages alongside the decompressed ones together with the decoded messages.

To see the encoding for each single message `#define TRICE_DEFERRED_TRANSFER_MODE TRICE_SINGLE_PACK_MODE` inside the project specific *triceConfig.h*.

Without `-debug` CLI switch:

```
ms@MacBook-Pro G0B1_inst % trice log -p /dev/tty.usbmodem0007722641261 -prefix off -li off -hs off -ts off
...
This is a message without values and without stamp.
...
```

With `-debug` CLI switch:

```
ms@MacBook-Pro G0B1_inst % trice log -p /dev/tty.usbmodem0007722641261 -prefix off -li off -hs off -ts off
-debug
...
TCOBSv1: c1 74 e2 23 00
->TRICE: c1 74 e2 00
This is a message without values and without stamp.
...
```

The TCOBS encoding cannot compress in the example above, because the data are too small, but here is a significant compression result shown:

```
ms@MacBook-Pro G0B1_inst % trice log -p /dev/tty.usbmodem0007722641261 -prefix off -hs off -debug
...
TCOBSv1: b8 76 7b 18 84 fe e1 fd e1 fc e1 fb e1 fa e1 00
->TRICE: b8 76 7b 18 ff ff ff ff fe ff ff ff fd ff ff ff fc ff ff ff fb ff ff ff fa ff ff ff
```

```
_test/testdata/triceCheck.c   805             value=-1, -2, -3, -4, -5, -6
...
```

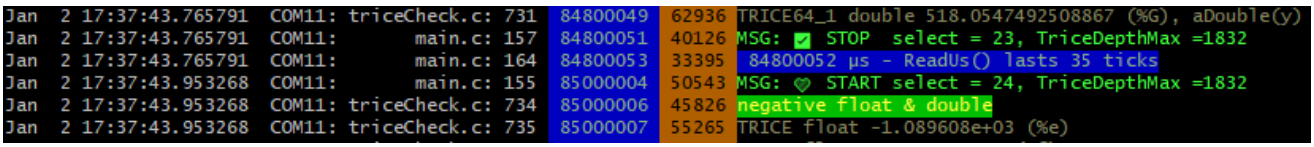
- The `TRICE_SINGLE_PACK_MODE` inserts after each Trice a package delimiter `0`.
- The `TRICE_MULTI_PACK_MODE` inserts after a group of Trice messages a package delimiter `0`, what minimizes the transmit data amount.

When encryption is active, a compression makes no sense, but the `TRICE_MULTI_PACK_MODE` can help to reduce the total amount of padding bytes, because each encrypted package must have a multiple of 8 as length.

```
ms@MacBook-Pro G0B1_inst % trice log -p /dev/tty.usbmodem0007722641261 -prefix off -hs off -pw MySecret -
pf cobs -debug
...
cobs: 21 84 b7 60 8b 21 89 1e e3 07 6d dc d9 2d 6f 59 04 8e 50 8f 24 1c a2 63 2e 3d 4a 57 ef 39 63 01 cb
00
->TRICE: 84 b7 60 8b 21 89 1e e3 07 6d dc d9 2d 6f 59 04 8e 50 8f 24 1c a2 63 2e 3d 4a 57 ef 39 63 01 cb
-> DEC:  cc b6 63 01 71 02 ff fe cd 76 72 03 ff fe fd ce f6 64 81 00 00 73 04 ff fe fd fc 00 00 00 00 00
_test/testdata/triceCheck.c   827        0_355 value=-1, -2
_test/testdata/triceCheck.c   828              value=-1, -2, -3
_test/testdata/triceCheck.c   829    0,033_124 value=-1, -2, -3, -4
cobs: 19 50 70 79 d7 75 6f d7 99 dc d8 ec 06 e1 66 e7 a7 c1 0d 96 85 df 19 25 55 00
->TRICE: 50 70 79 d7 75 6f d7 99 dc d8 ec 06 e1 66 e7 a7 c1 0d 96 85 df 19 25 55
-> DEC:  cf b6 64 01 74 05 ff fe fd fc fb d0 76 75 06 ff fe fd fc fb fa 00 00 00
_test/testdata/triceCheck.c   830        0_356 value=-1, -2, -3, -4, -5
_test/testdata/triceCheck.c   831              value=-1, -2, -3, -4, -5, -6
...
```

4.7. More comfort than printf-like functions but small differences

Trice is usable also inside interrupts and extended format specifier possibilities give options like binary or bool output. Transmitting runtime generated strings could be a need, so a `triceS` macro exists supporting the `%s` format specifier for strings up to 32737 bytes long. It is possible to log float/double numbers using `%f` and its relatives, but the numbers need to be covered with the fast converter function `aFloat(x)` or `aDouble(y)`. Also UTF-8 encoded strings are implicit supported, if you use UTF-8 for the source code. See chapter Trice Similarities and differences to printf usage for more details.

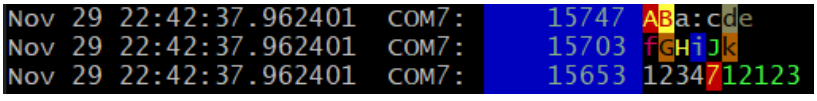


4.8. Tags, Color and Log Levels

You can label each Trice with a tag specifier to colorize the output. This is free of any runtime costs because the tags are part of the Trice log format strings, which are not compiled into the target. The Trice tool will strip full lowercase tag descriptors from the format string after setting the appropriate color, making it possible to give each message its color.

Loggers use log levels and offer a setting like "log all above **INFO**" for example. The Trice tags can cover that but can do better: Inside package *emitter.ColorChannels* in a single file ./internal/emitter/lineTransformerANSI.go all common log levels defined as Trice tags alongside with user tags. The user can adjust this. The Trice tool has the `-pick` and `-ban` switches to control the display in detail. Also a `-logLevel` switch is usable to determine a display threshold as tag position inside ColorChannels.

If an inside-target log selection is needed (routing), the Trice tool can assign each log tag a separate ID range and a target side ID based log selector can control which IDs are transmitted over which output channel. See chapter Trice ID management or type `trice help -insert` and look for `-IDRange`.



4.9. Compile Time Enable/Disable Trice Macros on File or Project Level

After debugging code in a file, there is no need to remove or comment out Trice macros. Write a `#define TRICE_OFF 1` just before the `#include "trice.h"` line and all Trice macros in this file are ignored completely by the compiler, but not by the Trice tool. In case of re-constructing the **T**rice **ID** **L**ist, these no code generating macros are regarded.

```
#define TRICE_OFF 1 // Disable trice code generation for this file object.
#include "trice.h"
```

When you wish to build a firmware without any Trice code, it is sufficient to add

```
C_DEFS += -DTRICE_OFF=1 // Define TRICE_OFF=1 for the whole project.
```

or similar to your Makefile.

4.10. Target and host timestamps

For each Trice you can have (time) stamps or not:

- `trice( "...", ...);` or `TRICE( id(0), ( "...", ...)`: no stamp:
- `Trice( "...", ...);` or `TRICE( Id(0), ( "...", ...)`: 16-bit stamp:
- `TRice( "...", ...);` or `TRICE( ID(0), ( "...", ...)`: 32-bit stamp:

The optional 16- or 32-bit value carry than the system clock, a millisecond second or an other event counter configured in the project specific triceConfig.h. The Trice tool will automatically recognize and display the stamps in a mode you can control. If several Trice macros form a single line, the Trice tool only displays the target timestamp of the first Trice macro.

Embedded devices often lack a real-time clock and some scenarios can last for weeks. Therefore the Trice tool precedes each Trice line with a PC timestamp, if not disabled. This is the Trice reception time on the PC, what can be some milliseconds later than the target Trice event.

4.11. Target source code location

Some developers like to see the `filename.c` and `line` in front of each log line for quick source location. During `trice i` a file `li.json` is generated containing the location information. If `trice log` finds this file, filename and line number are displayed in front of each log line, otherwise not.

Because software is a matter of change it could happen you get obsolete information this way. Therefore the Trice tool log option `-showID` exists to display the Trice ID in front of each log line what gives a more reliable way for event localization in some cases. Also you can get it for free, because no target code is needed for that.

4.12. Several target devices in one log output

Several Trice tool instances can run parallel on one or different PCs. Each Trice tool instance receives *Trices* from one embedded device. Instead of displaying the log lines, the Trice tool instances can transmit them over TCP/IP (`trice l -p COMx -ds`) to a Trice tool instance acting as display server (`trice ds`). The display server can fold these log lines in one output. For each embedded device a separate Trice line prefix and suffix is definable. This allows comparable time measurements in distributed systems.

4.13. Any byte capable 1-wire connection usable

The usual Trice output device is an UART but also SEGGER-RTT is supported over J-Link or ST-Link devices. Many micro controller boards can act as Trice bridge to a serial port from any port (Trice without UART).

4.14. Scalability

The various Trice ID management options allow the organization also of bigger software systems. 16383 possible different IDs should match also large projects. Just in case: 16-bit for the ID is a not to hard changeable value.

4.15. Portability and Modularity

The Trice tool is written in the open source language *Go* and is therefore usable on many platforms. That means the automatic code patching and ID handling side with `trice insert`.

All C-compilers should be usable to compile the target Trice code and there is no hardware dependency despite the byte transmission. MCUs with 8-bit to 64-bit, little or big endian are supported.

Any user program able to read a JSON file, can receive the documented Trice message format, look-up the ID and perform a printf-like action to translate into log strings. The Trice tool with its `log` switch is a working example.

Using no framing, COBS or TCOBS packages starting with a package descriptor allows alongside user protocols. The other way around is also implementable: In a user protocol embedded `Trice` messages.

The Trice tool is expandable with several decoders. So it is possible to implement a minimal Trice encoding, if bandwidth matters heavily and control that with switches.

When less RAM usage is more important the target double buffer is replaceable with a ring buffer. So the user will be able to decide at compile time about that. A ring buffer mode is selectable inside triceConfig.h avoiding any buffer by paying a time toll.

The Trice tool supports many command line switches.

4.16. Optional Trice messages encryption

The encryption opportunity makes it possible to test thoroughly a binary with log output and releasing it without the need to change any bit but to make the log output unreadable for a not authorized person. Implemented is the lightweight XTEA as option, what will do for many cases. It should be no big deal to add a different algorithm.

4.17. Trice Protection

When using Trice, data are written into buffers. A buffer overflow is impossible with the default configuration `#define TRICE_PROTECT 1` by simply ignoring possible overflow causing Trice statements. Those cases are not detectable by the cycle counter evaluation because non-existing Trice data on the embedded system cannot cause cycle errors. Therefore overflow error counters exists, which the user can watch. In ./examples/exampleData/triceLogDiagData.c an option is shown. Of course this buffer overflow protection costs valuable execution time. If you prefer speed over protection, simply write into your project specific *triceConfig.h* `#define TRICE_PROTECT 0`.

4.18. Trice Diagnostics

A trice statement produces 4 bytes buffer data plus optional values data. When for example `TRice16("Voltage=%u\n"), x);` is called inside the ms system tick interrupt every 5th time, 10 bytes data are generated each 5 milliisecond. This needs a transfer baudrate of at least 20.000 bit/s. A UART running at 115.200 baud can easily handle that. Anyway after 100 ms, a 200 Bytes buffer is filled and the question arises what is the optimal Trice buffer size. A caclulation is error prone, so measuring is better. So configure the buffer sizes bigger than estimated and watch the max depth of their usage. In ./examples/exampleData/triceLogDiagData.c an option is shown. After you optimized your buffer sizes, you can deactivate the Trice diagnostics in your project specific *triceConfig.h* with `#define TRICE_DIAGNOSTICS 0`.

4.19. Trice Cache

One may think, automatically cleaning the IDs in the target code with `trice c` after building and re-inserting them just for the compilation needs file modifications all the time and a permanent rebuild of all files containing Trices will slow down the re-build process. That is true, but by using the Trice cache this is avoidable. Simply one-time create a `.trice/cache` folder in your home directory and use `trice insert -cache` and `trice clean -cache` in your build.sh script. More details you find in chapter Trice Cache for Compilation Speed.

4.20. Avoiding False-Positive Editor Warnings

When the user writes

```
trice("msg: Hello! 👌😊\n");
```

after `trice insert` this gets

```
trice(iD(123), "msg: Hello! 👌😊\n");
```

and the compiler builds and then with `trice clean`, this gets again

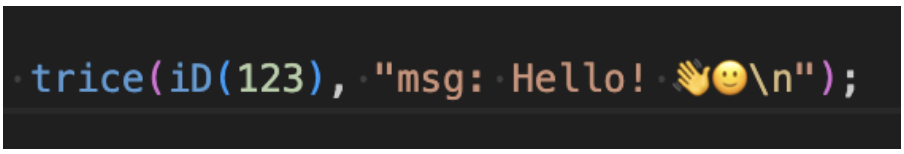
```
trice("msg: Hello! 👌😊\n");
```

Sophisticated editors may detect the missing ID and warn by underlining the trice command:

To avoid this you can add the following line to your project specific *triceConfig.h* file:

```
#define TRICE_CLEAN 1
```

The Trice tool, will change the value to 0 and change it back to 1, when performing the ID insertion and cleaning, when this line occurs inside the *triceConfig.h* file. This way these false-positive editor warnings are avoidable:



It is recommended to use the Trice cache in conjunction with this to avoid a permanent re-translation of files including Trice code.

TRICE_CLEAN==1 changes all Trice macros into empty ones. It is used only to silence sophisticated editors. In the cleaned state, when the IDs are removed from the files, the editor could underline the Trice macros indicating a false positive.

Do not use TRICE_CLEAN for disabling Trice macros. The *triceConfig.h* line `#define TRICE_CLEAN 0` changes to `1` with every `trice clean` and to `0` with every `trice insert`. This line is optional and must not be in a different file. If you want to disable Trice macros use TRICE_OFF.

4.21. Trice Generator

The Trice tool is able to generate colors or code to support various tasks. One interesting option is the Remote Procedure Call support, allowing RPC usage in a network of embedded devices.

Read chapter Trice Generate or type:

```
trice help -generate
```

4.22. Versions and Variants Trice Stability

When developing firmware, we get often different versions and variants in the developing process. When, for example, getting an older device back, it could be, we do not know the flashed firmware version at all. Because the Trice tool adds only IDs and their Trices to the project specific *til.json* file, the complete development history remains in that file. So connecting an old device to the Trice tool will deliver correct output. Of course the location information will be outdated. But when reading the Trice logs the compiled version should get visible and it is no big deal to get the correspondenting *li.json* from the repository. If not, using the `-showID "%6d"` Trice log option displays the Trice IDs and you can easily grab the source code file and line.

4.23. Legacy Project Code Integration

When it comes to instrument a legacy project with Trice or to intergrate legacy project files into a Trice instrumented project different approaches are possible:

1. Use for user specific log statements a different output channel. No special care has to be taken. This is maybe acceptable in some cases.
2. Replace user specific log statements with Trice statements using a text processor and adapt the float, double or runtime strings handling manually. This is acceptable for small code amounts and when is is no problem to edit the legacy sources.

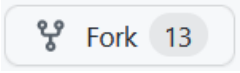
3. Get the legacy output packages before transmitting them, add a 2-byte count in little-endian (0-16383) in front and frame them the same way the trice packages get framed (for example with COBS). This will set the 2 most significant bits to 00 and the Trice tool, can get informed via CLI switch to treat those packages accordingly. The user code containing specific logs will work unchanged together with Trice code over the same output channel.
4. Take advantage of the new support for dynamic trice and triceS macro aliases (Legacy User Code Option: Trice Aliases Adaption](#legacy-user-code-option:-trice-aliases-adaption)).

(back to top)

5. Start with Trice

5.1. Get it

- Download latest release assets for your system: Compressed source code and binaries.
- OR Get the repo: 

  - Create a Github Account
  - Create SSH key pair inside `~/.ssh/`: `ssh-keygen -t ed25510`
  - Add content of `~/.ssh/id_ed25519.pub` as SSH key to Github.
  - Execute `git clone git@github.com:rokath/trice.git` to get the trice repository.
- OR use the 

button

5.2. Install It

- Place the extracted Trice binary somewhere in your PATH.
- Copy the src folder into your project and add all files.
- Copy a triceConfig.h from a subfolder in the examples or test folder and optionally adapt it. See file *triceDefaultConfig.h* for help.
  - Inside the triceConfig.h file you can control, if Trice works in direct or deferred mode or both parallel.

5.3. Try it

- Create a file `tryTrice.c` and write in it:

```
#include "trice.h"

int tryIt( void ){
    trice( "Hello! 👋😊\a\n" ); // A message with sound and without target timestamp.
}
```

You can also edit any of your existing project files accordingly. Just replace any `printf` with `trice`. (Handle float or double numbers and runtime-generated stings, according to Trice Similarities and Differences to printf Usage. The file _test/testdata/triceCheck.c shows many usage examples. The uppercase Trice macros are inlining the complete Trice code and the lowercase Trice macros are function calls, so most probably you want use `trice` to keep the overall code size smaller.

- Create 2 empty files `til.json` and `li.json` in your project root.
- Run `trice insert` and the trice code line changes to `trice( iD(1234), "Hello! 👋😊\a\n" );`.
- The 2 JSON files are now filled with information.
- Run `trice clean` and the trice code line changes back to `trice( "Hello! 👋😊\a\n" );`.

You can use `trice insert` as pre- and `trice clean` as post-compile step, to not spoil your source code with IDs.

> **The optional Trice cache technique avoids un-edited file changes at all, what means no Trice releated build speed disadvantages.**

See Trice Cache for Compilation Speed for more details and examples/G1B1_inst/build.sh as example.

- Or, use `trice insert` in a post-checkout and `trice clean` in a pre-check-in script to keep just the repository clean of Trice IDs. Using only `trice insert` as pre-compile step is possible too, especially when the code is used just in a single project and you wish to have it as compiled.

- When using Trice in libraries for several projects, it may make sense to check-in the libraries with IDs and to use a dedicated ID space for them. See ../_test/testdata/triceCheck.c as an example - especially when building several projects parallel like shown in the examples folder.

A quick setup is possible when using RTT as output channel. Otherwise you need to setup a serial port for Trice data transmission. Other output paths possible too using the auxiliary interface.

5.4. Use It

- In a console, like git bash, type `trice help -all`. You should see the complete Trice tool CLI documentation.
  - Don´t worry, most of it you will never need.
  - There are only 2 important commands: `trice insert` and `trice log`. Call them with the right CLI switches.
    - `trice help -insert` and `trice help -log` show partial help.
    - Examples:

CLI command	Description
`touch ./til.json`	Create an empty `til.json file`. This is needed only the very first time.
`trice i -src . -src ../myLib`	Insert IDs to the current and your `../myLib` folder. This will read|extend|modify `./til.json` and use & create the `./li.json` file.
...	Compile your project
`trice c -src . -src ../myLib`	Optionally restore the current and your `../myLib` folder. This will read|extend|modify `./til.json` and use & create the `./li.json` file.
`trice l -p com1 -baud 921600 -lf my/path/auto`	Start Logging over UART and create automatically a new log file in `my/path/`.
`cat filename.log`	View a recorded log file.
`trice l -p JLINK -args "..."`	Start Logging over RTT. Binary log files are collected in `./temp`.
`trice l -p FILEBUFFER -args logfile.bin`	Play a recorded binary log file.

    - It is recommended to add `trice insert ...` as pre-compile step into the tool chain.
    - Hint: It is possible to add `trice clean ...` as a post-compile step, so that you can check in your project sources without IDs. That is supported in v0.61.0 and later. This allows to use library sources with trices in different projects and the source code is not spoiled with IDs. The `-cache` CLI switch is recommended then. See Trice Cache for Compilation Speed.
- The command `trice` does not make any assumptions about the target processor - 8-bit to 64-bit, supports little and big endianness.
- The command `trice` is compiler agnostic - it should work with any compiler.
- The vsCode editor is free downloadable and free usable, like shown in the `examples/F030_inst` project.
  - Even if you do not have such hardware, you can compile the `examples/F030_inst` project just to get started.
  - When adding or modifying Trice macros inside examples/F030_inst/Core/Src/main.c and recompiling you should see automatically changed ID numbers inside the code.
- The examples and test sub-folders contains several vsCode Makefile projects and they are also usable as starting points for your configuration.
- You can use Trice calls also inside header files but when running `trice insert` as pre- and `trice clean` as post-compile step, all files including these headers will be re-compiled every time, what may be too time consuming. Enable the Trice cache then. See Trice Cache for Compilation Speed for more information.

(back to top)

5.5. Fork It (get a contributor)

If you wish to get a contributor please fork the Trice repository.

5.5.1. ☑ What “forking” means

Forking creates **your own copy** of someone else's repository under your account.
You can then:

- freely make changes,
- push commits to your fork,
- and later submit a **pull request** to propose changes back to the original repo.

5.5.2. 🍴 How to Fork (GitHub)

1. Go to the repository you want to fork

Example: `https://github.com/rokath/trice`

2. Click the "Fork" button (top-right)

You'll be taken to a *Create Fork* page.

3. Choose options (usually leave defaults)

- **Owner** → your GitHub account
- **Repository name** → auto-filled
- Optional: copy only the default branch

Click **Create Fork**.

4. Clone your fork locally

```
git clone https://github.com/YOUR_USERNAME/trice.git && cd trice
```

5. (Optional but recommended) Add the original repo as `upstream`

This lets you pull updates later.

```
git remote add upstream https://github.com/rokath/trice.git
```

Check remotes:

```
git remote -v
```

6. Keep your fork updated

```
git fetch upstream git merge upstream/main
```

Or:

```
git pull upstream main
```

5.6. Clone It

1. Make sure Git is installed

Check with:

```
git --version
```

If not installed, download from https://git-scm.com

2. Clone the repository

Run this command in your terminal or command prompt:

```
git clone https://github.com/rokath/trice.git
```

This creates a local folder named **trice** with the full project history.

3. (Optional) Enter the project folder

```
cd trice
```

5.7. Build It

See Build Trice tool from Go sources.

5.8. Modify It

If for example you wich to change the logging capablilities, like changing/extending CLI switches, thanks to **Go** this is very easy also if you are not familar with **Go**. See this example.

5.9. Port it

Trice should be usable on any MCU with any compiler. On ARM MCUs the easiest way is to use SEGGER J-Link with RTT as output. Setting up UART transmission as alternative or additionally is also no big deal.

Compare folders of one of these folder groups:

Without Instrumentation	With Trice Instrumentation	Remarks
`./examples/F030_bare`	`./examples/F030_inst`	no RTOS
`./examples/G0B1_bare`	`./examples/G0B1_inst`	FreeRTOS
`./examples/L432_bare`	`./examples/L432_inst`	FreeRTOS

This way you see in a quick way any needed adaptions for your target project to port trice to it.

The chapter Example Projects without and with Trice Instrumentation contains further helpful information.

5.9.1. Target Macros

The easiest and mostly sufficient way to use Trice on the target side is the Trice macro

```
trice("Hello world!"); // without     time stamp
Trice("Hello world!"); // with 16-bit time stamp
TRice("Hello world!"); // with 32-bit time stamp
```

which you can mostly use as a `printf` replacement in legacy code. See Trice Similarities and differences to printf usage for more details. Is uses the `TRICE_DEFAULT_PARAMETER_BIT_WIDTH` value (usually 32), which is equal for all values.

The additional macros

- `trice8`, `trice16`, `trice32`, `trice64`
- `Trice8`, `Trice16`, `Trice32`, `Trice64`
- `TRice8`, `TRice16`, `TRice32`, `TRice64`

are always usable and the number 8, 16, 32, 64 specifies the parameter width, which is equal for all values within one macro. Trice macros are partially disabled, when the value TRICE_SINGLE_MAX_SIZE is defined to be smaller than 104. For example with TRICE_SINGLE_MAX_SIZE == 8, `TRice32` can have no parameter value (4 byte Trice header, 4 byte stamp) and `trice8` can have up to 4 parameter values (4 byte Trice header, 4 byte values) That's mainly to get compiler errors rather than runtime errors.

More examples:

Trice	Header	Stamp	max. Values	Trice Size
`trice8`	4	0	0 *1 byte	4
...	...	...	...	...
`trice8`	4	0	12 *1 byte	16
`Trice8`	4	2	0 *1 byte	6
...	...	...	...	...
`Trice8`	4	2	12 *1 byte	18

Trice	Header	Stamp	max. Values	Trice Size
TRice8	4	4	0 *1 byte	8
...	...	...	...	...
TRice8	4	4	12 *1 byte	20
trice16	4	0	2 *2 byte	8
Trice16	4	2	1 *2 byte	8
trice32	4	0	1 *4 byte	8
Trice32	4	2	1 *4 byte	10
TRice32	4	4	2 *4 byte	16
trice64	4	0	1 *8 byte	12
TRice64	4	4	1 *8 byte	16
...	...	...	...	...
TRice64	4	4	12 *8 byte	104

The value TRICE_DEFAULT_PARAMETER_BIT_WIDTH is the parameter bit with for the macros `trice`, `Trice`, `TRice` (without number). It can make sense to set this value to 16 on smaller machines.

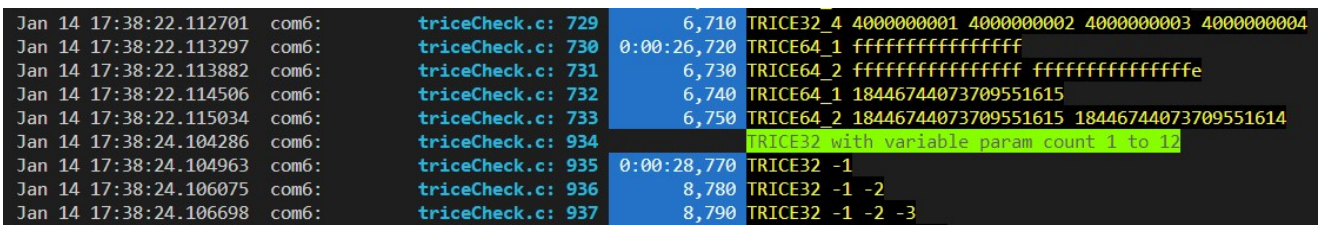
The full uppercase macro Trice is a Trice macro only using inline code. Because the main design aim was speed, this was the original design. Then it became clear, that several hundred of Trice macros increase the needed code amount too much and that it is better to have just a function call instead of having inline macros. If speed matters use `TRICE(id(0)`, `TRICE(Id(0)`, `TRICE(ID(0)` else use `trice(iD(0)`, `Trice(iD(0)`, `TRice(iD(0)` or mix usage as you like. The lower case macros internally use Trice like code but each is only a function call and therefore needs less space.

5.9.2. Target Trice Stamps

- If you wish to have your Trice messages stamped, most probably time stamped, add the 2 hardware specific macros/functions to your project (example in ./examples/F030_inst/Core/Inc/triceConfig.h and ./examples/F030_inst/Core/Src/stm32f0xx_it.c). The time base is in your hands and is allowed to be different for the 16-bit and 32-bit stamps. Example:

```
//! ms32 is a 32-bit millisecond counter, counting circular in steps of 1 every ms.
extern uint32_t ms32;
#define TriceStamp16 (SysTick->VAL) // Counts from 31999 -> 0 in each ms.
#define TriceStamp32  ms32
```

- In the code snippet above the 32-bit timestamp is used for milliseconds and the 16.bit timestamp is used as clock counter what allows fine grained time measurements.
- In the screenshot below, the 16-bit timestamp is a parallel counter running between 0-9999 milliseconds, which allows to have 16-bit timestamps all the time and only every 10 seconds is a full 32-bit timestamp needed.



- The trice tool `-ts*` CLI switches allow customization. With `-hs off` host time stamps are suppressed.
- It is also possible to use the stamp option not for time stamps but for any values, like addresses or a voltage or a random number.

Hint: I usually have the 32-bit timestamp as millisecond counter and the 16-bit timestamp as systick counter to measure short execution times.

5.9.3. Trice Checks

- Optionally copy parts of ./_test/testdata/triceCheck.c to your project if you wish to perform some checks.
  - Do not includce this file directly, because it could get changed when `updateTestData.sh` is executed inside the `./test` folder.
  - The only-uppercase `TRICE*` macros include trice code sequences what can led to a significant code amount if you use plenty of them, whereas the lowercase macros `trice`, `Trice`, `TRice` and their relatives are just function calls and better suited to be used normally.
- In your source files add line `#include "trice.h"` at the top.
- In a function write a trice message like: `TRice( "1/11 = %g\n", aFloat( 1.0/11 ) );`.
- In **project root**:
  - Create empty file: `touch til.json`.
  - `trice insert` should perform **automatically** the following things (The numbers are just examples.):
    - Patch source.c to `TRice( iD(12363), "1/11 = %g\n", aFloat( 1.0/11 ) );`
      - C & H files containing Trice macros, are only modified if needed (missing IDs or changed format strings).
    - Extend `til.json`
      - If no `til.json` is found nothing happens. At least an empty file is needed (Safety feature).
- When the program runs later, it should output something similar to
```
Jan 22 15:43:34.028751  COM13:          main.c: 163       55 1/11 = 0.09090909
```
- Look into Trice Similarities and differences to printf usage for options.
- Read chapter Trice Project Image Size Optimization if needed.

5.9.4. Communication Ports

- For RTT the SEGGER source is already included. See Trice over RTT for more info.
  - If RTT is used, no hardware specific adaptions needed and it is the fastest possible data transfer. But you cannot use it in the field usually.
  - The direct trice mode is recommended for RTT. The single trice execution is a bit longer then, but the log is completely done in one shot. It takes about 100-150 processor clocks, aka 1-2 microseconds.
    - Info: All deferred trice modes are faster in the runtime execution but the Trice logs appear slightly delayed. You can tune the Trices down to only 3 Assembler instructions **executable within 6 processor clocks**. See Trice Speed as example.
- For UART transfer add UART write functionality. The deferred mode is recommended for UART transfer.
- It is possible to log over several channels parallel and to select an ID range for each tag.
- An additional device, like local file, GPIO pin or SPI, is possible by providing an appropriate write functionality.
- See also Trice without UART.

5.9.5. Target Code Overview

- `./src`: **User Interface**

File	description
trice.h	trice runtime lib user interface, `#include trice.h` in project files, where to use Trice macros. Add `./src` to your compiler include path.
`triceConfig.h`	Create this file to overwrite triceDefaultConfig.h as needed.

- `./src`: **Internal Components** (only partially needed, add all to your project - the configuration selects automatically)

File	description
cobs.h	message packaging, alternatively for tcobs
cobsEncode.c	message encoding, alternatively for tcobs
cobsDecode.c	message decoding, normally not needed
trice.c	trice core lib
trice8McuOrder.h	trice MCU endianness lib
trice8McuReverse.h	trice MCU reverse endianness lib
trice16McuOrder.h	trice MCU endianness lib

File	description
trice16McuReverse.h	trice MCU reverse endianness lib
trice32McuOrder.h	trice MCU endianness lib
trice32McuReverse.h	trice MCU reverse endianness lib
trice64McuOrder.h	trice MCU endianness lib
trice64McuReverse.h	trice MCU reverse endianness lib
SEGGER_RTT.h	Segger RTT code interface
SEGGER_RTT.c	Segger RTT code
tcobs.h	message compression and packaging interface
tcobsv1Encode.c	message encoding and packaging
tcobsv1Decode.c	message decoding and packaging, normally not needed
tcobsv1Internal.h	message decoding and packaging internal interface
trice8.h	8-bit trice code interface
trice8.c	8-bit trice code
trice16.h	16-bit trice code interface
trice16.c	16-bit trice code
trice32.h	32-bit trice code interface
trice32.c	32-bit trice code
trice64.h	64-bit trice code interface
trice64.c	64-bit trice code
triceAuxiliary.c	trice code for auxiliary interfaces
triceDefaultConfig.h	This file contains the most probably settings and serves also as a reference for tuning your project *triceConfig.h*
triceDoubleBuffer.c	trice runtime lib extension needed for fastest deferred mode
triceStackBuffer.c	trice runtime lib extension needed for direct mode
triceRingBuffer.c	trice runtime lib extension needed for recommended deferred mode
xtea.h	XTEA message encryption/decryption interface
xtea.c	XTEA message encryption/decryption code

- The *tcobs*.** files are copied from tcobs v1. They are maintained there and extensively tested and probably not a matter of significant change.
- The SEGGER files are copied and you could check for a newer version at https://www.segger.com/downloads/jlink/.

(back to top)

5.9.6. User Code Adaption

- Replace all strings `puts` with the string `trice`, when the string follows immediately. For runtime generated strings see `triceS`.
- Replace all strings `printf` with the string `trice`, when the format string follows immediately.
- Check for float and double format specifiers in the format strings. The appropriate parameters need to be covered with `aFloat()` or `a double()`. Example:

```
printf( "%d, %3.2f EUR, %g rate\n", i, price, change );
```

```
trice64( "%d, %3.2f EUR, %g rate\n", i, aFloat(price), aDouble(change) );
```

  - Because double needs 8 bytes the trice macro in this case needs to be trice64 (see Trice Parameter Bit Widths).

- Check for string format specifiers in the format strings. Put each in a separate trice message. Example:

```
printf( "name: %16s, surname: %32s, birthday: %4u-%02u-%02u\n", n, s, y, m, d);
```

```
trice( "name: %16s, ", n); trice( "surname: %32s, ", s ); trice( "birthday: %4u-%02u-%02u\n" y, m,
d);
```

The Trice macros are designed for maximal execution speed and therefore we have to pay the price for their limited capabilities.

- Optionally add tags to get color. Example:

```
puts( "A message");
```

```
trice( "msg:A message");
```

- Add `#include trice.h` to all user files using trice.

5.9.7. Limitations

- The maximum parameter count per trice is 12, but buffer transfer allows up to 32764 bytes payload. See `triceB` and its relatives.
- Each trice must fit into a single line in trice versions before v0.61.0.
  - Not ok before v0.61.0 but ok for later versions:

```
trice( "hello %u\n",
       year);
```

- But several trices can be in one line.
  - Ok:

```
trice( "hello %u\n", year); trice( "good time");
```

- Strings directly as parameter are possible now.
  - Ok from v0.61.0 with `trice insert` and `trice clean`:

```
triceS( "hello %s\n", "world" );
```

  - Ok always:

```
s = "world"; TRICE_S( "hello %s\n", s );
#define WORLD "world"
triceS( "hello %s\n", WORLD );
```

You should be aware that these parameter strings go into the target and slow down the execution. So, whenever a string is known at compile time it should be part of the Trice format string.

The Trice source code parser has very limited capabilities, so it cannot handle C-preprocessor string concatenation.

- Excluded trices are seen by the trice insert process.
  - Example: The following code will be patched and get an ID as well:

```
// trice( "Hi!" );
```

- All parameters inside one trice have the same bit width. If for example there are a single double and 10 bytes values, the needed trice macro is `trice64` providing 8 bytes space for all parameter values, therefore increasing the transmit overhead. With the default TCOBS framing the overhead is marginal because of the compression. Also this can be handled by splitting into 2 trices:

```
// 92 bytes: 4 bytes header plus 11 times 8 bytes
trice64( "%g: %c%c%c%c%c%c%c%c%c%c", aDouble(3.14159), 61, 62, 63, 64, 65, 66, 67, 68, 69, 10 );

// 24 bytes: 4 bytes header plus 1 times 8 bytes plus 4 bytes header plus 8 times 1 byte
trice64( "%g: ", aDouble(3.14159)); trice8( "%c%c%c%c%c%c%c%c%c%c", 61, 62, 63, 64, 65, 66, 67, 68,
69, 10 );
```

- See also Avoid it.

5.9.8. Trice (Time) Stamps

- Trice messages can have no or 16-bit or 32-bit (time) stamps.
  - recommended (function calling) syntax:

```
trice( "hello %u\n", year); // no (time) stamp
Trice( "hello %u\n", year); // 16-bit (time) stamp
TRice( "hello %u\n", year); // 32-bit (time) stamp
```

  - legacy (inlining) syntax (usable for fastest execution):

```
TRICE( id(0), "hello %u\n", year); // no (time) stamp
TRICE( Id(0), "hello %u\n", year); // 16-bit (time) stamp
TRICE( ID(0), "hello %u\n", year); // 32-bit (time) stamp
```

5.9.9. Trice Parameter Bit Widths

- The macros `trice`, `Trice`, `TRice` and `TRICE` use 32-bit parameter values per default. See `TRICE_DEFAULT_PARAMETER_BIT_WIDTH` inside src/triceDefaultConfig.h to change that.

- If for example the bit width of all trice parameters is 8-bit, it is writable as trice8 macro, reducing the transmitted byte count per parameter from 4 to 1:

```
char b[8] = {1,2,3,4,5,6,7,8};

// 36 bytes: 4 bytes plus 32 (8 times 4) bytes payload
trice( "%02x %02x %02x %02x %02x %02x %02x %02x\n", b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7]);`

// 12 bytes: 4 bytes plus 8 (8 times 1) bytes payload
trice8( " %02x %02x %02x %02x %02x %02x %02x %02x\n", b[0], b[1], b[2], b[3], b[4], b[5], b[6],
b[7]);`
```

```
// 12 bytes: 4 bytes plus 8 (8 times 1) bytes payload in short notation.
triceB( "deb: %02x\n", &b, sizeof(b) );
```

Hint: With the default TCOBS framing 8-bit values as 32-bit parameters typically occupy only 2-bytes during transmission.

5.10. Avoid it

5.10.1. Parser Limitation

Because the implemented source code parser for `trice insert` and `trice clean` is only a simple one, there is one important limitation:

- Do not use an unescaped singe double quote in source code comments. Example:

```
trice( "hi 0" );
// An "allowed" example comment.
trice( "hi 1");
// An \" allowed example comment.
trice( "hi 2");
// A " NOT allowed example comment. This disrupts the parsing.
trice( "hi 3");
// A " NOT allowed example comment. This enables the parsing after a disruption.
trice( "hi 4");
```

- The `trice insert` and `trice clean` will not see the `trice( "hi 3");` line here, but the compiler will mark an error then.
- See also issue #427, issue #465 and see also Limited Trice Parser Capabilities.

5.10.2. Trice macros in header files

- There is nothing wrong, when putting *trice* macros into header files.
- But: When you use `trice insert` as pre-build command and `trice clean` as post build command, those header files get touched on each build and therefore all source code files including them will be re-translated every time.
- For efficiency avoid that.
- **With inventing the Trice Cache this is of no relevance.**

5.10.3. Trice macros inside other macros

There is nothing wrong, when putting Trice macros into other macros. But: When running the self made macro, the location information of the inner *trice* macro will point to the self made macro definition and not to its execution location.

Example: When Functions fnA and fnB are executed, the MY_MESSAGE location information points to *file.h* and not into the appropriate lines inside *file.c*.

file.h:

```
#define MY_MESSAGE trice("msg:Hi\n"); // self made macro
```

file.c:

```
void fnA( void ){
  ...
  MY_MESSAGE
  ...
}

void fnB( void ){
  ...
  MY_MESSAGE
  ...
}
```

5.10.4. Upper case only TRICE macros should be written with id(0), Id(0) or ID(0)

The stamp size 0, 16 or 32 is usually controlled by writing `trice`, `Trice` or `TRICE` or for upper case only Trice macros by using id(0), Id(0) or ID(0). When wrting `TRICE("hi");` for example, the Trice CLI switch `-defaultStampSize` controls the ID insertion, but this is then equal for all new `TRICE` messages.

(back to top)

6. Trice Trouble Shooting Hints

6.1. Initial Data Transfer Setup Hints

If you do not succeed initially, you can try this:

triceConfig.h:

```
#define TriceStamp32 0x44434241 // a fixed value

#define TRICE_DIRECT_OUT_FRAMING TRICE_FRAMING_NONE   // default
#define TRICE_DEFERRED_OUT_FRAMING TRICE_FRAMING_NONE // no framing to interpret the byte stream manually
```

main.c:

```
int main( void) {
    // system init...
    TriceInit();
    TRice(iD(170), "Fun %x!\n", 0xadded ); // with "fixed" iD(170), 32-bit stamp, and with `\n`
    // system run ...
}
```

- Command line with expected output (`-s` is):

```
trice log -s -port com1 -v -ts32="att:%08x fix" # enter this (adapted)
#       /------------------------------------ ID low byte (170)
#       |  /--------------------------------- ID high byte (6 bits=0) with 2 most significant bits set
(32-bit stamp follows)
#       |  |       /--------------------------- 32-bit (time) stamp
#       |  |       |      /-------------------- initial cycle counter: 192
#       |  |       |      |  /----------------- payload size
#       |  |       |      |  |       /--------- payload (0x00added)
#       |  |       |      |  |       |      / - 0-delimiter or next Trice
#       |  |       |      |  |       |      |
#       v  v  vvvvvvvvvv v  v  vvvvvvvvvv v
# Input(aa c0 41 42 43 44 c0 04 ed dd 0a 00 ... ) # expected byte stream
# ...
#              main.c    84 44434241 fix   170 Fun added!
# ...
```

- If you receive s.th. different, you have to debug your system.
- To interpret the bytes see Trice Binary encoding chapter.
  - `33 ff` ID as 16-bit little endian
    - `33` low part of ID 0x3333
    - `ff` high part if ID 0x3333 - the 6 least significant bits ored with 0b11000000 to signal a 32-bit timestamp
  - `41 42 43 44` 32-bit timerstamp, usually as little endian
  - `c0` cycle counter, initial value is 192
  - `04` parameter size
  - `22 22 22 22` 4 parameter bytes

6.2. Short Trouble Shooting Hints

Problem	Hint
Missing `objcopy` in MacOS	`brew install bunutils`
Small GUI Editor for MacOs	`brew install cotedit` Usage: `cot` (not as root)
Small In-Terminal Editor Linux	https://cte.e10labs.com/, tilde, micro, joe, https://craigbarnes.gitlab.io/dte/ (also as root)
Nothing shown with `trice -s`	Check that format strings end with `\n` and/or use `-addNL`

(back to top)

7. Trice Cache for Compilation Speed

The `trice insert` and `trice clean` commands are parsing and modifying the source code files. Even this is a reasonable fast procedure, this could get time consuming on large projects, especially when using these commands as permanent pre-compile and post-compile steps. It is assumed, that usually between 2 compile steps not all project files are changed. The project files majority will stay unchanged despite the ID insertion and removal. This repeated parsing and modifying of unchanged source code is avoidable with the Trice cache technique. Also it could get annoying to recompile files all the time only because they got Trice IDs removed and inserted. With the Trice cache we get also a solution not to re-compile un-edited files as well.

7.1. Trice Cache Idea

Lets talk about just one source file `$HOME/my/src/foo.c` and imagine we process many in one shot.

- On `trice insert foo.c`, get full path of `foo.c`, then: If `.trice/cache/cleaned/home/my/src/foo.c` exists and has the same modification time as `/home/my/src/foo.c`, copy `.trice/cache/inserted/home/my/src/foo.c` (if existing) to `/home/my/src/foo.c`. Otherwise insert IDs into `/home/my/src/foo.c` and afterwards copy it to `.trice/cache/inserted/home/my/src/foo.c`.
- On `trice clean foo.c`, get full path of `foo.c`, then: If `.trice/cache/inserted/home/my/src/foo.c` exists and has the same modification time as `/home/my/src/foo.c`, copy `.trice/cache/cleaned/home/my/src/foo.c` (if existing) to `/home/my/src/foo.c`. Otherwise remove IDs from `/home/my/src/foo.c` and copy it to `.trice/cache/cleaned/home/my/src/foo.c`.
- On any repeated or alternate `trice insert` and `trice clean`, we are done.
- When a file in cleaned or inserted ID state was edited somehow, its IDs are inserted/cleaned and the cache is updated accordingly on `trice clean` or `trice insert` because the file modification time has changed.

7.2. Trice Cache Logic

When `id.TriceCacheEnabled` is true (applied `-cache` CLI switch) and the folder `~/.trice/cache` exists, we have

- optionally a *cleaned cache file* `~/.trice/cache/cleaned/fullpath/file` with mtime of *IDs cleaned*
- optionally an *inserted cache file* `~/.trice/cache/inserted/fullpath/file` with mtime of *IDs inserted*
- `fullpath/file` with mtime of *IDs cleaned* **OR** *IDs inserted* **OR** *last edit*. When mtime of `path/file` is:
  - *IDs cleaned*:
    - On command `trice c`, nothing to do
    - On command `trice i`, copy, if existing, *inserted cache file* into `fullpath/file`. Otherwise process `trice i` and copy result into *inserted cache file*.
  - *IDs inserted*:
    - On command `trice c`, copy, if existing, *cleaned cache file* into `fullpath/file`. Otherwise process `trice c` and copy result into *cleaned cache file*.
    - On command `trice i`, nothing to do
  - *last edit*:
    - On command `trice c`, invalidate cache, process `trice c` and update *cleaned cache file*, file gets a new mtime, the mtime of *IDs cleaned*. On a following command `trice i`, file mtime is *IDs cleaned*, BUT the cache is invalid, so process `trice i` and update cache/inserted.
    - On command `trice i`, invalidate cache, process `trice i` and update *inserted cache file*, file gets a new mtime, the mtime of *IDs inserted*. On a following command `trice c`, file mtime is *IDs inserted*, BUT the cache is invalid, so process `trice c` and update cache/cleaned.

7.3. Trice Cache Remarks

- `fullpath/file` means `/home/me/proj3/file` for example. When copied to the cache, the real "fullpath" is there `/home/me/.trice/cache/cleaned/home/me/proj3/file`.

> Should the `.trice/cache` be better located inside the project folder? What, if the user has several projects and several users on the same machine working on projects together? What about libraries containing trice code?

- The `~/.trice/cache` folder should the Trice tool **not** create automatically in the users home folder `$HOME`. The existence of this folder is user controlled. The folder must exist. If several users work on the same project and some use the cache and some not - it is is possible this way, even build scripts are shared.
- The `~/.trice/cache` folder should **not** go under revision control.
- A CLI switch `-cache` does enable/disable the Trice cache. Default is off.
- The user should consider what happens, if other pre-compile or post-compile steps are modifying files as well, before enabling the Trice cache.

7.4. Trice Cache Tests

Nr	Action	cCache	iCache	ID state	Edid state	Test function
0,1	0:clean	0:inval	0:inval	0:cleaned	X:any	Test_0_1_0000X_clean_on_invalid_cCache_invalid_iCache_cleaned_file
2,3	0:clean	0:inval	0:inval	1:inserted	X:any	Test_2_3_00011_clean_on_inalid_cCache_invalid_iCache_inserted_edited_file
4,5	0:clean	0:inval	1:valid	0:cleaned	X:any	Test_4_5_0010X_clean_on_invalid_cCache_valid_iCache_cleaned_file
6	0:clean	0:inval	1:valid	1:inserted	0:not	Test_6_00110_clean_on_invalid_cCache_valid_iCache_inserted_not_edited_file
7	0:clean	0:inval	1:valid	1:inserted	1:yes	Test_7_00111_clean_on_invalid_cCache_valid_iCache_inserted_edited_file
8	0:clean	1:valid	0:inval	0:cleaned	0:not	Test_8_01000_clean_on_valid_cCache_invalid_iCache_cleaned_not_edited_file
9	0:clean	1:valid	0:inval	0:cleaned	1:yes	Test_9_01001_clean_on_valid_cCache_invalid_iCache_cleaned_edited_file
10	0:clean	1:valid	0:inval	1:inserted	0:not	Test_10_01011_clean_on_valid_cCache_invalid_iCache_inserted_not_edited_file
11	0:clean	1:valid	0:inval	1:inserted	1:yes	Test_11_01011_clean_on_valid_cCache_invalid_iCache_inserted_edited_file
12	0:clean	1:valid	1:valid	0:cleaned	0:not	Test_12_01100_clean_on_valid_iCache_valid_cCache_clean_file_not_edited
13	0:clean	1:valid	1:valid	0:cleaned	1:yes	Test_13_01101_clean_on_valid_iCache_valid_cCache_clean_file_edited
14	0:clean	1:valid	1:valid	1:inserted	0:not	Test_14_01110_clean_on_valid_iCache_valid_cCache_inserted_file_not_edited
15	0:clean	1:valid	1:valid	1:inserted	1:yes	Test_15_01111_clean_on_valid_iCache_valid_cCache_inserted_file_edited
16,17	1:insert	0:inval	0:inval	0:cleaned	X:any	Test_16_17_1000X_insert_on_invalid_cCache_invalid_iCache_cleaned_file
18,19	1:insert	0:inval	0:inval	1:inserted	X:any	Test_18_19_1001X_insert_on_invalid_cCache_invalid_iCache_inserted_edited_file
20,21	1:insert	0:inval	1:valid	0:cleaned	X:any	Test_20_21_1010X_insert_on_invalid_cCache_valid_iCache_cleaned_file
22	1:insert	0:inval	1:valid	1:inserted	0:not	Test_22_10100_insert_on_invalid_cCache_valid_iCache_inserted_not_edited_file
23	1:insert	0:inval	1:valid	1:inserted	1:yes	Test_23_10101_insert_on_invalid_cCache_valid_iCache_inserted_edited_file
24	1:insert	1:valid	0:inval	0:cleaned	0:not	Test_24_11000_insert_on_valid_cCache_invalid_iCache_cleaned_not_edited_file
25	1:insert	1:valid	0:inval	0:cleaned	1:yes	Test_25_11001_insert_on_valid_cCache_invalid_iCache_cleaned_edited_file
26,27	1:insert	1:valid	0:inval	1:inserted	X:any	Test_26_27_1010X_insert_on_invalid_cCache_valid_iCache_cleaned_file
28	1:insert	1:valid	1:valid	0:cleaned	0:not	Test_28_11100_insert_on_valid_cCache_valid_iCache_cleaned_not_edited_file
29	1:insert	1:valid	1:valid	0:cleaned	1:yes	Test_29_11100_insert_on_valid_cCache_valid_iCache_cleaned_edited_file
30	1:insert	1:valid	1:valid	1:inserted	0:not	Test_30_11110_insert_on_valid_cCache_valid_iCache_inserted_not_edited_file
31	1:insert	1:valid	1:valid	1:inserted	1:yes	Test_31_11111_insert_on_valid_cCache_valid_iCache_inserted_edited_file

7.5. Possible Trice Cache Editor-Issues And How To Get Around

- When a `trice i -cache && make && trice c -cache` sequence is executed, it could happen that the editor-view is not refreshed for opened and unedited files containing Trice statements.
  - It looks like the Trice IDs were not cleaned.
  - Closing and opening the file again shows, that the Trice IDs are indeed cleaned.
  - If the file is edited then without refreshing the view, that means with the shown Trice IDs, this is no problem, because after saving the edited file, it gets processed anyway, so no data loss is possible.
  - An automatic view refresh (close & open) for the editor could help here. But how to do that in an universal way?
- A workaround is, at least for vsCode, to first run `trice clean` in the build script.
  - See examples/G1B1_inst/build.sh for an implementation.

7.6. Activating the Trice Cache

- Create Trice cache folder:

```
mkdir -p ~/.trice/cache
```

- Apply `-cache` CLI switch on `trice insert` and `trice clean`. See ./trice_insertIDs_in_examples_and_test_folder.sh and ./trice_cleanIDs_in_examples_and_test_folder.sh which both call ./trice_environment.sh and used for example in ./examples/G0B1_inst/build.sh
- Do **NOT** add the Trice cache to the version control.
- It is safe to `rm -rf ~/.trice/cache` and not to use the `-cache` CLI switch anymore.

(back to top)

8. Embedded system code configuration

Check comments inside triceDefaultConfig.h and adapt your project configuration like shown in triceConfig.h as example.

A Trice macro is avoiding all the `printf()` internal overhead (space and time) but is nearly as easy to use. For example instead of writing

```
printf("time is %d:%d:%d\n", hour, min, sec);
```

you can write

```
trice8("time is %d:%d:%d\n", hour, min, sec);
```

into a source file of your project. The `8` stands here for 8 bit values (`16`, `32` and `64` also possible). Values of mixed size up to 32-bit size are allowed in one `trice` macro, so you can use Trice consequently to match most cases for the prize of little data overhead.

(back to top)

---

9. Trice tool in logging action

With `trice log -port COM12` you can visualize the trices on the PC, if for example `COM12` is receiving the data from the embedded device at the 115200 default baudrate.

The following capture output comes from an (old) example project inside ../examples.

```
Mar 12 18:59:29.261077: ISR:alive time 22816660 milliseconds
Mar 12 18:59:29.262078: ISR:alive time 22816670 milliseconds
Mar 12 18:59:29.264079: ISR:alive time 22816680 milliseconds
Mar 12 18:59:29.284098: ISR:alive time 22816690 milliseconds
Mar 12 18:59:29.286099: ISR:alive time 22816700 milliseconds
Mar 12 18:59:29.288101: timing      message, SysTick is 33534 @ 22816706 ms
Mar 12 18:59:29.290103: Hello world this is a 1st printf replacement.
Mar 12 18:59:29.291104: 12345 as 16bit is 0b0011000000111001
Mar 12 18:59:29.293106: ------------------------------------------------------------------------------------------

Mar 12 18:59:29.294107: This ASSERT error is just a demo and no real error: triceCheck.c line 27 ERR:ASSERT failed
Mar 12 18:59:29.297109: ------------------------------------------------------------------------------------------

Mar 12 18:59:29.318129: TRICE8  %03x ->  001  07f  -80  -01
Mar 12 18:59:29.320131: TRICE8   %4d ->    1  127 -128   -1
Mar 12 18:59:29.321131: TRICE8   %4o ->    1  177 -200   -1
Mar 12 18:59:29.323133: TRICE16 %05x ->   00001   07fff   -8000   -0001
Mar 12 18:59:29.325135: TRICE16  %6d ->       1   32767  -32768      -1
Mar 12 18:59:29.327137: TRICE16  %7o ->       1   77777 -100000      -1
Mar 12 18:59:29.329139: TRICE32_4 %09x ->      000000001      07fffffff       -80000000     -00000001
Mar 12 18:59:29.351158: TRICE32_4 %10d ->              1     2147483647     -2147483648           -1
Mar 12 18:59:29.354161: 64bit 0b1000100100010001100110100010001010101011001100111011110001000
Mar 12 18:59:29.356163: TRICE8 -1
Mar 12 18:59:29.358165: TRICE8 -1 -2
Mar 12 18:59:29.359166: TRICE8 -1 -2 -3
Mar 12 18:59:29.360167: TRICE8 -1 -2 -3 -4
Mar 12 18:59:29.362168: TRICE8 -1 -2 -3 -4 -5
Mar 12 18:59:29.363169: TRICE8 -1 -2 -3 -4 -5 -6
Mar 12 18:59:29.385189: TRICE8 -1 -2 -3 -4 -5 -6 -7
Mar 12 18:59:29.387191: TRICE8 -1 -2 -3 -4 -5 -6 -7 -8
Mar 12 18:59:29.388192: TRICE16 -1
Mar 12 18:59:29.390194: TRICE16 -1 -2
Mar 12 18:59:29.391195: TRICE16 -1 -2 -3
Mar 12 18:59:29.392196: TRICE16 -1 -2 -3 -4
Mar 12 18:59:29.394198: TRICE32 -1
Mar 12 18:59:29.395199: TRICE32 -1 -2
Mar 12 18:59:29.397200: TRICE32 -1 -2 -3
Mar 12 18:59:29.418220: TRICE32 -1 -2 -3 -4
Mar 12 18:59:29.420221: TRICE64 -1
Mar 12 18:59:29.421222: TRICE64 -1 -2
Mar 12 18:59:29.423224: ERR:error       message, SysTick is 21653
Mar 12 18:59:29.425226: WRN:warning     message, SysTick is 21637
Mar 12 18:59:29.427228: ATT:attension   message, SysTick is 21619
Mar 12 18:59:29.429230: DIA:diagnostics message, SysTick is 21607
Mar 12 18:59:29.451249: TIM:timing      message, SysTick is 21585
Mar 12 18:59:29.453251: DBG:debug       message, SysTick is 21567
Mar 12 18:59:29.455253: SIG:signal      message, SysTick is 21550
Mar 12 18:59:29.457255: RD_:read        message, SysTick is 21532
Mar 12 18:59:29.458256: WR_:write       message, SysTick is 21515
Mar 12 18:59:29.460258: TST:test        message, SysTick is 21497
Mar 12 18:59:29.462260: MSG:normal      message, SysTick is 21480
Mar 12 18:59:29.464261: error       message, SysTick is 21462
Mar 12 18:59:29.485280: warning     message, SysTick is 21445
Mar 12 18:59:29.487282: attension   message, SysTick is 21427
Mar 12 18:59:29.489284: diagnostics message, SysTick is 21406
Mar 12 18:59:29.491286: timing      message, SysTick is 21390
Mar 12 18:59:29.493288: debug       message, SysTick is 21372
Mar 12 18:59:29.495290: signal      message, SysTick is 21355
Mar 12 18:59:29.497291: read        message, SysTick is 21337
Mar 12 18:59:29.518311: write       message, SysTick is 21320
Mar 12 18:59:29.520312: test        message, SysTick is 21302
Mar 12 18:59:29.522314: normal      message, SysTick is 21285
Mar 12 18:59:29.523315: ABC
Mar 12 18:59:29.525317:
Mar 12 18:59:29.526317:                1
Mar 12 18:59:29.527319:               12
Mar 12 18:59:29.528319:              123
Mar 12 18:59:29.529320:             1234
Mar 12 18:59:29.529320:            12345
Mar 12 18:59:29.530321:           123456
Mar 12 18:59:29.550339:          1234567
Mar 12 18:59:29.551341:         12345678
Mar 12 18:59:29.552341:        123456789
Mar 12 18:59:29.553342:        123456789a
Mar 12 18:59:29.554343:       123456789ab
Mar 12 18:59:29.555344:      123456789abc
Mar 12 18:59:29.556345:        MSG:Berlin
Mar 12 18:59:29.557346: ------------------------------------------------------------------------------------------

Mar 12 18:59:29.558347: timing      message, SysTick is 9383 @ 22816706 ms
Mar 12 18:59:29.560348: ISR:alive time 22816710 milliseconds
Mar 12 18:59:29.562350: ISR:alive time 22816720 milliseconds
Mar 12 18:59:29.564352: ISR:alive time 22816730 milliseconds
```

See ../_test/testdata/triceCheck.c for reference. The *Trices* can come mixed from inside interrupts (light blue `ISR:...`) or from normal code. For usage with a RTOS, *Trices* are protected against breaks (`TRICE_ENTER_CRITICAL_SECTION`, `TRICE_LEAVE_CRITICAL_SECTION`). Regard the differences in the read SysTick values inside the GIF above These differences are the MCU clocks needed for one trice (~0,25µs@48MHz).

Use the `-color off` switch for piping output in a file. More convenient is the `-lf auto` switch.

(back to top)

10. Encryption

- You can deliver your device with encrypted trices. This way only the service [wo]men is able to read the *Trices*.
- Implemented is XTEA but this is exchangeable.
- The to 8 byte padded blocks can get encrypted by enabling `#define ENCRYPT...` inside triceConfig.h. You need to add `-password MySecret` as `trice log` switch and you're done.
- Any password is usable instead of `MySecret`. Simply add once the `-show` switch and copy the displayed passphrase into the triceConfig.h file.
- The encryption takes part **before** the COBS encoding.
- TCOBS is usable but not recommended after encryption, because it cannot compress effective arbitrary data.

(back to top)

11. Trice Command Line Interface & Examples

The trice tool is very easy to use even it has a plenty of options. Most of them normally not needed. The trice tool can be started in several modes (sub-commands), each with several mandatory or optional switches. Switches can have a single parameter string or not.

```
trice sub-command -switch1 -switch2 parameter -switch3 ...
```

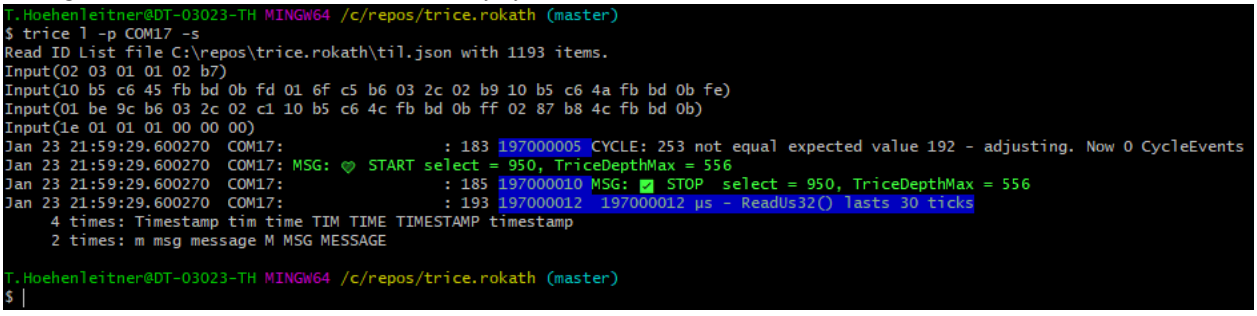
Which sub-command switches are usable for each sub-command is shown with `trice help -all`. This gives also information about their default values.

Info for a special sub-command is shown with `trice help -log` for example.

- The command history is usable for example inside the bash, simply enter CTRL-R and start typing `trice...` and you can select from the history.
- The most convenient way is to use trice inside scripts like in this example.

11.1. Common information

- `trice h -all` shows all options of the current version.
- `trice ver` prints version information.
- `trice s` shows you all found serial ports for your convenience.
- `trice l -p COM17` could fail if something is wrong. Additional switches are for help tracking the issue:
    - Use log witch `-s[howInputBytes]` to check if any bytes are received at all.



- With `-debug` you can see the [T]COBS packages and decoded Trice packages.

- `trice i` in the root of your project parses all source files for Trice macros, adds automatically ID´s if needed and updates a file named **til.json** containing all ID´s with their format string information. To start simply generate an empty file named **til.json** in your project root. You can add `trice i` to your build process and need no further manual execution.

- `trice ds` starts a display server listening on default ip address *127.0.0.1:61487* or any specified value. This is possible also on a remote device, lets say with ip address 192.168.1.200.

- `trice l -p COM18 -ds` sends the log strings to a display server with default ip address *127.0.0.1:61487* or any specified value, if for example `-ipa 192.168.1.200` the trice logs go to the remote device. You can start several trice log instances, all transmitting to the same display server.

11.2. Further examples

11.2.1. Automated pre-build insert command example

- Scan directories `../src`, `../lib/src` and `./` to insert the IDs there and extend list file `../../../til.json`

```
trice i -v -i ../../../til.json -src ../src -src ../lib/src -src ./
```

This is a typical line you can add to your project as an automatic pre-compile step.

11.2.2. Some Log examples

- Log trice messages on COM3 8N1 115200 baud

```
trice log -i ./myProject/til.json -p=COM3
```

- Log trice messages on COM3 8N1 9600 baud and use default til.json

```
trice l -s COM3 -baud=9600
```

11.2.3. Logging over a display server

- Start displayserver on ip 127.0.0.1 (localhost) and port 61497

```
trice ds
```

- Log trice messages on COM3 and display on display server

```
trice l -ds -p COM3
```

- Shutdown remote display server on IP 192.168.1.23 port 45678

```
trice sd -r 192.168.1.23:45678
```

The IP address and port are free selectable. Using a display server, allows to watch the logs of one or many MCUs on a local or remote machine with the same or different display servers.

A local Trice instance sends Trice messages to a display server only, when a log line is complete (if consisting of several Trices). By using the CLI switches `-prefix` and `-suffix` you can decorate the loglines target specific to distinguish them in the output window(s).

##13.2.4. Logfile output

```
trice l -p COM3 -logfile auto
```

This creates a new logfile `2022-05-16_2216-40_trice.log` with the actual timestamp on each Trice start.

```
trice l -p COM3 -logfile trice.log
```

This creates a new logfile `trice.log` on first start and appends to it on each next Trice start.

Logfiles are text files one can see with 3rd party tools. Example: `cat trice.log`. They contain also the PC reception timestamps if where enabled.

11.2.4. Binary Logfile

```
trice l -p COM3 -binaryLogfile auto
```

This creates a new binary logfile `2022-05-16_2216-40_trice.bin` with the actual timestamp on each Trice start.

```
trice l -p COM3 -binaryLogfile trice.bin
```

This creates a new binary logfile `trice.bin` on first start and appends to it on each next Trice start.

Binary logfiles store the Trice messages as they come out of the target in binary form. They are much smaller than normal logfiles, but the Trice tool with the *til.json* is needed for displaying them and the PC timestamps are the displaying time: `trice l -p FILEBUFFER -args trice.log`.

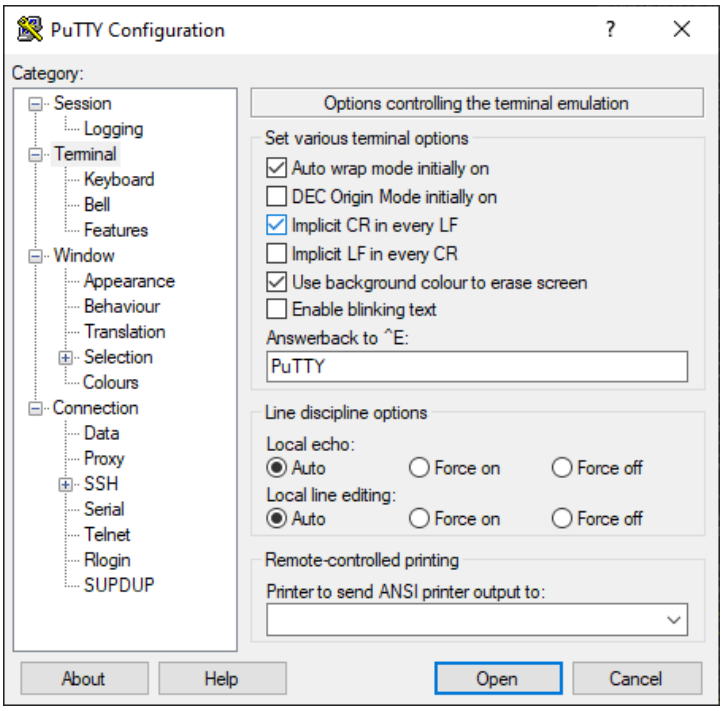
Binary logfiles are handy in the field for long data recordings.

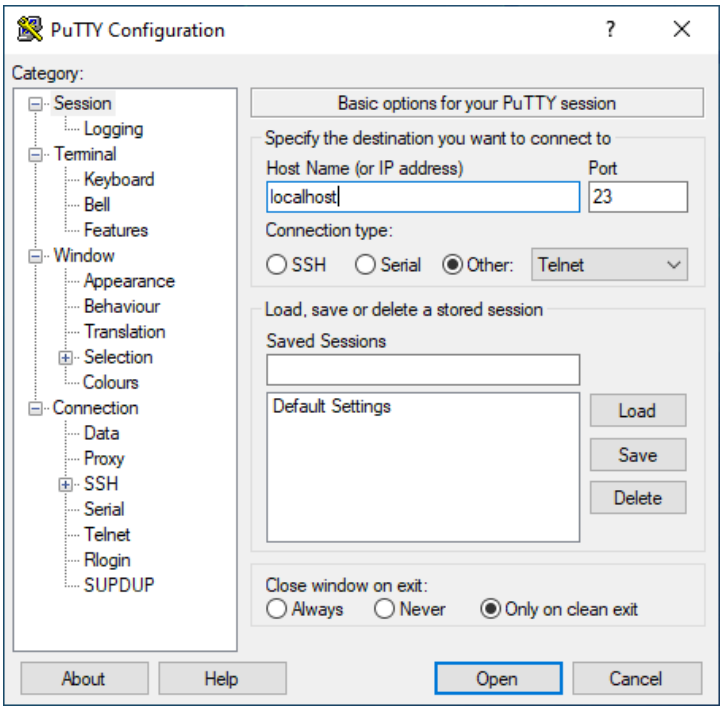
When using RTT, the data are exchanged over a file interface. These binary logfiles are stored in the project [./temp] folder and accessable for later view: `trice l -p FILEBUFFER -args ./temp/logfileName.bin`. Of course the host timestamps are the playing time then.

11.2.5. TCP output

```
trice l -p COM3 -tcp 127.0.0.1:23
```

This additionally sends Trice output to a 3rd party TCP listener, for example like Putty:


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
SSH
Serial
Telnet
Rlogin
SUPDUP
Options controlling the terminal emulation
Set various terminal options
Auto wrap mode initially on
DEC Origin Mode initially on
Implicit CR in every LF
Implicit LF in every CR
Use background colour to erase screen
Enable blinking text
Answerback to ^E:
PuTTY
Line discipline options
Local echo:
Auto
Force on
Force off
Local line editing:
Auto
Force on
Force off
Remote-controlled printing
Printer to send ANSI printer output to:
About
Help
Open
Cancel


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
SSH
Serial
Telnet
Rlogin
SUPDUP
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
localhost
23
Connection type:
SSH
Serial
Other:
Telnet
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Help
Open
Cancel

11.2.6. TCP4 input

```
trice l -p TCP4 -args "192.168.2.3:45678"
```

This expects a TCP4 server at IP address `192.168.2.3` with port number `45678` to read binary Trice data from.

11.2.7. UDP4 input (accepted pull request 529)

The pull request #529 introduces key enhancement:

```
    IPv4 UDP Receiver
    Adds support for receiving data over IPv4 using UDP. This enables integration with systems that
broadcast or transmit telemetry, logs, or other messages over the network.
```

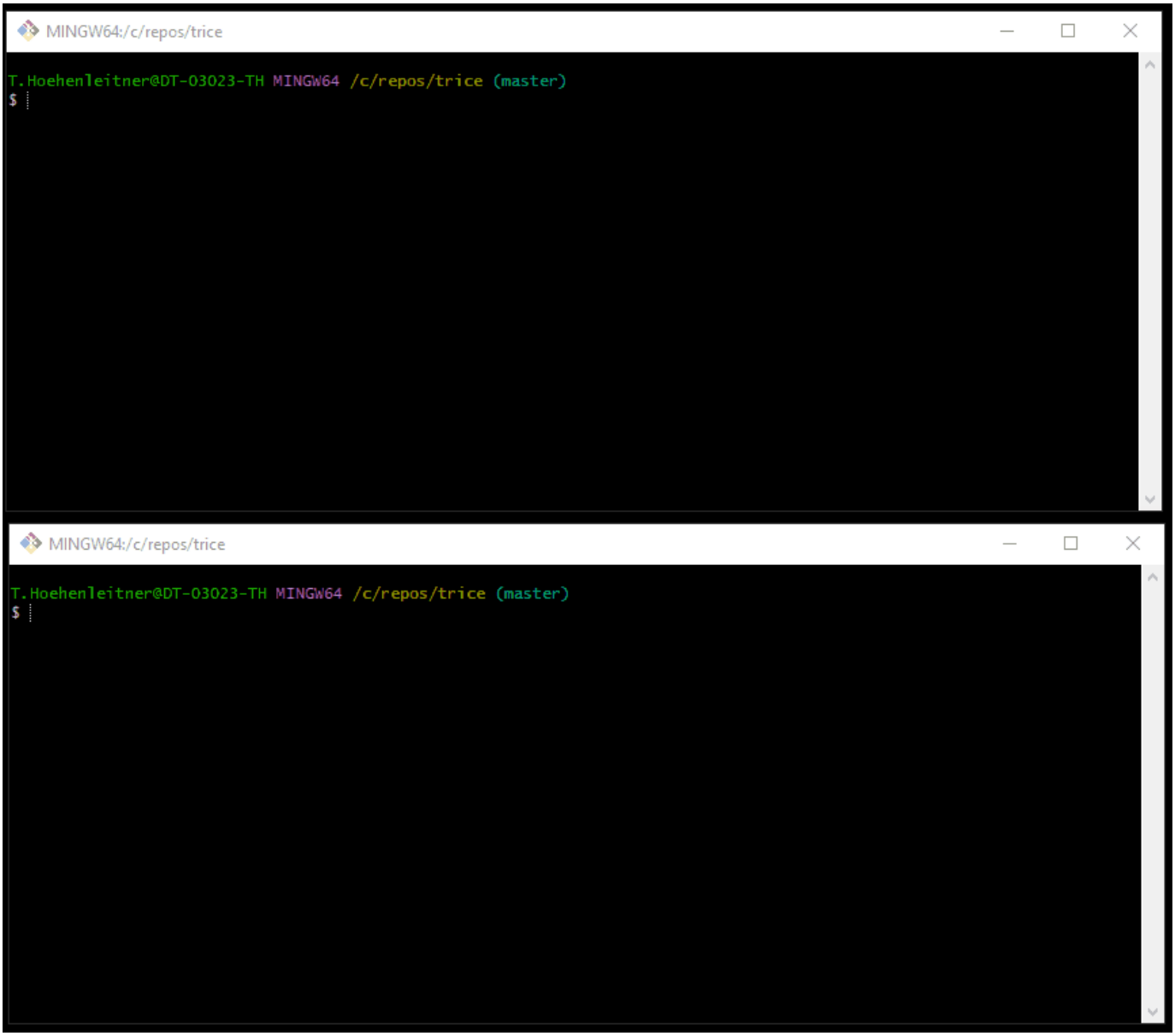
-port UDP4 Example

To receive Trice logs over IPv4 UDP, use the -port UDP4 option. By default, it listens on 0.0.0.0:17005, which accepts packets on all network interfaces. You can specify a different address or multicast group via -args.

trice log -p UDP4

11.2.8. Stimulate target with a user command over UART

Sometimes it is handy to stimulate the target during development. For that a 2nd screen is helpful what is possible using the display server option:



11.2.9. Explpore and modify tags and their colors

See chapter Trice Tags and Color.

11.2.10. Location Information

When running `trice insert`, a file `li.json` is created, what you can control with the `-li|locationInformation` switch. During logging, when `li.json` is found, automatically the filename and line number is displayed in front of each log line, controllable with the `-liFmt`

switch. This information is correct only with the right version of the `li.json` file. That is usually the case on the PC during development. Out in the field only the `til.json` reference is of importance. It serves as an accumulator of all firmware versions and usually the latest version of this file is the best fit. The `li.json` file should stay with the software developer only and needs no version control in the usual case because it is rebuild with each compilation, when `trice i` is a prebuild step. When `trice clean` is used, the file `li.json` should go into the version management too to secure that identical trices get the same ID back.

(back to top)

12. Limitations

12.1. Permanent Limitations

12.1.1. Limitation TRICE in TRICE not possible

- No-Good Example:

```
int f0( void ){ TRICE( "msg:f0\n"); return 0; }
void f1( void ){ TRICE( "No; %d", f0() ); }
```

- This will compile normally but corrupt TRICE output.

The reason is: When f1() gets active, the "No" Trice header is created, than the f0() Trice is executed and afterwards the "No" Trice tail is written. This works well during compile time but causes a mismatch during runtime.

- Workaround:

```
int f0( void ){ TRICE( "msg:f0\n"); return 0; }
void f1( void ){ int x = f0(); TRICE( "Yes: %d", x ); }
```

12.2. Current Limitations

12.2.1. String Concatenation Within TRICE Macros Not Possible

String concatenation within TRICE macros does not work. The reason lays inside the way the trice tool parser works:

```
void f0( void ){ TRICE( "msg:" ## "Hello\n" ); } // ERROR!
```

To implement this would need to build a trice preprocessor or to run the C preprocessor first and to modify the preprocessor output with the trice tool. That would make things unneccessary complicate and fragile for now.

12.2.2. Limited Trice Parser Capabilities

The Trice tool internal parser has only limited capabilities. In works well in most cases, but could led to problems in some cases. The compiler run will for sure end up with some error messages in the following examples, so the developer can fix the code.

An example, provided by @KammutierSpule, is this:

- started from a empty li.json/til.json

```
void trice0_test() {
    Trice0( "OK");
    Trice( InvalidUse );
    Trice( "%u", Variable );
}
```

- run `trice insert`

```
void trice0_test() {
    Trice0( iD(2740), "OK"); // ok, iD is added
    Trice( InvalidUse ); // no warning or error
    Trice( "%u", Variable ); // id is not added / inserted
}
```

As said, the compiler will complain about that in any case.

12.2.3. Special Care Demands

More than 12 printf parameters

- use several printf-calls
- Use triceB and its relatives

Float Numbers

- surround each with `aFloat()`

Double numbers

- surround each with `aDouble()` and use the `trice64` macro and relatives

Runtime Generated Strings

- Each needs its own `triceS` macro, example:
    - Legacy code:

      ```
      printf( "Entered name is %20s %30s, favorite numbers %d, %f\n", "Paul", "Luap", 42, 3.14159 );
      ```

    - Trice code:

      ```
      name = "Paul"; triceS( "Entered name is %20s", name );`
      surname = "Luap";  triceS( " %30s, ", surname );`
      trice( "favorite numbers %d, %f\n", 42, aFloat(3.14159) );`
      ```

The `triceS` macro is ment to be used with strings not known at compile time.

Usage intention and recommendation: (given by @escherstair)

```
char runtime_string[50];
fillRuntimeStringFromSomewhere(runtime_string); // the content of runtime_string is filled at run time
triceS( "msg:This part of the string is known at compile time. This part is dynamic: %s\n",
runtime_string);
```

All the string literals (i.e. compile-time known strings) should be put inside the format string. Only the runtime strings should be used as variables in triceS macro for best performance.

(back to top)

13. Additional hints

13.1. Pre-built executables are available

See https://github.com/rokath/trice/releases.

13.2. Configuration file triceConfig.h

- When setting up your first project you need a `triceConfig.h` file.
- You should **not** use the `./_test/cgo.../triceConfig.h` directly, because these are customized for internal tests with CGO. But you can use their settings as helper for a starting point.
- Please choose one of these files as starting point:
  - ../examples/F030_inst/Core/Inc/triceConfig.h
  - ../examples/G0B1_inst/Core/Inc/triceConfig.h
  - ../examples/L432_inst/Core/Inc/triceConfig.h
- Comparing them and understandig the differences helps quick starting.
- The file triceDefaultConfig.h contains all possible config keys with descriptions.

13.3. Setting up the very first connection

If you see nothing in the beginning, what is normal 😉, add the `-s` (`-showInputBytes`) switch to see if any data arrive. There is also a switch `-debug` showing you the received packages, if you are interested in.

13.4. Avoid buffer overruns

It is your responsibility to produce less data than transmittable. If this is not guarantied, a data loss is not avoidable or you have to slow down the user application. The buffers have an optional overflow protection (`TRICE_PROTECT`), which is enabled by default. Recommendation: Make the buffer big and emit the maxDepth cyclically, every 10 or 1000 seconds. Then you know the needed size. It is influenced by the max Trice data burst and the buffer switch interval. See ./examples/exampleData/triceLogDiagData.c for help.

If the target application produces more Trice data than transmittable, a buffer overrun can let the target crash, because for performance reasons no overflow check is implemented in versions before v0.65.0. Such a check is added now per default using `TRICE_PROTECT`, but the Trice code can only throw data away in such case. Of course you can disable this protection to get more speed.

Configuring the ring buffer option with `TRICE_PROTECT == 0` makes buffer overruns not completely impossible, because due to partial Trice log overwrites, false data are not excluded anymore and overwriting the buffer boundaries is possible, because of wrong length information. Also losses will occur when producing more data than transmittable. This is detectable with the cycle counter. The internal 8-bit cycle counter is usually enabled. If Trice data are lost, the receiver side will detect that because the cycle counter is not as expected. There is a chance of 1/256 that the detection does not work for a single case. You can check the detection by unplugging the trice UART cable for a time. Also resetting the target during transmission should display a cycle error.

Gennerally it is recommended to enable `TRICE_PROTECT` during development and to disable it for performance, if you are 100% sure, that not more data are producable than transmittable.

Important to know: If the `TRICE_PROTECT` code inhibits the writing into a buffer, there will be later no cycle error because a non existing Trice cannot cause a cycle error. Therefore the `TriceDirectOverflowCount` and `TriceDeferredOverflowCount` values exist, which could be monitored.

13.5. Buffer Macros

(Examples in ../_test/testdata/triceCheck.c)

Macro Name	Description
`triceS` |`TriceS` |`TRiceS` |`TRICE_S`	Output of runtime generated 0-terminated strings.
`triceN` |`TriceN` |`TRiceN` |`TRICE_N`	Is for byte buffer output as string until the specified size. It allows limiting the string size to a specific value and does not rely on a terminating 0. If for example len = 7 is given and "Hello\0World\n" is in the buffer, the byte sequence "Hello\0W" is transmitted but the trice tool probably shows only "Hello".
`trice8B` |`Trice8B` |`TRice8B` |`TRICE8_B`	Is for byte buffer output according to the given format specifier for a single byte.
`trice16B`|`Trice16B`|`TRice16B`|`TRICE16_B`	Is for 16-bit buffer output according to the given format specifier for a 16-bit value.
`trice32B`|`Trice32B`|`TRice32B`|`TRICE32_B`	Is for 32-bit buffer output according to the given format specifier for a 32-bit value.
`triceB` |`TriceB` |`TRiceB` |`TRICE_B`	Is buffer output according to the given format specifier for a default unit according to configuration (8|16|32|64-bit value) - default is `#define TRICE_B TRICE8_B`.

13.6. Logfile viewing

Logfiles, Trice tool generated with sub-command switch `-color off`, are normal ASCII files. If they are with color codes, these are ANSI escape sequences.

- Simply `cat trice.log`. One view option is also `less -R trice.log`. The Linux command `less` is also available inside the windows git bash.
- Under Windows one could also download and use ansifilter for logfile viewing. A monospaced font is recommended.
- See also Color issues under Windows

13.7. Using the Trice tool with 3rd party tools

Parallel output as logfile, TCP or binary logfile is possible. See examples above.

13.8. Several targets at the same time

You can connect each target over its transmit channel with an own Trice instance and integrate all transmissions line by line in an additional Trice instance acting as display server. See https://github.com/rokath/trice#display-server-option.

The C-code is executed during some tests. Prerequisite is an installed GCC.

13.9. TRICE_STACK_BUFFER could cause stack overflow with -o0 optimization

As discussed in issue #294 it can happen, that several TRICE macros within one function call increase the stack usage more than expected, when compiler optimization is totally switched off.

13.10. Cycle Counter

- The trice tool expects the first cycle counter to start with 0xC0 (=192). If the target is already running and you connect the trice tool then, the first message is marked with "CYCLE: ? not equal expected value 192 - adjusting. Now 1 CycleEvents".
- If the target is resetted asynchronous, the trice tool receives a cycle counter 192. Most probably the last cycle counter was not 191, so this triggers also a message with "CYCLE: 192 not equal expected value ?- adjusting. Now n CycleEvents".
- In the Trice tool is some heuristics to suppress such obvious false positives.

(back to top)

14. Switching Trice ON and OFF

14.1. Target side compile-time Trice On-Off

- If your code works well after checking, you can add `#define TRICE_OFF 1` just before the `#include "trice.h"` line and no Trice code is generated anymore for that file, so no need to delete or comment out Trice macros:

```
#define TRICE_OFF 1
#include "trice.h"
void fn(void) {
    trice( iD(123), "Hi"); // Will generate code only, when TRICE_OFF == 0.
    trice( "Lo");          // Will generate code only, when TRICE_OFF == 0.
}
```

With `#define TRICE_OFF 1`, macros in this file are ignored completely by the compiler, but not by the Trice tool. In case of re-constructing the **T**rice **ID** **L**ist these no code generating macros are regarded and go into (or stay inside) the ID reference list.

- Hint from @escherstair: With `-D TRICE_OFF=1` as compiler option, the trice code diappears completely from the binary.
- No runtime On-Off switch is implemented for several reasons:
  - Would need a control channel to the target.
  - Would add little performance and code overhead.
  - Would sligtly change target timing (testing).
  - User can add its own switches anywhere.
  - The short Trice macro code is negligible.
  - The trice output is encryptable, if needed.
- Because of the low Trice bandwidth needs and to keep the target code as clear as possible the runtime On-Off decision should be done by the Trice tool.

(back to top)

14.2. Host side Trice On-Off

- The PC Trice tool offers command line switches to `-pick` or `-ban` for Trice tags and will be extended with display switches.
- A Trice tool `-logLevel` switch is usable too.

(back to top)

15. Framing

- Trice messages are framed binary data, if framing is not disabled.
- Framing is important for data disruption cases and is done with TCOBS (has included data compression) but the user can force to use COBS, what makes it easier to write an own decoder in some cases or disable framing at all.
  - Change the setting `TRICE_FRAMING` inside `triceConfig.h` and use the Trice tool `-packageFraming` switch accordingly.
- For robustness each Trice can get its own (T)COBS package (`TRICE_DEFERRED_TRANSFER_MODE == TRICE_SINGLE_PACK_MODE`). That is configurable for transfer data reduction. Use `#define TRICE_DEFERRED_TRANSFER_MODE TRICE_MULTI_PACK_MODE` inside `triceConfig.h` (is now default). This allows to reduce the data size a bit by avoiding many 0-delimiter bytes but results in some more data loss in case of data disruptions.

(back to top)

16. Optional XTEA Encryption

- If XTEA is used, the encrypted packages have a multiple-of-8 byte length containing 1-7 padding bytes.
- The optional decryption is the next step after unpacking a data frame.
- Enabling XTEA, automatically switches to COBS framing. There is no need to use the Trice tool `-packageFraming` switch in that case because the Trice tool, when getting the CLI switch `-password "phrase"` automatically assumes COBS encoded data, overwriting the default value for `-packageFraming`.

(back to top)

17. Endianness

- To interpret a decoded package, it´s endianness needs to be known.
- For efficiency, binary trice data are normally stored and transmitted in MCU endianness and the Trice tool expects binary data in little endian format as most MCUs are little endian.
- On big endian MCUs the compiler switch `TRICE_MCU_IS_BIG_ENDIAN` needs to be defined as 1 and `TRICE_TRANSFER_ORDER_IS_BIG_ENDIAN` should have the same value. The Trice tool has a CLI switch "triceEndianness" which needs to be set to "bigEndian" then.
- If trice transmit data are needed to be not in MCU order for some reason, that increases the critical trice storage time and target code amount.
- De facto different values for `TRICE_MCU_IS_BIG_ENDIAN` and `TRICE_TRANSFER_ORDER_IS_BIG_ENDIAN` are mainly used to test the Trice CLI switch `-triceEndianness bigEndian` automatically.

(back to top)

18. Trice (Time)Stamps

- Each Trice message can carry stamp bits, which are free usable like for time, addressing or filtering.
- By selecting the letter case (**tr**ice, **Tr**ice, **TR**ice) you decide for each single Trice macro about the stamp size.
- Default notation (function call):

notation	stamp size	remark
`trice( iD(n), "...", ...);`	0-bit	no stamp at all, shortest footprint
`Trice( iD(n), "...", ...);`	16-bit	calls internally `uint16_t TriceStamp16( void )` for trice message stamping
`TRice( iD(n), "...", ...);`	32-bit	calls internally `uint32_t TriceStamp32( void )` for trice message stamping

- No upper case macro, like `TRICE_S` works with the internal `iD(n)` macro. They need `id(n)`, `Id(n)` or `ID(n)`. See next table.

- Legacy notation (code inlining):

notation	stamp size	remark
`TRICE( id(n), "...", ...)`	0-bit	no stamp at all, shortest footprint
`TRICE( Id(n), "...", ...)`	16-bit	calls internally `uint16_t TriceStamp16( void )` for trice message stamping
`TRICE( ID(n), "...", ...)`	32-bit	calls internally `uint32_t TriceStamp32( void )` for trice message stamping

It is up to the user to provide the functions `TriceStamp16` and/or `TriceStamp32`. Normally they return a µs or ms tick count but any values are allowed.

18.1. Target (Time)Stamps Formatting

To get a short overview run trice help -log and read about the CLI switches `ts`, `ts0`, `ts16`, `ts32`. The `ts32` switch supports also "epoch" now as format. That is useful for example, if the binary logs are stored internally in the device flash and read out later. Such usage assumes 1 second as ts32 unit in `uint32_t` format and the Trice tool displays the UTC time. It is also possible to adapt the displayed format like this for example: `trice log -ts32='epoch"06-01-02_15:04:05"'`. The additional passed string must match the Go time package capabilities. A few examples:

```
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Mon Jan _2 15:04:05 2006"'             # ANSIC
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Mon Jan _2 15:04:05 MST 2006"'         # UnixDate
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Mon Jan 02 15:04:05 -0700 2006"'       # RubyDate
trice log -port FILEBUFFER -args myLogs.bin -ts32='"02 Jan 06 15:04 MST")'                 # RFC822
trice log -port FILEBUFFER -args myLogs.bin -ts32='"02 Jan 06 15:04 -0700")'               # RFC822Z
(RFC822 with numeric zone)
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Monday, 02-Jan-06 15:04:05 MST")'      # RFC850
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Mon, 02 Jan 2006 15:04:05 MST")'       # RFC1123
trice log -port FILEBUFFER -args myLogs.bin -ts32='"Mon, 02 Jan 2006 15:04:05 -0700")'     # RFC1123Z
(RFC1123 with numeric zone)
trice log -port FILEBUFFER -args myLogs.bin -ts32='"2006-01-02T15:04:05Z07:00")'           # RFC3339
trice log -port FILEBUFFER -args myLogs.bin -ts32='"2006-01-02T15:04:05.999999999Z07:00")' # RFC3339Nano
trice log -port FILEBUFFER -args myLogs.bin -ts32='"3:04PM")'                              # Kitchen
```

After the year 2106 the Trice tool needs a small modification to correctly compute the epoch time then. Probably I will not be alive anymore to do that then, but, hey, Trice is Open Source!

(back to top)

19. Binary Encoding

19.1. Symbols

Symbol	Meaning
`i`	ID bit
`I`	`iiiiiiii` = ID byte
`n`	number bit
`z`	count selector bit
`s`	stamp selector bit
`N`	`znnnnnnnn` = count selector bit plus 7-bit number byte
`c`	cycle counter bit
`C`	z==0 ? `cccccccc` : `nnnnnnnn` = cycle counter byte or number byte extension
`t`	(time)stamp bit
`T`	`tttttttt` = (time)stamp byte
`d`	data bit

Symbol	Meaning
D	dddddddd = data byte
...	0 to 32767 data bytes
"..."	format string
W	bit width 8, 16, 32 or 64 (uW stands for u8, u16, or u64)
x	unspecified bit
X	=xxxxxxxx unspecified byte

19.2. Package Format

- Because of the **TCOBS** or **COBS** package framing, the package sizes are detectable by the trice tool without additional length information.
- All decoded frames of 0-, 1-, 2- and 3-byte size are considered as user data and ignored by the Trice tool.

bytes	Comment
	This is an empty package, which can have also a meaning. It is detectable by 2 consecutive 0-delimiter bytes.
X	1-byte message, reserved for extensions or user data
X X	2-byte message, reserved for extensions or user data
X X X	3-byte message, reserved for extensions or user data

- In decoded frames with >= 4-bytes the first 2 bytes contain 2 stamp selector bits at the most significant position in the known endianness.
- The 0 stamp selector is usable for any user encoding. The Trice tool handles such packages according to a CLI switch -typeX0.
- The 1, 2 and 3 stamp selector bits are followed by the 14-bit ID.

16-bit groups	Stamp Selector (2 msb)	Comment	Endianness sizes
_____ 00xxxxxx X ...	0	[typeX0 Trice](typeX0 Trices), >= 4-byte message, reserved for extensions or user data	__ u16 ?...?
_____ 01iiiiii I NC ...	1	>= 4-byte message, Trice format without stamp	__ u16 u16 [uW] ... [uW]
_____ 10iiiiii I TT NC ...	2	>= 4-byte message, Trice format with 16-bit stamp	__ u16 u16 u16 [uW] ... [uW]
10iiiiii I 10iiiiii I TT NC ...	2	First 16bit are doubled. Info over -d16 trice switch.	u16 u16 u16 u16 [uW] ... [uW]
_____ 11iiiiii I TT TT NC ...	3	>= 4-byte message, Trice format with 32-bit stamp	__ u16 u32 u16 [uW] ... [uW]

- The stamp selector 2 encoding has 2 possibilities. When using TRICE_DIRECT_SEGGER_RTT_32BIT_WRITE or encryption, for alignment reasons the first 16bit ID field is doubled. The trice tool discards these 2 doubled bytes when the CLI switch -d16 is given or encryption is active.
- Default endianness is little endian as most MCUs use little endianness. Otherwise the -triceEndianness=bigEndian CLI switch is needed.
- The receiving tool evaluates firstly the 2 stamp bits and follows some rules:
  - 0: treat it as user data when CLI switch -typeX0 is passed or error and ignore the whole package (discard).
  - 1: next 14 bits are the ID followed by 2 bytes u16=NC and optional parameter values. Package size is >= 4 bytes.

    - 2 and `-d16` CLI switch not provided: next 14 bits are the ID and convert then u16=TT=stamp16 followed by 2 bytes u16=NC and optional parameter values. Package size is >= 6 bytes.
    - 2 and `-d16` CLI switch provided: next 14 bits are the ID, discard 2 following bytes and convert then u16=TT=stamp16 followed by 2 bytes u16=NC and optional parameter values. Package size is >= 8 bytes.
    - 3: next 14 bits are the ID and convert then u32=TTTT=stamp32 followed by 2 bytes u16=NC and optional parameter values. Package size is >= 8 bytes.

- use ID to get parameters width `W`=8,16,32,64 from file *til.json* and and parameters count and convert appropriate.
    - Within one trice message the parameter bit width `W` does not change.

19.2.1. typeX0 Trices

The user can insert any data with a well-defined structure into the Trice data stream. The Trice tool, when interpreting the Trice binary data, will behave on typeX0 Trices according to a passed CLI switch `-typeX0`.

One possible use case is to have user **printi** statements parallel to Trices (see Legacy User Code Option Print Buffer Wrapping and Framing. The user needs to prepend a generated **printi** buffer with its size as 16-bit count (<16384!) for example. See ./_test/userprint_dblB_de_tcobs_ua/TargetActivity.c for an implementation option.

19.2.2. Framing - NONE or with COBS or TCOBS encoding

> Summary Information for Trice Data Parsing

- With (T)COBS framing:
    - 1-3 package delimiter zeroes possible between 2 packages.
    - One or more Trices packed together and only at the package end are 0-3(7) padding zero bytes possible.
    - typeX0 Trices do not occur together with normal Trices together in a package.
- With NONE framing:
    - With XTEA encryption and `-pf=none`or `-pf=none64` 64-bit alignment: 0-7 zero bytes after a single Trice.
    - Without encryption the stream is compact or 32-bit aligned. That does not change for one session and is detectable.
        - `-pf=none` -> detect stream (depreciated, only for backward compability)
        - `-pf=none8` -> stream is compact
        - `-pf=none32` -> stream is 32-bit aligned
    - A stream with alignment is allowed to have only a single Trice between two alignments. An alignment is just a multiple of 4(8)-bytes distance.
    - These combinations are forbidden, because we cannot safely know the actual padding count: `|TriceATriceB0|TriceC00|` <- Is the `0` after `TriceB` a padding zero or part of `TiceC`?
        - Framing NONE && TRICE_MULTI_PACK_MODE && 32-bit write
        - Framing NONE && TRICE_MULTI_PACK_MODE && XTEA encryption

> Details

- For maximum storage speed each Trice message starts at a 32-bit boundary and has 1-3 padding bytes inside the target device RAM.
- The macro `TRICE_LEAVE` and/or function `TriceTransfer` (Trice Target Code Implementation) are the Trice data output.
    - In **direct mode** each single message gets its own transfer buffer.
    - In **deferred mode** any count of Trice messages is in the transfer buffer.
    - Additional **typeX0 Trices** are always in a separate transfer buffer or, when without framing, follow immediate after 0-3 padding bytes.

Framing NONE Overview Table:

mode	***packed***	***-pf=***	**encr**	***wr***	***use***	**pad**	**stream**	**remark**
di	*single*	`none32`	NONE	*32*	*32*	0-3	aligned	done
de	*single*	`none8`	NONE	*8*	*8*	0	compact	done
de	*single*	`none32`	NONE	*8*	*32*	0	aligned	**plan**
de	*multi*	`none8`	NONE	*8*	*8*	0	compact	done

mode	*packed*	`-pf=`	encr	*wr*	*use*	pad	stream	remark
de	*multi*	`none32`	NONE	*8*	*32*	0	unknown	forbid
de	*multi*	`none`	NONE	*8*	*32*	0	unknown	forbid
di	*single*	`none64`	XTEA	*32*	*32*	0-7	aligned	done
de	*single*	`none64`	XTEA	*8*	*8*	0-7	aligned	done
de	*single*	`none64`	XTEA	*8*	*32*	0-7	aligned	**plan**
de	*multi*	`none`	XTEA	*8*	*8*	0-7	unknown	forbid
de	*multi*	`none`	XTEA	*8*	*32*	0-7	unknown	forbid

- wr: The internal write function bit width.
- use: The possible user write function bit width (auxiliary write)

(back to top)

20. Trice Decoding

The 14-bit IDs are used to display the log strings. These IDs are pointing in two reference files.

20.1. Trice ID list til.json

- This file integrates all firmware variants and versions and is the key to display the message strings. With the latest version of this file all previous deployed firmware images are usable without the need to know the actual firmware version.
- The files `til.h` and `til.c` are generated to help writing an own trice decoder tool in your preferred language. Use `trice generate -tilH -tilC` for creation. That can be interesting in environments, where Go compiled binaries not executable, like PCs running QNX OS. See also chapter Trice Generate.

20.2. Trice location information file li.json

- If the generated `li.json` is available, the Trice tool automatically displays file name and line number. But that is accurate only with the exact matching firmware version. That usually is the case right after compiling and of most interest at the developers table.
- The Trice tool will silently not display location information, if the `li.json` file is not found. For in-field logging, the option `-showID "inf:%5d"` could be used. This allows later an easy location of the relevant source code.
- An other option is to record the binary trice messages (`trice log -p com1 -blf aFileName`) and to play them later with the Trice tool using the correct `li.json` (`trice log -p FILEBUFFER -args aFileName`).
- The user has the possibility to change the default kind of the location information path. Normally it is "base" and just the source code file names, containing Trice macros occur inside the *li.json* file. That is sufficient as long there are no files with identical names on different places possibly containing identical Trices. Then it is not guarantied that the same IDs are always assigned. For such cases "relative" or "full" is selectable for the location information path kind (CLI -liPath). But both have weaknesses: The "full" path can differ between different machines and "relative" can differ between different projects on the same PC. Again: we are talking here about identical Trice maessges in files with identical names and the worst happening is that these "exchange" their IDs between `trice insert && trice clean && trice insert`. When using the Trice cache even less inconsistencies are expected.
    - The Trice folder itself is an example, how to deal with several projects and user library code. The "user libraries" here are inside `./_test/testdata` and `./examples/exampleData`. The "user projects" are the folders `./examples/*_inst/`. You can take the `*.sh` scripts as examples how to deal with such case.

(back to top)

21. Trice ID Numbers

21.1. ID number selection

- The default encoding TREX supports 14-bit IDs, so over 16000 IDs possible. Other encodings can work with other ID sizes.
- `trice("Hi!\n");` ➡ `trice i` ➡ `trice( iD(12345), "Hi!\n");` ➡ `trice c` ➡ `trice("Hi!\n");`
- The **ID** `12345` is a number assigned to `trice( "Hi!\n");` in the above example.
    - It is a so far unused number, according to rules you can control:
        - The `-IDMethod` switch allows a selection method for new IDs.
            - Per default new IDs determined randomly to keep the chance low, that several developers grab the same ID.
            - Example: `trice insert -IDMin 1000 -IDMethod upward` will choose the smallest free ID >= 1000.

        - This allows to use the ID space without wholes.
    - The `-IDMin` and `-IDMax` switches are usable to control the ID range, a new ID is selected from, making it possible to divide the ID space. Each developer can gets it region.
        - Example: `trice insert -IDMin 6000 -IDMax 6999` will choose new randomly IDs only between 6000 and 6999.
- It is possible to give each Trice tag an **ID** range making it possible to implement Trice tag specific runtime on/off on the target side if that is needed. This could be interesting for routing purposes also. Please run `trice help -insert` and read about the `-IDRange` switch for more details.

21.1.1. Trice tool internal Method to get fast a random ID

- Create Slice with numbers 1...16383
- Remove all used ID numbers from this slice
- Get random number between 0 and len(slice)
- remove this ID from slice and use it as new ID

21.2. ID number usage and stability

- If you write `trice( "msg:%d", 1);` again on a 2nd location, the copy gets a different **ID**, because each Trice gets its own **ID**.
- If you change `trice( "msg:%d", 1);` to `trice8( "msg:%d", 1);`, to reduce the needed parameter space, a new **ID** is assigned. That is because the parameter bit width is implicit a part of the now changed Trice. If you change that back, the previous **ID** is assigned again.
- If you change `trice( "msg:%d", 1);` to `TRice8( "msg:%d", 1);`, to get a 32-bit stamp, the associated **ID** remains unchanged. That is because the optional stamp is not a part of the Trice itself.
- IDs stay constant and get only changed to solve conflicts.
- To make sure, a single ID will not be changed, you could change it manually to a hexadecimal syntax.
    - This lets the `trice insert` command ignore such Trice macros and therefore a full til.json rebuild will not add them anymore. Generally this should not be done, because this could cause future bugs.
    - It is possible to assign an ID manually as decimal number. It will be added to the ID list automatically during the next `trice i|c` if no conflicts occur.
- If a Trice was deleted inside the source tree (or file removal) the appropriate ID stays inside the ID list.
- If the same string appears again in the same file this ID is active again.
- If a trice occurs more than one time, each occurrence gets a different ID. If then 2 of them disappear, their ID numbers stay in `til.json`. If then one of them comes back, it gets its ID back.

21.3. Trice ID 0

- The trice ID 0 is a placeholder for "no ID", which is replaced automatically during the next `trice insert` according to the used trice switches `-IDMethod`, `-IDMin` and `IDMax`.
    - It is sufficient to write the `TRICE` macros just without the `id(0), Id(0), ID(0),`. It will be inserted automatically according the `-defaultStampSize` switch. With `trice clean` these stay with 0-values in the source code to encode the intended stamp size.
    - It is recommended to use the `trice`, `Trice` and `TRice` macros instead of `TRICE`. They encode the stamp size in their names already. There may be cases, where the user prefers to use the code inserting macros `TRICE` to get maximum performnce.

(back to top)

22. Trice ID management

22.1. Trice inside source code

22.1.1. Trice in source code comments

- Trice macros commented out, are **visible** for the `trice insert` command and therefore regarded.
    - Example: `// trice("Hi!\n");` is still regarded by the `trice i`.
- During `trice insert` commented out Trice macros, are treated in the same way as active Trice macros. Even after deletion their content stays inside til.json. This is intensionally to get best stability across several firmware versions or variants.
- The trice tool does treat trice statements inside comments or excluded by compiler switches also.

22.1.2. Different IDs for same Trices

- When the same Trice is used several times with identical IDs, after copying, and `trice insert` is called, only one ID survives in the source code. The other Trices get assigned new IDs. Otherwise the location information would not be correct everywhere.

22.1.3. Same IDs for different Trices

- If duplicate ID's with different format strings found inside the source tree (case several developers or source code merging) one ID is replaced by a new ID. The probability for such case is low, because of the default random ID generation.
- Also you can simply copy a Trice statement and modify it without dealing with the ID.
- The Trice tool will detect the 2nd (or 3rd) usage of this ID and assign a new one, also extending the ID list.
- That is done silently for you during the next `trice insert`.
- When you use the Trice Cache, the IDs are invisible and all happens in the background automatically.

22.1.4. ID Routing

With the Trice insert CLI switch `-IDRange` each Trice tag can get a specific ID range assigned and inside the project specific *triceConfig.h* the user can control, which ID range is routed to specific output channels. Search for `_MIN_ID` inside **triceDefaultConfig.h** and extend your search than for example `TRICE_UARTA_MIN_ID` to explore how to use.

22.1.5. Possibility to create new tags without modifying trice tool source

According to the demand in 541 a CLI switch `-ulabel` exists now.

Use `-ulabel` for additional user labels. Try this example in an empty folder:

- File *main.c*:

```
#include "trice.h"

int main(void){
    trice("msg:hi\n");
    trice("man:hi\n");
    trice("wife:hi\n");
    trice("any:hi\n");
}
```

- Bash:

```
touch til.json li.json
trice i -IDMin 1004 -IDMax 6999 -IDRange wife:16000,16009 -IDRange man:1000,1003 -ulabel man -ulabel wife
```

- File *main.c*:

```
#include "trice.h"

int main(void){
    trice(iD(5778), "msg:hi\n");
    trice(iD(1002), "man:hi\n");
    trice(iD(16004), "wife:hi\n");
    trice(iD(2184), "any:hi\n");
}
```

(back to top)

23. Trice version 1.0 Log-level Control

23.1. Trice version 1.0 Compile-time Log-level Control

In Trice version 1.0 is no compile-time log-level control. You can only disable **all** Trice logs

- on file level by adding a `#define TRICE_OFF` line before `#include "trice.h"`
- or project level by using `-DTRICE_OFF` as compiler switch.

23.2. Trice version 1.0 Run-time Log-level Control

Because the target Trice code is so fast and generates only a few bytes per log, in Trice version 1.0 is no direct run-time log-level control inside the target code. The user has the Trice CLI switches `-ban`, `-pick` and `-logLevel`, to control, which Trice messages are displayed by the Trice tool.

23.3. Trice Version 1.0 Compile-time - Run-time Log-level Control

During compilation the developer can control which Trice tags, like *info* in `trice( "info:...\n");` get get which ID range. Look for `-IDRange` in `trice h -i` output. By defining values like `TRICE_UARTA_MIN_ID` in the project specific *triceConfig.h* during compile-time is controllable, which Trice tags get routed to an output device or not.

(back to top)

24. ID reference list til.json

- The `trice insert` command demands a **til.json** file - it will not work without it. That is a safety feature to avoid unwanted file generations. If you are sure to create a new **til.json** file, create an empty one: `touch til.json`.
- The name **til.json** is a default one. With the command line parameter `-i` you can use any filename.
- It is possible to use several **til.json** files - for example one for each target project but it is easier to maintain only one **til.json** file for all projects.
- The ID reference list keeps all obsolete IDs with their format strings allowing compatibility to former firmware versions.
- One can delete the ID reference list when IDs inside the code. It will be reconstructed automatically from the source tree with the next `trice clean` command, but history is lost then.
- Keeping obsolete IDs makes it more comfortable during development to deal with different firmware variants at the same time.

24.1. til.json Version control

- The ID list should go into the version control repository of your project.
- To keep it clean from the daily development garbage one could `git restore til.json`, and re-build just before check-in.

```
--> Deleting til.json should not not be done when the sources are without IDs.
--> That would result in a loss of the complete ID history and a assignment of a complete new set of IDs.
```

You could write a small bash script similar to this (untested):

```
trice insert -cache # Insert the IDs into the source code.
git restore til.json # Forget the todays garbage.

# Add the todays IDs to the restored til.json and clean the code.
# We have to deactivate the cache to force the file processing to get the new IDs into til.json.
trice clean # Remove the IDs from the source code with deactivated cache.
```

24.2. Long Time Availability

- You could place a download link for the Trice tool and the used **til.json** list.
- Optionally add the (compressed/encrypted) ID reference list as resource into the target FLASH memory to be sure not to loose it in the next 200 years.

(back to top)

25. The Trice Insert Algorithm

25.1. Starting Conditions

```
@@ To understand this chapter you should look into the Trice tool source code. @@
```

- Before `trice i` is executed on a source tree, the starting conditions are partially undefined:

    - A trice ID list file `til.json` file must exist, but it is allowed to be empty.
        - The `til.json` is a serialized key-value map, where
            - the keys are the IDs i and
            - the values are Trice format string structs (bit width plus format string) named f.
            - When de-serializing, it is not impossible, that an ID is used more than one times. This can only happen, when **til.json** was edited manually, what normally is not done. But could be the result of a `git merge`.
                - The trice tool will report that as error and stop. The user then has to correct the error manually, for example by deleting one of the doubled keys.
            - This ID look-up is the key-value map `idToFmt TriceIDLookUp` as `map[TriceID]TriceFmt`.
                - Each ID i as key, points to one and only one f.
                - The TriceFmt structs contains the parameter width and the format string.
            - The idToFmt is reverted then into `fmtToId triceFmtLookUp` as map[TriceFmt]TriceIDs.
                - `TriceIDs` is a triceID slice because the identical f can have several ids (no shared IDs).
                - The format struct f look-up map fmtToId is used internally for faster access and always in sync with idToFmt.
        - idToFmt and fmtToId together are named lu.
    - A location information file `li.json` may exist or not.
        - The `li.json` is a serialized key-value map `idToLocRef TriceIDLookUpLI`, a `map[TriceID]TriceLI`, where
            - the keys are the IDs i and
            - the values are the location information (filename, line and position in line) structs.
        - Each ID as key points to one and only one location information.
- The `til.json` IDs may occur in the source tree not at all, once or several times. Also it is not guarantied, that the source tree Trices match the `til.json` value.
    - That is possible after code edit, for example or code copied or modified.
    - One and only one position is used and relevant, all others are ignored. If no `til.json` exists on the expected location the user must provide one, at least an empty file.
- The `li.json` IDs may occur in the source tree not at all, once or several times. Also it is not guarantied, that the source tree Trices match the `li.json` value.
    - One and only one position is used and relevant, all others are ignored. If no `li.json` exists on the expected location trice insert creates one there.
- The src tree can contain IDs not present inside `til.json`. This state is seldom, for example after adding sources containing IDs.

25.2. Aims

- The `trice insert` main aim is to have a consistent state between `til.json`, `li.json` and the source tree with no **ID** used twice.
- Also the changes should be minimal.
- As a general rule lu is only extendable.
- li is rebuild from scratch.
- For faster operation files will be processed parallel.
- To keep the Trice ID management simple, the `insert` operation acts "per file". That means, that in case a file is renamed or code containing trice statements is copied to an other file, new IDs are generated for the affectes trices.
    - File name changes occur are not that often, so that should be acceptable.

25.3. Method

25.3.1. Trice Insert Initialization

```
// insertIDsData holds the insert run specific data.
type insertIDsData struct {
    idToFmt    TriceIDLookUp     // idToFmt is a trice ID lookup map and is generated from existing
til.json file at the begin of SubCmdIdInsert. This map is only extended during SubCmdIdInsert and goes
back into til.json afterwards.
    fmtToId    triceFmtLookUp    // fmtToId is a trice fmt lookup map (reversed idToFmt for faster
operation) and kept in sync with idToFmt. Each fmt can have several trice IDs (slice).
    idToLocRef TriceIDLookUpLI   // idToLocInf is the trice ID location information as reference generated
from li.json (if exists) at the begin of SubCmdIdInsert and is not modified at all. At the end of
SubCmdIdInsert a new li.json is generated from itemToId.
    itemToId   TriceItemLookUpID // itemToId is a trice item lookup ID map, extended from source tree
during SubCmdIdInsert after each found and maybe modified trice item.
    idToItem   TriceIDLookupItem // idToItem is a trice ID lookup item map (reversed itemToId for faster
operation) and kept in sync with itemToId.
}
```

- Create an `insertIDsData` instance.
- De-serialize `til.json` into `idToFmt` and `fmtToId`. On error abort and report for manual correction. One result is a slice with used IDs.
- De-serialize `li.json` into `idToLocRef`. On error abort and report for manual correction. As result the slice with used IDs is extended.
  - If `li.json` contains IDs not already inside `til.json`, these are reported as warning.
  - `idToLocRef` stays untouched and is used only in cases when identical f are found.
- Create a slice `IDSpace` with numbers IDMin ... IDMax (1 ... 16383, or 1000 ... 1999 if specified in the command line that way)
- Remove all used IDs from `IDSpace`.
  - If used IDs outside IDMin and IDmax, for example IDMin=1000, IDmax=1999 and some used IDs are bigger or smaller these are not removable from IDroom what is ok.
- Create empty `itemToId` and `idToItem`.
- Walk the src and create a **s**ource **t**ree **m**ap STM with
  - key=`Trice+LI` and
  - value=**ID**.
- During STM creation use these rules:
  - If the next found f src ID == n != 0:
    - If ID n already inside STM set ID = 0 (that is brutal but ok)
    - Otherwise extend STM with ID n and remove n from ID space
      - It is possible, f is used n times with different IDs, so that is no problem.
      - It is possible, f is used n times with the same ID, so the first occurrence is the winner.
  - If the next found f src ID == 0 (normal case after trice z):
    - Look in flu
      - If not there, create new id and extend STM.
        - The new ID is "new", so forbidden to be inside ilu.
        - If it is accidentally somewhere in the so far unparsed src, we do not know that and therefore do not care about.
          - That is a seldom case and not worth to parse the source tree twice all the time.
        - Patch id into source and extend STM.
      - If the ID slice has len 1 (usually the case), take that n, extend STM and remove f from flu.
        - That is important because f could be copied before.
      - If the ID slice has a len > 1 (several IDs on the same string) check li
        - If li is empty, just remove the first id from the slice and extend STM
        - Loop over slice IDs
          - If a file matches, take the first occurrence, extend STM and remove id from the ID slice
          - If no file matches do the same as when li is empty.
          - That means, after file renaming or code copying between files during trice z state, new IDs are generated for that parts.
            - That is only for same f with several IDs cases
          - File changes during trice i state are ok, because STM is generated with the IDs inside the sources.

Until here the algorithm seem to be ok.

- STM is not needed but maybe helpful during debugging.
- STM than is usable to regenerate li.json and to extend til.json
- If after `trice i` a `trice c` and a `trice i` again is executed, all IDs are expected to be at the same place again. If in between `trice i`, an optional `trice c`and a `trice i` src was edited, most IDs are expected to be at the same place again.

25.4. User Code Patching (trice insert)

- A Trice **ID** is inserted by `trice insert` as shown in the table:

Unpatched User Code	**After `trice insert`**	**Remark**
`trice( "Hi!\n");`	`trice( iD(12345), "Hi!\n");`	no stamps
`Trice( "Hi!\n");`	`Trice( iD(12345), "Hi!\n");`	16-bit stamps
`TRice( "Hi!\n");`	`TRice( iD(12345), "Hi!\n");`	32-bit stamps

- Legacy code is handled this way:

Unpatched User Code	After `trice insert`	Remark
`TRICE( "Hi!\n");`	`TRICE( id(12345), "Hi!\n");`	no stamps after `trice i -defaultStampSize 0`
`TRICE( "Hi!\n");`	`TRICE( Id(12345), "Hi!\n");`	16-bit stamps after `trice i -defaultStampSize 16`
`TRICE( "Hi!\n");`	`TRICE( ID(12345), "Hi!\n");`	32-bit stamps after `trice i -defaultStampSize 32`
`TRICE( id(0), "Hi!\n");`	`TRICE( id(12345), "Hi!\n");`	no stamps
`TRICE( Id(0), "Hi!\n");`	`TRICE( Id(12345), "Hi!\n");`	16-bit stamps
`TRICE( ID(0), "Hi!\n");`	`TRICE( ID(12345), "Hi!\n");`	32-bit stamps

- A pre-build step `trice insert` generates the `Id(12345)` part. Examples:
  - `trice i` in your project root expects a til.json file there and checks sources and **til.json** for changes to insert.
  - `trice i -v -i ../../../til.json -src ../src -src ../lib/src -src ./` is a typical case as automated pre-build step in your project settings telling Trice to scan the project dir and two external directories. Even `trice i` is fast, it is generally quicker to search only relevant places.

25.5. User Code Patching Examples

- A Trice **ID** is modified as shown in these cases:
  - Previously inserted (patched) user code copied to a different location:

```
trice(iD(12345), "Hi!\n"); // copied
trice(iD(12345), "Hi!\n"); // original
trice(iD(12345), "Hi!\n"); // copied
```

  - After updating (patching) again:

```
trice(iD(12345), "Hi!\n");
trice(iD( 1233), "Hi!\n"); // re-patched
trice(iD( 1234), "Hi!\n"); // re-patched
```

    - If the code is copied inside the same file, the first occurrence after the copy stays unchanged and the following are modified.
    - If the code is copied to other files only, the copies get new IDs.
  - Previously inserted (patched) user code copied and modified:

```
trice(iD(12345), "Ha!\n"); // copied and modified
trice(iD(12345), "Hi!\n"); // original
trice(iD(12345), "Ha!\n"); // copied and modified
```

  - After updating (patching) again:

```
trice(iD( 2333), "Ha!\n"); // re-patched
trice(iD(12345), "Hi!\n"); // unchanged
trice(iD( 1234), "Ha!\n"); // re-patched
```

  - If the code is copied to other files, it is re-patched.
- A Trice **ID** is stays the same if the stamp size is changed. Example:

```
trice( iD(12345), "Hi!" ); // original
```

```
TRice( iD(12345), "Hi!" ); // manually changed stamp size and then "trice i" performed.
```

25.6. Exclude folders & files from being parsed (pull request 529)

The pull request #529 introduces key enhancement:

```
    -exclude Flag
    Introduces a command-line flag -exclude that allows users to specify one or more source addresses to
be omitted from scanning or processing. This improves flexibility in environments with known noisy or
irrelevant sources.
```

-exclude Flag Example

The -exclude flag can be used multiple times to omit specific files or directories from scanning. Wildcards are not supported.

trice insert -v -src ./_test/ -exclude _test/src/trice.h -exclude _test/generated/

25.7. ID Usage Options

- Per default the `trice insert` command chooses randomly a so far unused ID for new format strings and extends `til.json`.
- After `trice c` all src IDs are removed or 0. In this state the src should go into the version management system.

25.8. General ID Management Information

- Each format string gets its unique trice ID. If the same format string is used on different source code locations it gets different trice IDs this way allowing a reliable location information.
- The trice ID-instead-of-String idea lives from pre-compile patching of the user code.
- The user has full control how to deal with that.
- There are the 3 following options and the user has to decide which fits best for him. The Trice Cache is probably the best fitting setup for many users.

25.8.1. Option Cleaning in a Post-build process

- The code is visually free of IDs all the time.

25.8.2. Option Let the inserted Trice ID be a Part of the User Code

- This is the legacy method. It allows unchanged src translation into code without using the trice tool.
- It is very robust and maybe needed in nasty debugging situations.
- It allows to reconstruct lost til.json information.
- Recommendet for small projects.

25.8.3. Option Cleaning on Repository Check-In

- The code is visually free of IDs only inside the repository.

(back to top)

26. Trice Speed

A Trice macro execution can be as cheap like **3 Assembler instructions or 6 processor clocks**:

- Disassembly:

```
    83:                                     TRICE( Id(59696), "msg:MEASURE execution\n" );
0x080008BE 61CD      STR      r5,[r1,#0x1C]
0x080008C0 620C      STR      r4,[r1,#0x20]
0x080008C2 6985      LDR      r5,[r0,#0x18]
    84:                                     TRICE( Id(59696), "msg:MEASURE execution\n" );
0x080008C4 624D      STR      r5,[r1,#0x24]
0x080008C6 628C      STR      r4,[r1,#0x28]
0x080008C8 6985      LDR      r5,[r0,#0x18]
    85:                                     TRICE( Id(59696), "msg:MEASURE execution\n" );
0x080008CA 62CD      STR      r5,[r1,#0x2C]
0x080008CC 630C      STR      r4,[r1,#0x30]
0x080008CE 6985      LDR      r5,[r0,#0x18]
```

- Measurement: The blue SYSTICK clock counts backwards 6 clocks for each Trice macro (on an ARM M0+), what is less than 100 ns @64 MHz MCU clock:

```
Nov 29 22:28:09.635560  COM7:     15815 MEASURE execution
Nov 29 22:28:09.635560  COM7:     15809 MEASURE execution
Nov 29 22:28:09.635560  COM7:     15803 MEASURE execution
```

A more realistic (typical) timing with target location and µs timestamps, critical section and parameters is shown here with the STM32F030 M0 core:

```
Jan  2 15:10:52.495068  COM6: triceCheck.c: 746  50000030 TRICE64 double -518.054749 (%F), aDouble(y)
Jan  2 15:10:52.495068  COM6: triceCheck.c: 747  50000032 TRICE64 double -518.0547492508867 (%G), aDouble(y)
Jan  2 15:10:52.495068  COM6: triceCheck.c: 749  50000035 formatted float & double
Jan  2 15:10:52.495068  COM6: triceCheck.c: 750  50000037 TRICE32 float 1.089608e+03 (%12.6e)
Jan  2 15:10:52.495068  COM6: triceCheck.c: 751  50000039 TRICE32 float  1089.608276 (%12.6f)
Jan  2 15:10:52.495571  COM6: triceCheck.c: 752  50000041 TRICE32 float      1089.61 (%12.6g)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 753  50000043 TRICE32 float 1.089608E+03 (%12.6E)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 754  50000045 TRICE32 float  1089.608276 (%12.6F)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 755  50000046 TRICE32 float      1089.61 (%12.6G)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 757  50000049 TRICE64 double 5.180547e+02 (%12.6e), aDouble(y)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 758  50000051 TRICE64 double   518.054749 (%12.6f), aDouble(y)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 759  50000053 TRICE64 double      518.055 (%12.6g), aDouble(y)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 760  50000055 TRICE64 double 5.180547E+02 (%12.6E), aDouble(y)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 761  50000057 TRICE64 double   518.054749 (%12.6F), aDouble(y)
Jan  2 15:10:52.495614  COM6: triceCheck.c: 762  50000060 TRICE64 double      518.055 (%12.6G), aDouble(y)
Jan  2 15:10:52.495614  COM6:       main.c: 158  50000062 MSG: STOP  select = 24, TriceDepthMax =1832
Jan  2 15:10:52.496122  COM6:       main.c: 165  50000065  50000064 µs - ReadUs() lasts 45 ticks
Jan  2 15:10:52.718370  COM6:       main.c: 156  50200005 MSG: START select = 0, TriceDepthMax =1832
Jan  2 15:10:52.718370  COM6: triceCheck.c:  34  50200008 Various single arguments
Jan  2 15:10:52.718370  COM6: triceCheck.c:  35  50200010 TRICE8 line false (%t ,0)
Jan  2 15:10:52.718872  COM6: triceCheck.c:  36  50200012 TRICE8 line true (%t ,2)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  37  50200014 TRICE8 line 255 (%u ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  38  50200016 TRICE8 line 11111111 (%b ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  39  50200018 TRICE8 line 377 (%o ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  40  50200020 TRICE8 line 0o377 (%O ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  41  50200022 TRICE8 line FF (%X ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  42  50200024 TRICE8 line ff (%x ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  43  50200026 TRICE8 line -1 (%d ,-1)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  44  50200028 TRICE16_1 line false (%t ,0)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  45  50200030 TRICE16_1 line true (%t ,2)
Jan  2 15:10:52.718917  COM6: triceCheck.c:  46  50200032 TRICE16_1 line 65535 (%u -1)
Jan  2 15:10:52.719420  COM6: triceCheck.c:  47  50200034 TRICE16_1 line 1111111111111111 (%b -1)
Jan  2 15:10:52.719459  COM6: triceCheck.c:  48  50200036 TRICE16_1 line 177777 (%o -1)
```

The MCU is clocked with 48 MHz and a Trice duration is about 2 µs, where alone the internal ReadUs() call is already nearly 1 µs long:

```
//! ReadUs32 reads the 1us tick in the assumption of an 48MHz systick clock using the microSecond variable and current systick value.
//! ATTENTION: This is a quick and dirty implementation working well only if this function is called in intervals smaller than 1 ms.
//! :-( Because the STM32F030 has no 32-bit sysclock counter we need to compute this value or concatenate two 16-bit timers. )
//! I see no way to find out if the systick ISR was already active shortly after a systick counter wrap, despite calling this
//! function in intervals smaller than 1 ms if not using hardware timers. To make it clear: You can use ReadUs32 to measure long
//! intervals up to over 1 hour (4294 seconds), but the "OS" needs to call ReadUs32  internally regularely in <1ms intervals.
//! \retval us count since last reset modulo 2^32
uint32_t ReadUs32( void ){
    static uint32_t us_1 = 0; // result of last call
    uint32_t us = ((uint32_t)microSecond) + (((SysTick->LOAD - SysTick->VAL) * 87381) >> 22); // Divide clock by 48,0001831 to get us.
    if( us < us_1){ // Possible very close to systick ISR, when milliSecond was not incremented yet, but the systic wrapped already.
        us += 1000; // Time cannot go backwards, so correct the 1ms error in the assumption last call is not longer than 1ms back.
    }
    us_1 = us; // keep result for next call
    return us;
}
```

26.1. Target Implementation Options

All trice macros use internally this sub-macro:

```
#define TRICE_PUT(x) do{ *TriceBufferWritePosition++ = TRICE_HTOTL(x); }while(0); //! PUT copies a 32 bit
x into the TRICE buffer.
```

The usual case is `#define TRICE_HTOTL(x) (x)`. The `uint32_t* TriceBufferWritePosition` points to a buffer, which is codified and used with the Trice framing sub-macros `TRICE_ENTER` and `TRICE_LEAVE` in dependence of the use case.

26.1.1. Trice Use Cases TRICE_STATIC_BUFFER and TRICE_STACK_BUFFER - direct mode only

1. Each singe Trice is build inside a common buffer and finally copied inside the sub-macro `TRICE_LEAVE`.
2. Disabled relevant interrupts between `TRICE_ENTER` and `TRICE_LEAVE` are mantadory for `TRICE_STATIC_BUFFER`.
3. Usable for multiple non-blocking physical Trice channels but **not** recommended for some time blocking channels.
4. A copy call is executed inside `TRICE_LEAVE`.

- With appropriate mapping a direct write to physical output(s) is possible:
  - RTT0 without extra copy.
    - With `TRICE_DIRECT_SEGGER_RTT_32BIT_WRITE` about 100 MCU clocks do the whole work, what is within 1.5 us @ 64 MHz.
  - AUX without extra copy.
  - Not (yet) supported UART transfer loop with polling. With 1MBit baud rate, 4-12 bytes would last 40-120 µs.

26.1.2. Trice Use Case TRICE_DOUBLE_BUFFER - deferred mode, fastest Trice execution, more RAM needed

1. Several *trices* are build in a half buffer.
2. No stack used.
3. Disabled interrupts between `TRICE_ENTER` and `TRICE_LEAVE`.
4. Usable for multiple blocking and non-blocking physical Trice channels.
5. No copy call inside `TRICE_LEAVE` but optionally an additional direct mode is supported.

26.1.3. Trice Use Case TRICE_RING_BUFFER - deferred mode, balanced Trice execution time and needed RAM

1. Each single *trices* is build in a ring buffer segment.
2. No stack used.
3. Disabled interrupts between `TRICE_ENTER` and `TRICE_LEAVE`.
4. Usable for multiple blocking and non-blocking physical Trice channels.
5. No copy call inside `TRICE_LEAVE` but optionally an additional direct mode is supported.
6. Allocation call inside `TRICE_ENTER`

26.2. A configuration for maximum Trice execution speed with the L432_inst example

- To not loose any clocks, the function `SomeExampleTrices` in triceExamples.c uses the upper case macro `TRICE` for the first "🦎 Speedy Gonzales" Trices.

- The `triceConfig.h` settings are

```
#define TriceStamp16 (*DWT_CYCCNT) // @64MHz wraps after a bit more than 1ms (MCU clocks)
#define TriceStamp32 (*DWT_CYCCNT) // @64MHz -> 1 µs, wraps after 2^32 µs ~= 1.2 hours

#define TRICE_DEFERRED_UARTA 1
#define TRICE_UARTA USART2

#define TRICE_DEFERRED_OUTPUT 1
#define TRICE_BUFFER TRICE_DOUBLE_BUFFER

#define TRICE_PROTECT 0
#define TRICE_DIAGNOSTICS 0
#define TRICE_CYCLE_COUNTER 0
```

- Both time stamps use the debug watchdog counter running with the 64 MHz MCU clock.
- No direct output to not loose time during the Trice macro execution.
- Critical sections are disabled (default), so be careful where Trices are used.

- The Trice double buffer allows the Trice macros to write without checks.
- The Trice protection, diagnostics and cycle counter are disabled to not perform unneeded clocks.
- Additionally in file flags.mak the optimization is set to`C_FLAGS += -Ofast`.

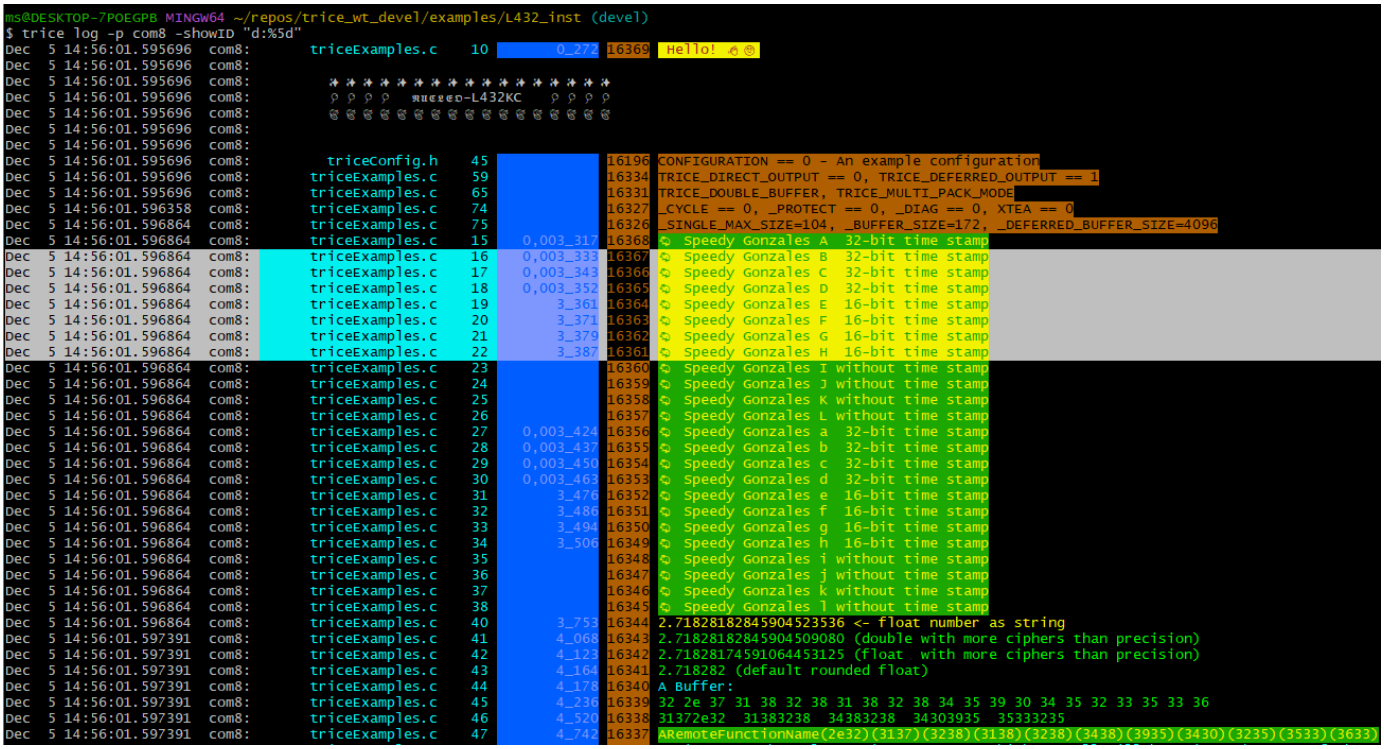
After running ./build.sh, executing `arm-none-eabi-objdump.exe -D -S -l out.clang/triceExamples.o` shows:

```
out.clang/triceExamples.o:     file format elf32-littlearm


Disassembly of section .text.TriceHeadLine:

00000000 <TriceHeadLine>:
TriceHeadLine():
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:10
#include "trice.h"

//! TriceHeadLine emits a decorated name. The name length should be 18 characters.
void TriceHeadLine(char * name) {
    //! This is usable as the very first trice sequence after restart. Adapt it. Use a UTF-8 capable
editor like VS-Code or use pure ASCII.
    TriceS("w: Hello! 👋😁\n\n        ✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨        \n        🍀🍀
🍀%s🍀🍀🍀🍀\n        🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀        \n\n\n", name);
   0:   f240 0100   movw    r1, #0
   4:   4602        mov r2, r0
   6:   f2c0 0100   movt    r1, #0
   a:   f643 70f1   movw    r0, #16369  @ 0x3ff1
   e:   f7ff bffe   b.w 0 <TriceS>

Disassembly of section .ARM.exidx.text.TriceHeadLine:

00000000 <.ARM.exidx.text.TriceHeadLine>:
   0:   00000000    andeq   r0, r0, r0
   4:   00000001    andeq   r0, r0, r1

Disassembly of section .text.SomeExampleTrices:

00000000 <SomeExampleTrices>:
SomeExampleTrices():
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:14
}

//! SomeExampleTrices generates a few Trice example logs and a burst of Trices.
void SomeExampleTrices(int burstCount) {
   0:   b5f0        push    {r4, r5, r6, r7, lr}
   2:   af03        add r7, sp, #12
   4:   e92d 0700   stmdb   sp!, {r8, r9, sl}
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:15
    TRICE(ID(0), "att:🐢 Speedy Gonzales A  32-bit time stamp\n");
   8:   f240 0100   movw    r1, #0
   c:   f2c0 0100   movt    r1, #0
  10:   680d        ldr r5, [r1, #0]
  12:   f240 0800   movw    r8, #0
  16:   f2c0 0800   movt    r8, #0
  1a:   4604        mov r4, r0
  1c:   6829        ldr r1, [r5, #0]
  1e:   f8d8 0000   ldr.w   r0, [r8]
  22:   f06f 0212   mvn.w   r2, #18
  26:   8041        strh    r1, [r0, #2]
  28:   0c09        lsrs    r1, r1, #16
  2a:   1cd3        adds    r3, r2, #3
  2c:   8081        strh    r1, [r0, #4]
  2e:   21c0        movs    r1, #192    @ 0xc0
  30:   8003        strh    r3, [r0, #0]
  32:   80c1        strh    r1, [r0, #6]
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:16
    TRICE(ID(0), "att:🐢 Speedy Gonzales B  32-bit time stamp\n");
  34:   682b        ldr r3, [r5, #0]
```

```
  36:   1c96        adds    r6, r2, #2
  38:   8143        strh    r3, [r0, #10]
  3a:   0c1b        lsrs    r3, r3, #16
  3c:   8106        strh    r6, [r0, #8]
  3e:   8183        strh    r3, [r0, #12]
  40:   81c1        strh    r1, [r0, #14]
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:17
    TRICE(ID(0), "att:🐢 Speedy Gonzales C  32-bit time stamp\n");
  42:   682b        ldr r3, [r5, #0]
  44:   1c56        adds    r6, r2, #1
  46:   8243        strh    r3, [r0, #18]
  48:   0c1b        lsrs    r3, r3, #16
  4a:   8206        strh    r6, [r0, #16]
  4c:   8283        strh    r3, [r0, #20]
  4e:   82c1        strh    r1, [r0, #22]
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:18
    TRICE(ID(0), "att:🐢 Speedy Gonzales D  32-bit time stamp\n");
  50:   682b        ldr r3, [r5, #0]
  52:   8302        strh    r2, [r0, #24]
  54:   0c1a        lsrs    r2, r3, #16
  56:   8343        strh    r3, [r0, #26]
  58:   8382        strh    r2, [r0, #28]
  5a:   83c1        strh    r1, [r0, #30]
  5c:   f64b 7ad4   movw    sl, #49108  @ 0xbfd4
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:19
    TRICE(Id(0), "att:🐢 Speedy Gonzales E  16-bit time stamp\n");
  60:   682a        ldr r2, [r5, #0]
  62:   f6cb 7ad4   movt    sl, #49108  @ 0xbfd4
  66:   f10a 1318   add.w   r3, sl, #1572888    @ 0x180018
  6a:   6203        str r3, [r0, #32]
  6c:   8482        strh    r2, [r0, #36]   @ 0x24
  6e:   84c1        strh    r1, [r0, #38]   @ 0x26
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:20
    TRICE(Id(0), "att:🐢 Speedy Gonzales F  16-bit time stamp\n");
  70:   682a        ldr r2, [r5, #0]
  72:   f10a 1317   add.w   r3, sl, #1507351    @ 0x170017
  76:   6283        str r3, [r0, #40]   @ 0x28
  78:   8582        strh    r2, [r0, #44]   @ 0x2c
  7a:   85c1        strh    r1, [r0, #46]   @ 0x2e
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:21
    TRICE(Id(0), "att:🐢 Speedy Gonzales G  16-bit time stamp\n");
  7c:   682a        ldr r2, [r5, #0]
  7e:   f10a 1316   add.w   r3, sl, #1441814    @ 0x160016
  82:   6303        str r3, [r0, #48]   @ 0x30
  84:   8682        strh    r2, [r0, #52]   @ 0x34
  86:   86c1        strh    r1, [r0, #54]   @ 0x36
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:22
    TRICE(Id(0), "att:🐢 Speedy Gonzales H  16-bit time stamp\n");
  88:   682a        ldr r2, [r5, #0]
  8a:   f10a 1315   add.w   r3, sl, #1376277    @ 0x150015
  8e:   8782        strh    r2, [r0, #60]   @ 0x3c
  90:   f647 72e8   movw    r2, #32744  @ 0x7fe8
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:23
    TRICE(id(0), "att:🐢 Speedy Gonzales I without time stamp\n");
  94:   f8a0 2040   strh.w  r2, [r0, #64]   @ 0x40
  98:   f647 72e7   movw    r2, #32743  @ 0x7fe7
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:24
    TRICE(id(0), "att:🐢 Speedy Gonzales J without time stamp\n");
  9c:   f8a0 2044   strh.w  r2, [r0, #68]   @ 0x44
  a0:   f647 72e6   movw    r2, #32742  @ 0x7fe6
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:25
    TRICE(id(0), "att:🐢 Speedy Gonzales K without time stamp\n");
  a4:   f8a0 2048   strh.w  r2, [r0, #72]   @ 0x48
  a8:   f647 72e5   movw    r2, #32741  @ 0x7fe5
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:22
    TRICE(Id(0), "att:🐢 Speedy Gonzales H  16-bit time stamp\n");
  ac:   6383        str r3, [r0, #56]   @ 0x38
  ae:   87c1        strh    r1, [r0, #62]   @ 0x3e
```

```
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:23
    TRICE(id(0), "att:🐢 Speedy Gonzales I without time stamp\n");
  b0:   f8a0 1042   strh.w  r1, [r0, #66]   @ 0x42
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:24
    TRICE(id(0), "att:🐢 Speedy Gonzales J without time stamp\n");
  b4:   f8a0 1046   strh.w  r1, [r0, #70]   @ 0x46
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:25
    TRICE(id(0), "att:🐢 Speedy Gonzales K without time stamp\n");
  b8:   f8a0 104a   strh.w  r1, [r0, #74]   @ 0x4a
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../exampleData/triceExamples.c:26
    TRICE(id(0), "att:🐢 Speedy Gonzales L without time stamp\n");
  bc:   f8a0 204c   strh.w  r2, [r0, #76]   @ 0x4c
  c0:   f8a0 104e   strh.w  r1, [r0, #78]   @ 0x4e
TRice0():
C:\Users\ms\repos\trice_wt_devel\examples\L432_inst/../../src/trice.h:741
    Trice32m_0(tid);
    TRICE_UNUSED(pFmt)
}

...
```

- There are only 7 assembler instructions between two `TRICE` macros(around line 17).
- The log output is:



As you can see in the highlighted blue timestamp bar, typical 8-10 clocks are needed for one Trice macro. One clock duration @64MHz is 15.625 ns, so we need about 150 ns for a Trice. Light can travel about 50 meter in that time.

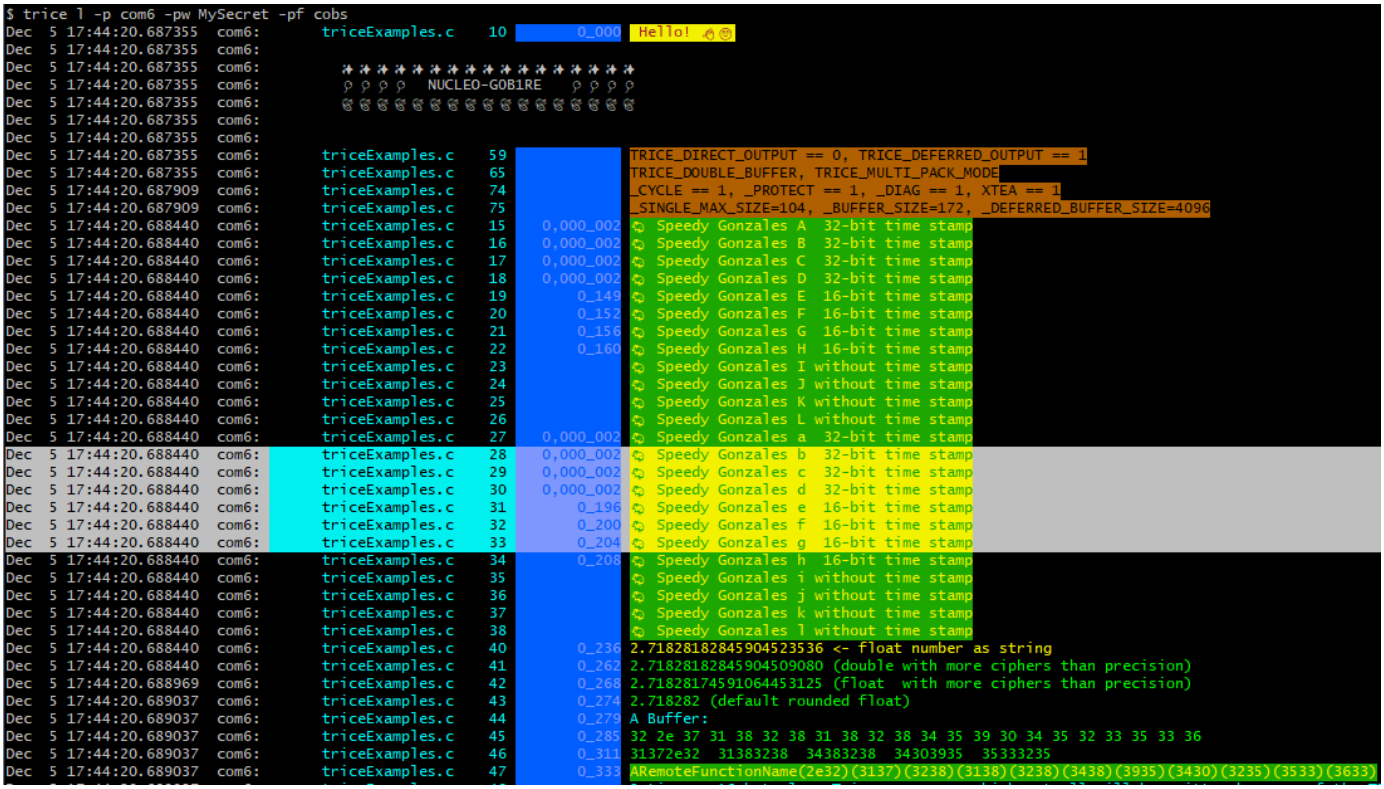
26.3. A configuration for normal Trice execution speed with the G0B1_inst example

- The `triceConfig.h` settings are

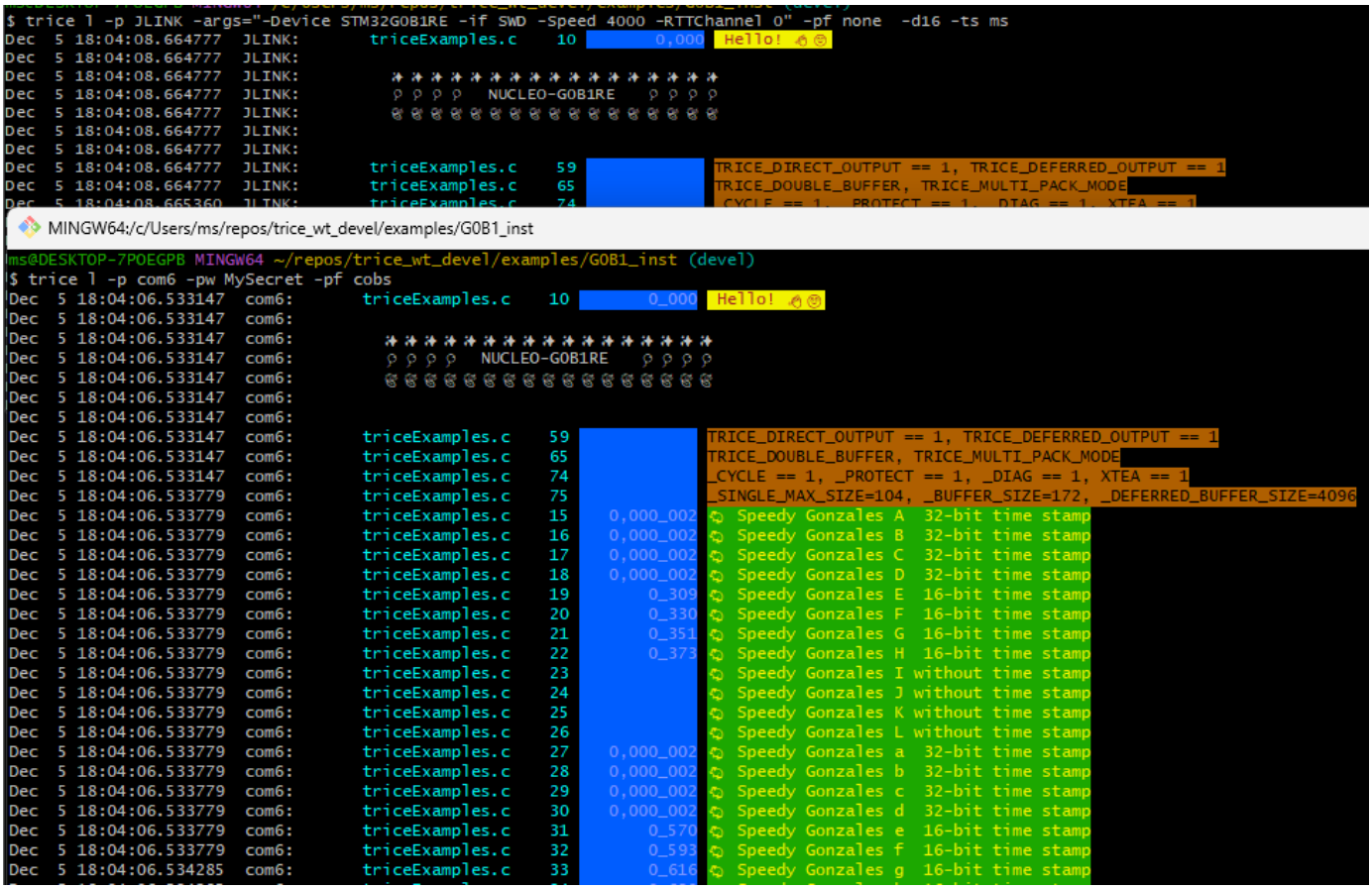
```
// hardware specific trice lib settings
#include "main.h"
#define TriceStamp16 TIM17->CNT     // 0...999 us
#define TriceStamp32 HAL_GetTick()  // 0...2^32-1 ms (wraps after 49.7 days)

#define TRICE_BUFFER TRICE_RING_BUFFER

// trice l -p JLINK -args="-Device STM32G0B1RE -if SWD -Speed 4000 -RTTChannel 0" -pf none  -d16 -ts ms
//#define TRICE_DIRECT_OUTPUT 1
```

```
//#define TRICE_DIRECT_SEGGER_RTT_32BIT_WRITE 1

// trice log -p com7 -pw MySecret -pf COBS
#define TRICE_DEFERRED_OUTPUT 1
#define TRICE_DEFERRED_XTEA_ENCRYPT 1
#define TRICE_DEFERRED_OUT_FRAMING TRICE_FRAMING_COBS
#define TRICE_DEFERRED_UARTA 1
#define TRICE_UARTA USART2

#include "cmsis_gcc.h"
#define TRICE_ENTER_CRITICAL_SECTION { uint32_t primaskstate = __get_PRIMASK(); __disable_irq(); {
#define TRICE_LEAVE_CRITICAL_SECTION } __set_PRIMASK(primaskstate); }
```

- The 16-bit time stamp counts the microseconds within 1 millisecond.
- The 32-bit time stamp counts the milliseconds.
- The ring buffer uses the RAM more effectively for the price of a bit speed.
- The encryption and framing has no influence on the Trice execution speed becuse this is done in the background.
- The critical section protects Trices in different tasks from each-other interruption and allows Trices inside interrupts parallel to normal usage.
- Per default are Trice protection, diagnostics and cycle counter active.



- A typical Trice duration is here 4 microseconds (with `-Ospeed`)
- Switcing the optimization to `-Oz` can result in typical 4-5 µs Trice execution time.
- Additionally enabling the Trice direct out over Segger RTT has this impact:



> ⚫ The Trice execution time is now over 20 microseconds ❗

Still fast enough for many cases but you hopefully have a good knowledge now how to tune Trice best for your application.

(back to top)

27. Trice memory needs

Depending on your target configuration the needed space can differ:

27.1. F030_bare Size

- `./build.sh`:

```
arm-none-eabi-size build/F030_bare.elf
   text    data     bss     dec     hex filename
   2428      12    1564    4004     fa4 build/F030_bare.elf
```

That is the basic size of an empty generated project just containing some drivers.

27.2. F030_inst Size with TRICE_OFF=1

- `./build.sh TRICE_OFF=1` :

```
arm-none-eabi-size build/F030_inst.elf
   text    data     bss     dec     hex filename
   2428      12    1564    4004     fa4 build/F030_inst.elf
```

This is exactly the same result, proofing that `TRICE_OFF 1` is working correctly.

27.3. F030_inst with ring buffer

- `./build.sh`:

```
arm-none-eabi-size out/F030_inst.elf
   text    data     bss     dec     hex filename
   9416      28    2692   12136    2f68 out/F030_inst.elf
```

This is about 7 KB Flash and 1.2 KB RAM size for the Trice library and we see:

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice_wt_devel/examples/F030_inst (devel)
$ trice l -p com5 -ts16 "time:     #%6d" -hs off
com5:       triceExamples.c    12      # 65535  Hello! 👌😊
com5:
com5:         ✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨
com5:         ☘☘☘☘  𝔑𝔘𝔈𝔏𝔈𝔒-F030R8    ☘☘☘☘
com5:         🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿
com5:
com5:
com5:       triceExamples.c    61              TRICE_DIRECT_OUTPUT == 0, TRICE_DEFERRED_OUTPUT == 1
com5:       triceExamples.c    67              TRICE_DOUBLE_BUFFER, TRICE_MULTI_PACK_MODE
com5:       triceExamples.c    76              _CYCLE == 1, _PROTECT == 1, _DIAG == 1, XTEA == 0
com5:       triceExamples.c    77              _SINGLE_MAX_SIZE=104, _BUFFER_SIZE=172,
_DEFERRED_BUFFER_SIZE=1024
com5:       triceExamples.c    29    0,031_804 🐇 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,031_646 🐇 Speedy Gonzales b  32-bit time stamp
com5:       triceExamples.c    31    0,031_488 🐇 Speedy Gonzales c  32-bit time stamp
com5:       triceExamples.c    32    0,031_330 🐇 Speedy Gonzales d  32-bit time stamp
com5:       triceExamples.c    33      # 31172 🐇 Speedy Gonzales e  16-bit time stamp
com5:       triceExamples.c    34      # 31012 🐇 Speedy Gonzales f  16-bit time stamp
com5:       triceExamples.c    35      # 30852 🐇 Speedy Gonzales g  16-bit time stamp
com5:       triceExamples.c    36      # 30692 🐇 Speedy Gonzales h  16-bit time stamp
com5:       triceExamples.c    42      # 30224 2.71828182845904523536 <- float number as string
com5:       triceExamples.c    43      # 29517 2.71828182845904509080 (double with more ciphers than
precision)
com5:       triceExamples.c    44      # 29322 2.71828174591064453125 (float  with more ciphers than
precision)
com5:       triceExamples.c    45      # 29145 2.718282 (default rounded float)
com5:       triceExamples.c    46      # 28969 A Buffer:
com5:       triceExamples.c    47      # 28790 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35 32 33 35
33 36
com5:       triceExamples.c    48      # 28076 31372e32  31383238  34383238  34303935  35333235
com5:       triceExamples.c    49      # 27430 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)(3438)
(3935)(3430)(3235)(3533)(3633)
com5:       triceExamples.c    50              5 times a 16 byte long Trice messages, which may not be
written all if the buffer is too small:
com5:       triceExamples.c    52      # 26554 i=44444400 aaaaaa00
com5:       triceExamples.c    52      # 26342 i=44444401 aaaaaa01
com5:       triceExamples.c    52      # 26130 i=44444402 aaaaaa02
com5:       triceExamples.c    52      # 25918 i=44444403 aaaaaa03
com5:       triceExamples.c    52      # 25706 i=44444404 aaaaaa04
com5:       triceExamples.c    29    0,031_790 🐇 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,031_632 🐇 Speedy Gonzales b  32-bit time stamp
com5:       triceExamples.c    31    0,031_474 🐇 Speedy Gonzales c  32-bit time stamp
com5:       triceExamples.c    32    0,031_316 🐇 Speedy Gonzales d  32-bit time stamp
com5:       triceExamples.c    33      # 31158 🐇 Speedy Gonzales e  16-bit time stamp
com5:       triceExamples.c    34      # 30998 🐇 Speedy Gonzales f  16-bit time stamp
com5:       triceExamples.c    35      # 30838 🐇 Speedy Gonzales g  16-bit time stamp
com5:       triceExamples.c    36      # 30678 🐇 Speedy Gonzales h  16-bit time stamp
com5:       triceExamples.c    42      # 30210 2.71828182845904523536 <- float number as string
com5:       triceExamples.c    43      # 29503 2.71828182845904509080 (double with more ciphers than
precision)
com5:       triceExamples.c    44      # 29308 2.71828174591064453125 (float  with more ciphers than
precision)
com5:       triceExamples.c    45      # 29131 2.718282 (default rounded float)
com5:       triceExamples.c    46      # 28955 A Buffer:
com5:       triceExamples.c    47      # 28776 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35 32 33 35
33 36
```

```
com5:       triceExamples.c    48      # 28062 31372e32  31383238  34383238  34303935  35333235
com5:       triceExamples.c    49      # 27416 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)(3438)
(3935)(3430)(3235)(3533)(3633)
com5:       triceExamples.c    50              5 times a 16 byte long Trice messages, which may not be
written all if the buffer is too small:
com5:       triceExamples.c    52      # 26540 i=44444400 aaaaaa00
com5:       triceExamples.c    52      # 26328 i=44444401 aaaaaa01
com5:       triceExamples.c    52      # 26116 i=44444402 aaaaaa02
com5:       triceExamples.c    52      # 25904 i=44444403 aaaaaa03
com5:       triceExamples.c    52      # 25692 i=44444404 aaaaaa04
com5:    triceLogDiagData.c    44              triceSingleDepthMax = 108 of 172 (TRICE_BUFFER_SIZE)
com5:    triceLogDiagData.c    67              TriceHalfBufferDepthMax = 388 of  512
com5:       triceExamples.c    29    0,031_344 🐑 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,031_186 🐑 Speedy Gonzales b  32-bit time stamp
```

27.4. F030_inst with ring buffer

- `./build.sh`:

We need 600 bytes more Flash but could have less RAM used:

```
   text    data     bss     dec     hex filename
  10060      24    2688   12772    31e4 out/F030_inst.elf
```

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice_wt_devel/examples/F030_inst (devel)
$ trice l -p com5 -ts16 "time:     #%6d" -hs off
com5:       triceExamples.c    12      # 65535  Hello! 👌😊
com5:
com5:         ✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨
com5:         ♌♌♌♌  𝔑𝔘𝔈𝔏𝔈𝔒-F030R8   ♌♌♌♌
com5:         🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃
com5:
com5:
com5:       triceExamples.c    61              TRICE_DIRECT_OUTPUT == 0, TRICE_DEFERRED_OUTPUT == 1
com5:       triceExamples.c    69              TRICE_RING_BUFFER, TRICE_MULTI_PACK_MODE
com5:       triceExamples.c    76              _CYCLE == 1, _PROTECT == 1, _DIAG == 1, XTEA == 0
com5:       triceExamples.c    77              _SINGLE_MAX_SIZE=104, _BUFFER_SIZE=172,
_DEFERRED_BUFFER_SIZE=1024
com5:       triceExamples.c    29    0,031_732 🐑 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,031_531 🐑 Speedy Gonzales b  32-bit time stamp
com5:       triceExamples.c    31    0,031_330 🐑 Speedy Gonzales c  32-bit time stamp
com5:       triceExamples.c    32    0,031_129 🐑 Speedy Gonzales d  32-bit time stamp
com5:       triceExamples.c    33      # 30928 🐑 Speedy Gonzales e  16-bit time stamp
com5:       triceExamples.c    34      # 30725 🐑 Speedy Gonzales f  16-bit time stamp
com5:       triceExamples.c    35      # 30522 🐑 Speedy Gonzales g  16-bit time stamp
com5:       triceExamples.c    36      # 30319 🐑 Speedy Gonzales h  16-bit time stamp
com5:       triceExamples.c    42      # 29808 2.71828182845904523536 <- float number as string
com5:       triceExamples.c    43      # 29058 2.71828182845904509080 (double with more ciphers than
precision)
com5:       triceExamples.c    44      # 28821 2.71828174591064453125 (float  with more ciphers than
precision)
com5:       triceExamples.c    45      # 28602 2.718282 (default rounded float)
com5:       triceExamples.c    46      # 28383 A Buffer:
com5:       triceExamples.c    47      # 28162 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35 32 33 35
33 36
com5:       triceExamples.c    48      # 27406 31372e32  31383238  34383238  34303935  35333235
com5:       triceExamples.c    49      # 26718 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)(3438)
(3935)(3430)(3235)(3533)(3633)
com5:       triceExamples.c    50              5 times a 16 byte long Trice messages, which may not be
written all if the buffer is too small:
com5:       triceExamples.c    52      # 25757 i=44444400 aaaaaa00
com5:       triceExamples.c    52      # 25502 i=44444401 aaaaaa01
com5:       triceExamples.c    52      # 25247 i=44444402 aaaaaa02
```

```
com5:       triceExamples.c    52      # 24992 i=44444403 aaaaaa03
com5:       triceExamples.c    52      # 24737 i=44444404 aaaaaa04
com5:       triceExamples.c    29    0,031_746 🐔 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,031_545 🐔 Speedy Gonzales b  32-bit time stamp
com5:       triceExamples.c    31    0,031_344 🐔 Speedy Gonzales c  32-bit time stamp
com5:       triceExamples.c    32    0,031_143 🐔 Speedy Gonzales d  32-bit time stamp
com5:       triceExamples.c    33      # 30942 🐔 Speedy Gonzales e  16-bit time stamp
com5:       triceExamples.c    34      # 30739 🐔 Speedy Gonzales f  16-bit time stamp
com5:       triceExamples.c    35      # 30536 🐔 Speedy Gonzales g  16-bit time stamp
com5:       triceExamples.c    36      # 30333 🐔 Speedy Gonzales h  16-bit time stamp
com5:       triceExamples.c    42      # 29822 2.71828182845904523536 <- float number as string
com5:       triceExamples.c    43      # 29072 2.71828182845904509080 (double with more ciphers than
precision)
com5:       triceExamples.c    44      # 28835 2.71828174591064453125 (float  with more ciphers than
precision)
com5:       triceExamples.c    45      # 28616 2.718282 (default rounded float)
com5:       triceExamples.c    46      # 28397 A Buffer:
com5:       triceExamples.c    47      # 28176 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35 32 33 35
33 36
com5:       triceExamples.c    48      # 27420 31372e32  31383238  34383238  34303935  35333235
com5:       triceExamples.c    49      # 26732 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)(3438)
(3935)(3430)(3235)(3533)(3633)
com5:       triceExamples.c    50              5 times a 16 byte long Trice messages, which may not be
written all if the buffer is too small:
com5:       triceExamples.c    52      # 25771 i=44444400 aaaaaa00
com5:       triceExamples.c    52      # 25516 i=44444401 aaaaaa01
com5:       triceExamples.c    52      # 25261 i=44444402 aaaaaa02
com5:       triceExamples.c    52      # 25006 i=44444403 aaaaaa03
com5:       triceExamples.c    52      # 24751 i=44444404 aaaaaa04
com5:    triceLogDiagData.c    44              triceSingleDepthMax = 108 of 172 (TRICE_BUFFER_SIZE)
com5:    triceLogDiagData.c    75              triceRingBufferDepthMax = 324 of 1024
com5:       triceExamples.c    29    0,031_188 🐔 Speedy Gonzales a  32-bit time stamp
com5:       triceExamples.c    30    0,030_987 🐔 Speedy Gonzales b  32-bit time stamp
```

27.5. A developer setting, only enabling SEGGER_RTT

- `./build.sh`:

```
arm-none-eabi-size out/F030_inst.elf
   text    data     bss     dec     hex filename
   6656      16    2768    9440    24e0 out/F030_inst.elf
```

About 4 KB Flash needed and we see:

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice_wt_devel/examples/F030_inst (devel)
$ trice l -p jlink -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0" -pf none -d16 -showID
"deb:%5d"
Dec  6 16:14:38.276356  jlink:       triceExamples.c    12       65_535 16369  Hello! 👌😀
Dec  6 16:14:38.276356  jlink:
Dec  6 16:14:38.276356  jlink:         ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
Dec  6 16:14:38.276356  jlink:         ♌ ♌ ♌ ♌  𝔑𝔘𝔠𝔏𝔢𝔒-F030R8   ♌ ♌ ♌ ♌
Dec  6 16:14:38.276356  jlink:         🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀 🍀
Dec  6 16:14:38.276356  jlink:
Dec  6 16:14:38.276356  jlink:
Dec  6 16:14:38.276356  jlink:       triceExamples.c    61             16334 TRICE_DIRECT_OUTPUT == 1,
TRICE_DEFERRED_OUTPUT == 0
Dec  6 16:14:38.276356  jlink:       triceExamples.c    63             16333 TRICE_STACK_BUFFER,
TRICE_MULTI_PACK_MODE
Dec  6 16:14:38.276920  jlink:       triceExamples.c    76             16327 _CYCLE == 1, _PROTECT == 1,
_DIAG == 1, XTEA == 0
Dec  6 16:14:38.277424  jlink:       triceExamples.c    77             16326 _SINGLE_MAX_SIZE=104,
_BUFFER_SIZE=172, _DEFERRED_BUFFER_SIZE=1024
Dec  6 16:14:39.181228  jlink:       triceExamples.c    29    0,031_848 16356 🐔 Speedy Gonzales a  32-bit
```

```
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    30    0,031_292 16355 🐎 Speedy Gonzales b  32-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    31    0,030_736 16354 🐎 Speedy Gonzales c  32-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    32    0,030_180 16353 🐎 Speedy Gonzales d  32-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    33       29_624 16352 🐎 Speedy Gonzales e  16-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    34       29_066 16351 🐎 Speedy Gonzales f  16-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    35       28_508 16350 🐎 Speedy Gonzales g  16-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    36       27_950 16349 🐎 Speedy Gonzales h  16-bit
time stamp
Dec  6 16:14:39.181228  jlink:       triceExamples.c    42       27_086 16344 2.71828182845904523536 <-
float number as string
Dec  6 16:14:39.181228  jlink:       triceExamples.c    43       25_906 16343 2.71828182845904509080
(double with more ciphers than precision)
Dec  6 16:14:39.181798  jlink:       triceExamples.c    44       25_305 16342 2.71828174591064453125
(float  with more ciphers than precision)
Dec  6 16:14:39.181798  jlink:       triceExamples.c    45       24_727 16341 2.718282 (default rounded
float)
Dec  6 16:14:39.181798  jlink:       triceExamples.c    46       24_148 16340 A Buffer:
Dec  6 16:14:39.181798  jlink:       triceExamples.c    47       23_578 16339 32 2e 37 31 38 32 38 31 38
32 38 34 35 39 30 34 35 32 33 35 33 36
Dec  6 16:14:39.181798  jlink:       triceExamples.c    48       22_394 16338 31372e32  31383238  34383238
34303935  35333235
Dec  6 16:14:39.181798  jlink:       triceExamples.c    49       21_295 16337 ARemoteFunctionName(2e32)
(3137)(3238)(3138)(3238)(3438)(3935)(3430)(3235)(3533)(3633)
Dec  6 16:14:39.181798  jlink:       triceExamples.c    50              16200 5 times a 16 byte long Trice
messages, which may not be written all if the buffer is too small:
Dec  6 16:14:39.182303  jlink:       triceExamples.c    52       19_555 16335 i=44444400 aaaaaa00
Dec  6 16:14:39.182303  jlink:       triceExamples.c    52       18_941 16335 i=44444401 aaaaaa01
Dec  6 16:14:39.182303  jlink:       triceExamples.c    52       18_327 16335 i=44444402 aaaaaa02
Dec  6 16:14:39.182834  jlink:       triceExamples.c    52       17_713 16335 i=44444403 aaaaaa03
Dec  6 16:14:39.182834  jlink:       triceExamples.c    52       17_099 16335 i=44444404 aaaaaa04
Dec  6 16:14:40.187121  jlink:       triceExamples.c    29    0,031_855 16356 🐎 Speedy Gonzales a  32-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    30    0,031_299 16355 🐎 Speedy Gonzales b  32-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    31    0,030_743 16354 🐎 Speedy Gonzales c  32-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    32    0,030_187 16353 🐎 Speedy Gonzales d  32-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    33       29_631 16352 🐎 Speedy Gonzales e  16-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    34       29_073 16351 🐎 Speedy Gonzales f  16-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    35       28_515 16350 🐎 Speedy Gonzales g  16-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    36       27_957 16349 🐎 Speedy Gonzales h  16-bit
time stamp
Dec  6 16:14:40.187121  jlink:       triceExamples.c    42       27_093 16344 2.71828182845904523536 <-
float number as string
Dec  6 16:14:40.187121  jlink:       triceExamples.c    43       25_913 16343 2.71828182845904509080
(double with more ciphers than precision)
Dec  6 16:14:40.187121  jlink:       triceExamples.c    44       25_310 16342 2.71828174591064453125
(float  with more ciphers than precision)
Dec  6 16:14:40.187121  jlink:       triceExamples.c    45       24_730 16341 2.718282 (default rounded
float)
Dec  6 16:14:40.187121  jlink:       triceExamples.c    46       24_149 16340 A Buffer:
Dec  6 16:14:40.187121  jlink:       triceExamples.c    47       23_577 16339 32 2e 37 31 38 32 38 31 38
32 38 34 35 39 30 34 35 32 33 35 33 36
Dec  6 16:14:40.187630  jlink:       triceExamples.c    48       22_391 16338 31372e32  31383238  34383238
34303935  35333235
Dec  6 16:14:40.187690  jlink:       triceExamples.c    49       21_290 16337 ARemoteFunctionName(2e32)
(3137)(3238)(3138)(3238)(3438)(3935)(3430)(3235)(3533)(3633)
```

```
Dec  6 16:14:40.187690  jlink:       triceExamples.c    50             16200 5 times a 16 byte long Trice
messages, which may not be written all if the buffer is too small:
Dec  6 16:14:40.187690  jlink:       triceExamples.c    52       19_546 16335 i=44444400 aaaaaa00
Dec  6 16:14:40.188195  jlink:       triceExamples.c    52       18_930 16335 i=44444401 aaaaaa01
Dec  6 16:14:40.188195  jlink:       triceExamples.c    52       18_314 16335 i=44444402 aaaaaa02
Dec  6 16:14:40.188195  jlink:       triceExamples.c    52       17_698 16335 i=44444403 aaaaaa03
Dec  6 16:14:40.188195  jlink:       triceExamples.c    52       17_082 16335 i=44444404 aaaaaa04
Dec  6 16:14:41.191648  jlink:    triceLogDiagData.c    21             16382 RTT0_writeDepthMax=325
(BUFFER_SIZE_UP=1024)
Dec  6 16:14:41.191648  jlink:    triceLogDiagData.c    44             16378 triceSingleDepthMax = 108 of
172 (TRICE_BUFFER_SIZE)
Dec  6 16:14:41.191648  jlink:       triceExamples.c    29    0,030_628 16356 🐔 Speedy Gonzales a  32-bit
time stamp
Dec  6 16:14:41.191648  jlink:       triceExamples.c    30    0,030_072 16355 🐔 Speedy Gonzales b  32-bit
time stamp
```

"🐔 Speedy Gonzales" needs about 500 MCU clocks.

27.6. A developer setting, only enabling SEGGER_RTT and without deferred output gives after running `./build.sh TRICE_DIAGNOSTICS=0 TRICE_PROTECT=0`:

```
arm-none-eabi-size out/F030_inst.elf
   text    data     bss     dec     hex filename
   5796      16    2736    8548    2164 out/F030_inst.elf
```

That is nearly 1 KB less Flash needs.

The output:

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice_wt_devel/examples/F030_inst (devel)
$ trice l -p jlink -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0" -pf none -d16 -showID
"deb:%5d"
Dec  6 16:20:10.545274  jlink:       triceExamples.c    12       65_535 16369  Hello! 👌😊
Dec  6 16:20:10.545274  jlink:
Dec  6 16:20:10.545274  jlink:         ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
Dec  6 16:20:10.545274  jlink:         ♌ ♌ ♌ ♌  𝔑𝔘𝔈𝔏𝔈𝔒-F030R8   ♌ ♌ ♌ ♌
Dec  6 16:20:10.545274  jlink:         🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃🍃
Dec  6 16:20:10.545274  jlink:
Dec  6 16:20:10.545274  jlink:
Dec  6 16:20:10.545274  jlink:       triceExamples.c    61             16334 TRICE_DIRECT_OUTPUT == 1,
TRICE_DEFERRED_OUTPUT == 0
Dec  6 16:20:10.545274  jlink:       triceExamples.c    63             16333 TRICE_STACK_BUFFER,
TRICE_MULTI_PACK_MODE
Dec  6 16:20:10.545890  jlink:       triceExamples.c    76             16327 _CYCLE == 1, _PROTECT == 0,
_DIAG == 0, XTEA == 0
Dec  6 16:20:10.546396  jlink:       triceExamples.c    77             16326 _SINGLE_MAX_SIZE=104,
_BUFFER_SIZE=172, _DEFERRED_BUFFER_SIZE=1024
Dec  6 16:20:11.448885  jlink:       triceExamples.c    29    0,031_859 16356 🐔 Speedy Gonzales a  32-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    30    0,031_661 16355 🐔 Speedy Gonzales b  32-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    31    0,031_463 16354 🐔 Speedy Gonzales c  32-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    32    0,031_265 16353 🐔 Speedy Gonzales d  32-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    33       31_067 16352 🐔 Speedy Gonzales e  16-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    34       30_867 16351 🐔 Speedy Gonzales f  16-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    35       30_667 16350 🐔 Speedy Gonzales g  16-bit
time stamp
Dec  6 16:20:11.448885  jlink:       triceExamples.c    36       30_467 16349 🐔 Speedy Gonzales h  16-bit
time stamp
```

```
Dec  6 16:20:11.549660  jlink:       triceExamples.c    42       29_961 16344 2.71828182845904523536 <-
float number as string
Dec  6 16:20:11.549660  jlink:       triceExamples.c    43       29_141 16343 2.7182818284590 4509080
(double with more ciphers than precision)
Dec  6 16:20:11.549660  jlink:       triceExamples.c    44       28_897 16342 2.71828174591064453125
(float  with more ciphers than precision)
Dec  6 16:20:11.549660  jlink:       triceExamples.c    45       28_675 16341 2.718282 (default rounded
float)
Dec  6 16:20:11.549660  jlink:       triceExamples.c    46       28_452 16340 A Buffer:
Dec  6 16:20:11.550166  jlink:       triceExamples.c    47       28_238 16339 32 2e 37 31 38 32 38 31 38
32 38 34 35 39 30 34 35 32 33 35 33 36
Dec  6 16:20:11.550247  jlink:       triceExamples.c    48       27_412 16338 31372e32  31383238  34383238
34303935  35333235
Dec  6 16:20:11.550247  jlink:       triceExamples.c    49       26_671 16337 ARemoteFunctionName(2e32)
(3137)(3238)(3138)(3238)(3438)(3935)(3430)(3235)(3533)(3633)
Dec  6 16:20:11.550247  jlink:       triceExamples.c    50              16200 5 times a 16 byte long Trice
messages, which may not be written all if the buffer is too small:
Dec  6 16:20:11.550754  jlink:       triceExamples.c    52       25_646 16335 i=44444400 aaaaaa00
Dec  6 16:20:11.550754  jlink:       triceExamples.c    52       25_389 16335 i=44444401 aaaaaa01
Dec  6 16:20:11.550754  jlink:       triceExamples.c    52       25_132 16335 i=44444402 aaaaaa02
Dec  6 16:20:11.551285  jlink:       triceExamples.c    52       24_875 16335 i=44444403 aaaaaa03
Dec  6 16:20:11.551285  jlink:       triceExamples.c    52       24_618 16335 i=44444404 aaaaaa04
Dec  6 16:20:12.453968  jlink:       triceExamples.c    29    0,031_859 16356 🐁 Speedy Gonzales a  32-bit
time stamp
Dec  6 16:20:12.453968  jlink:       triceExamples.c    30    0,031_661 16355 🐁 Speedy Gonzales b  32-bit
time stamp
```

"🐁 Speedy Gonzales" direct outout needs about 200 MCU clocks and not 500 as before.

27.7. Settings Conclusion

- 4-8 KB Flash and 1.2 KB RAM needed for the Trice library.
- The RAM size is mainly influenced by the configured buffer sizes.
- Switching off diagnostics and/or protection is ok for less memory needs and faster Trice execution after getting some experience with the project.

27.8. Legacy Trice Space Example (Old Version)

- STM32CubeMX generated empty default project: `Program Size: Code=2208 RO-data=236 RW-data=4 ZI-data=1636`
- Same project with default `Trice` instrumentation: `Program Size: Code=2828 RO-data=236 RW-data=44 ZI-data=1836`
- Needed FLASH memory: 620 Bytes
- Needed RAM: 40 Bytes plus 200 Bytes for the 2 times 100 Bytes double buffer
- With increased/decreased buffers also more/less RAM is needed.
- With each additional Trice macro a few additional FLASH memory bytes, like 10 assembler instructions, are needed.
- No printf-like library code is used anymore.
- No format strings go into the target code anymore.
- In general Trice instrumentation **reduces** the needed memory compared to a printf-like implementation.

27.9. Memory Needs for Old Example 1

The following numbers are measured with a legacy encoding, showing that the instrumentation code can be even smaller.

Program Size (STM32-F030R8 demo project)	**trice instrumentation**	**buffer size**	**compiler optimize for time**	**comment**
Code=1592 RO-data=236 RW-data= 4 ZI-data=1028	none	0	off	CubeMX generated, no trice
Code=1712 RO-data=240 RW-data=24 ZI-data=1088	core	64	off	core added without trices
Code=3208 RO-data=240 RW-data=36 ZI-data=1540	TriceCheckSet()	512	off	TRICE_SHORT_MEMORY is 1 (small)

Program Size (STM32-F030R8 demo project)	trice instrumentation	buffer size	compiler optimize for time	comment
Code=3808 RO-data=240 RW-data=36 ZI-data=1540	TriceCheckSet()	512	on	TRICE_SHORT_MEMORY is 0 (fast)

- The core instrumentation needs less 150 bytes FLASH and about 100 bytes RAM when buffer size is 64 bytes.
- The about 50 trices in TriceCheckSet() allocate roughly 2100 (fast mode) or 1500 (small mode) bytes.
- trices are removable without code changes with `#define TRICE_OFF 1` before `incude "trice.h"` on file level or generally on project level.

27.10. Memory Needs for Old Example 2

Project	Compiler	Optimization	Link-Time-Optimization	Result	Remark
MDK-ARM_STM32F030_bareerated	CLANG v6.19	-Oz	yes	Code=1020 RO-data=196 RW-data=0 ZI-data=1024	This is the plain generated project without trice instrumentation.
MDK-ARM_STM32F030_instrumented	CLANG v6.19	-Oz	yes	Code=4726 RO-data=238 RW-data=16 ZI-data=4608	This is with full trice instrumentation with example messages.

- The size need is less than 4 KB. See also Trice Project Image Size Optimization.

(back to top)

28. Trice Project Image Size Optimization

Modern compilers are optimizing out unused code automatically, but you can help to reduce trice code size if your compiler is not perfect.

28.1. Code Optimization -o3 or -oz (if supported)

For debugging it could be helpful to switch off code optimization what increases the code size. A good choice is `-o1`. See also TRICE_STACK_BUFFER could cause stack overflow with -o0 optimization.

28.2. Compiler Independent Setting (a bit outdated)

Maybe the following is a bit unhandy but it decreases the code amount, build time and the image size.

- For **X=8|16|32|64** and **N=0...12** selectively set `#define ENABLE_trice`**X**`fn_`**N** `1` to `0` for unused functions in project specific file `triceConfig.h`.
- For **X=8|16|32|64** and **N=0...12** selectively set `#define ENABLE_Trice`**X**`fn_`**N** `1` to `0` for unused functions in project specific file `triceConfig.h`.
- For **X=8|16|32|64** and **N=0...12** selectively set `#define ENABLE_TRice`**X**`fn_`**N** `1` to `0` for unused functions in project specific file `triceConfig.h`.

When having lots of program memory simply let all values be `1`. With specific linker optimization unused functions can get stripped out automatically.

It is possible to `#define TRICE_SINGLE_MAX_SIZE 12` for example in *triceConfig.h*. This automaticaly disables all Trice messages with payloads > 8 bytes (Trice size is 4 bytes).

28.3. Linker Option --split-sections (if supported)

In ARM-MDK uVision `Project -> Options -> C/C++ -> "One EFL section for each function"` allows good optimization and getting rid of unused code without additional linker optimization. This leads to a faster build process and is fine for most cases. It allows excluding unused functions.

28.4. Linker Optimization -flto (if supported)

- To get the smallest possible image, do *not* use option `--split sections`.
- Use linker optimization alone.

- This increases the build time but reduces the image size significantly.

28.4.1. ARMCC Compiler v5 Linker Feedback

- In ARM-MDK uVision, when using ARMCC compiler v5, there is a check box `Project -> Options -> Target -> "Cross Module Optimization"`.
- In ARMCC this works also with the lite version.

28.4.2. ARMCLANG Compiler v6 Link-Time Optimization

- In ARM-MDK uVision, when using ARMCLANG compiler v6, the check box `Project -> Options -> C/C++(AC6) -> "Link-Time Optimization"` is usable to set the CLI `-flto` switch.
- LTO is not possible with ARMCLANG6 lite: https://developer.arm.com/documentation/ka004054/latest.

28.4.3. GCC

With GCC use the `-flto` CLI switch directly.

28.4.4. LLVM ARM Clang

This compiler is much faster and creates the smallest images. Right now it uses the GCC libs and linker.

28.4.5. Other IDE´s and compilers

Please check the manuals and create a pull request or simply let me know.

28.5. Legacy STM32F030 Example Project - Different Build Sizes

28.5.1. ARMCC compiler v5

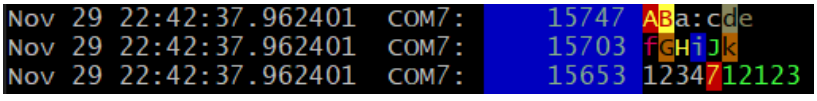
Compiler	Linker	Result	Comment
o0		Code=46942 RO-data=266 RW-data=176 ZI-data=4896	very big
o1		Code=22582 RO-data=258 RW-data=168 ZI-data=4896	
o3		Code=21646 RO-data=258 RW-data=168 ZI-data=4896	
o0	split sections	Code= 7880 RO-data=268 RW-data=156 ZI-data=4892	for debugging
o1	split sections	Code= 5404 RO-data=260 RW-data=148 ZI-data=4892	**for debugging**
o3	split sections	Code= 4996 RO-data=260 RW-data=148 ZI-data=4892	**good balance**
o0	flto	Code= 8150 RO-data=266 RW-data=176 ZI-data=4896	builds slower
o1	flto	Code= 5210 RO-data=258 RW-data=148 ZI-data=4892	builds slower
o3	flto	Code= 4818 RO-data=258 RW-data=148 ZI-data=4892	builds slower, **smallest image**

(back to top)

29. Trice Tags and Color

29.1. How to get

- Add a tag name as color descriptor in front of each Trice format string like `"wrn:Peng!"`.
- In file ../internal/emitter/lineTransformerANSI.go the colors are changeable and additional color tags definable.
- It is possible to concatenate single colorized letters to get output like this:

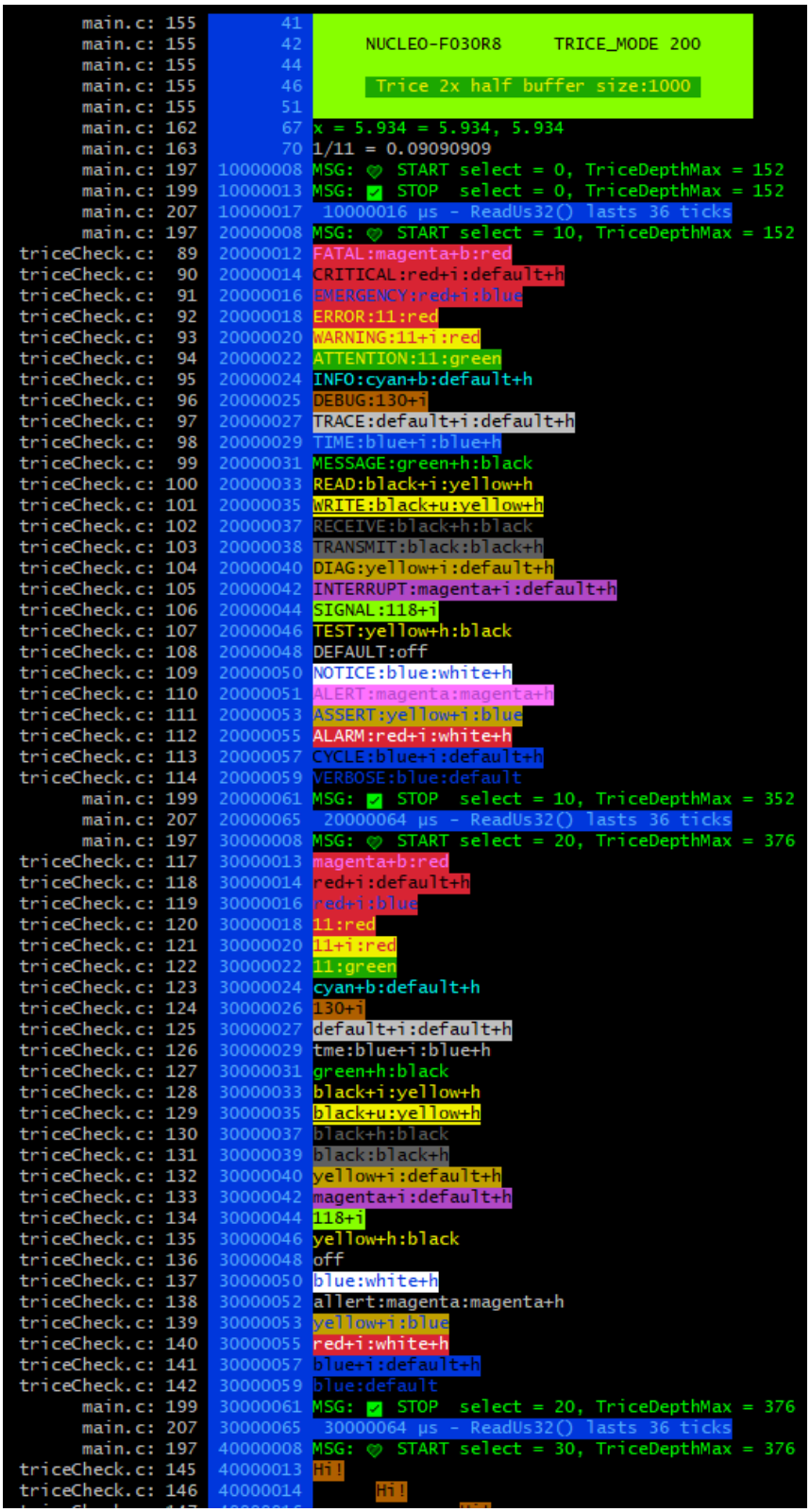


- ../_test/testdata/triceCheck.c contains the code for this example.

- The Trice tool, if knowing `wrn:` as pattern, prepends the appropriate color code. It removes the sequence `wrn:`, if it is known and completely lower case.
    - The Trice tool will strip full lowercase tag descriptors from the format string after setting the appropriate color, making it possible to give even each letter in a message its color.

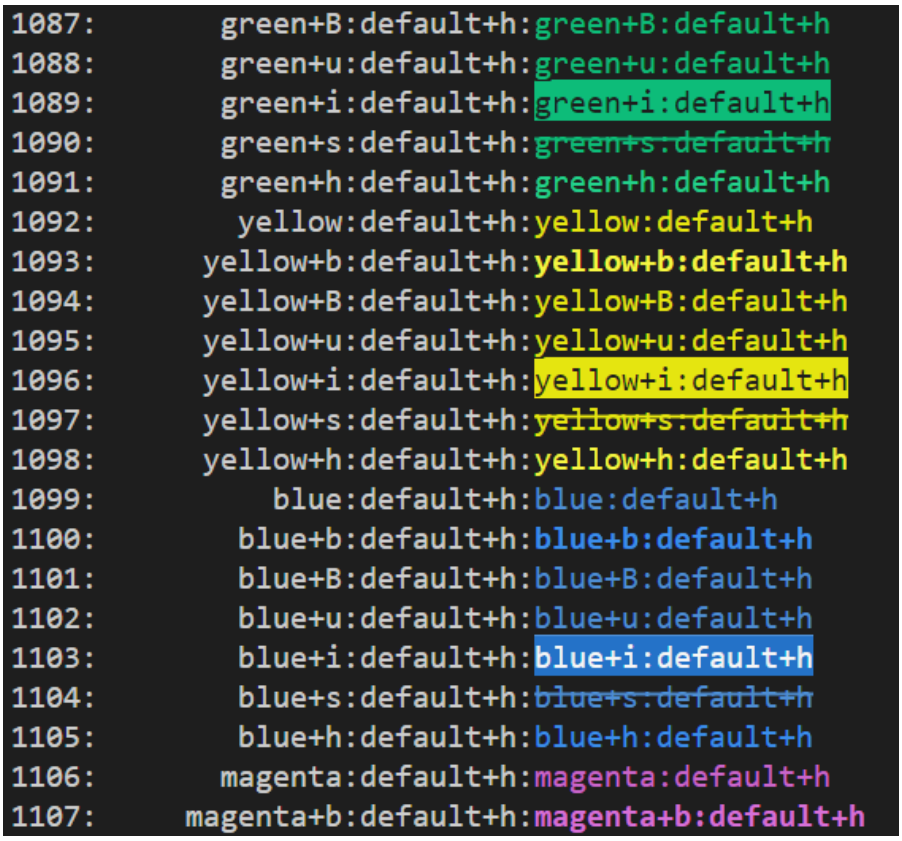
      `"wrn:fox"` will display colored "fox" `"Wrn:fox"` will display colored "Wrn:fox"
- The user can define any pattern with any color code to create colored output with the Trice tool.
- There is no tag enable switch inside the target code. It would need a back channel and add overhead.
- An option using tag specific ID ranges with optional routing exists.
- The Trice tool offers the 2 command line switches `-pick` and `-ban` to control tag visualization during runtime.

29.1.1. Output options



29.1.2. Check Alternatives

There are over 1000 possibilities:



To see them all run `trice generate -color`. Only file ../internal/emitter/lineTransformerANSI.go needs to be changed and the Trice tool needs to be rebuild afterwards: `go install ./...`. If you design a good looking flavour, feel free to propose it.

29.2. Color issues under Windows

Currently console colors are not enabled by default in Win10, so if you see no color but escape sequences on your powershell or cmd window, please refer to Windows console with ANSI colors handling or simply use a Linux like terminal under windows, like git-bash. One option is also to install Microsoft *Windows Terminal (Preview)* from inside the Microsoft store and to start the Trice tool inside there. Unfortunately this can not be done automatically right now because of missing command line switches. Alacritty is one of other alternatives.

(back to top)

30. Trice without UART

A very performant output path is RTT, if your MCU supports background memory access like the ARM-M ones.

Because the Trice tool needs only to receive, a single target UART-TX pin will do. But it is also possible to use a GPIO-Pin for Trice messages without occupying a UART resource.

- This slow path is usable because a Trice needs only few bytes for transmission.
- You can transmit each basic trice (4 or 8 bytes) as bare message over one pin:

Figure 1 shows the transitions to transmit an arbitrary binary value in biphase encoding—to be more specific, this variant is biphase mark coding. In the biphase encoding scheme, there is a transition at the start of each bit.

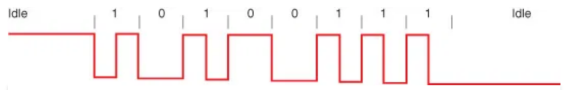


Figure 1: This is an example of a binary value transferred in biphase mark coding.

For a 1 bit there is also a transition halfway through the clock period. With a 0 bit, there is no extra transition. The absolute levels in biphase encoding are irrelevant, only the changes in the output line are important. In the previous example, the transmission starts with the idle state at a high logic level but ends in an idle state at a low logic level.

```
void trace_out(int byte, int iopin)
{
        int bit;
        for (bit = 0; bit < 8; bit++) {
                toggle_pin(iopin);
                trace_delay();
                if (byte & 0x80)
                        toggle_pin(iopin);
trace_delay();
byte <<= 1;
        }
}
```

Listing 1: Transmitting a byte in biphase encoding, based on a function to toggle an I/O pin, is shown.

*The biphase encoding signal goes from the DUT to a trace dongle. The dongle decodes the signal and forwards it as serial data from a virtual RS-232 port to the workstation (see **Photo 2** and the circuit in **Figure 2**).*

- The 2 images are taken from https://circuitcellar.com/cc-blog/a-trace-tool-for-embedded-systems/. See there for more information.
- As Trice dongle you can use any spare MCU board with an UART together with an FTDI USB converter.
    - This allowes also any other data path - method does'nt matter:
      UART,
      I^2C,
      SPI,
      GPIO,
      CAN,
      LIN, ...
- RTT is also a possible path to use - see Trice over RTT for options.

(back to top)

31. Trice over RTT

> Allows Trice over the debug probe without using a pin or UART.

- RTT works good with a SEGGER J-Link debug probe but needs some closed source software components.
- Also ST-Link is usable for Trice logs, but maybe not parallel with debugging.
- Most investigations where done with a NUCLEO64-STM32F030R8 evaluation board which contains an on-board debug probe reflashed with a SEGGER J-Link OB software (see below).
    - When using very high Trice loads over RTT for a long time, sometimes an on-board J-Link (re-flashed ST-Link) could get internally into an inconsistent state (probably internal buffer overrun), what needs a power cycle then.
- You could consider RTT over open-OCD as an alternative.
- The default SEGGER up-buffer size is 1024 bytes, good for most cases. If not, adapt it in your *triceConfig.h* file **AND** in the *SEGGER_RTT_Conf.h* file: You need only one up-channel for Trice:

```
#define BUFFER_SIZE_UP (128)  // "TRICE_DIRECT_BUFFER_SIZE"
```

- Inside the triceDefaultConfig.h you can find some other settings recommended for the *SEGGER_RTT_Conf.h* file. You have to set them manually in the *SEGGER_RTT_Conf.h* because the SEGGER target sources do not include *trice.h* (and implicit triceDefaultConfig.h and

triceConfig.h).

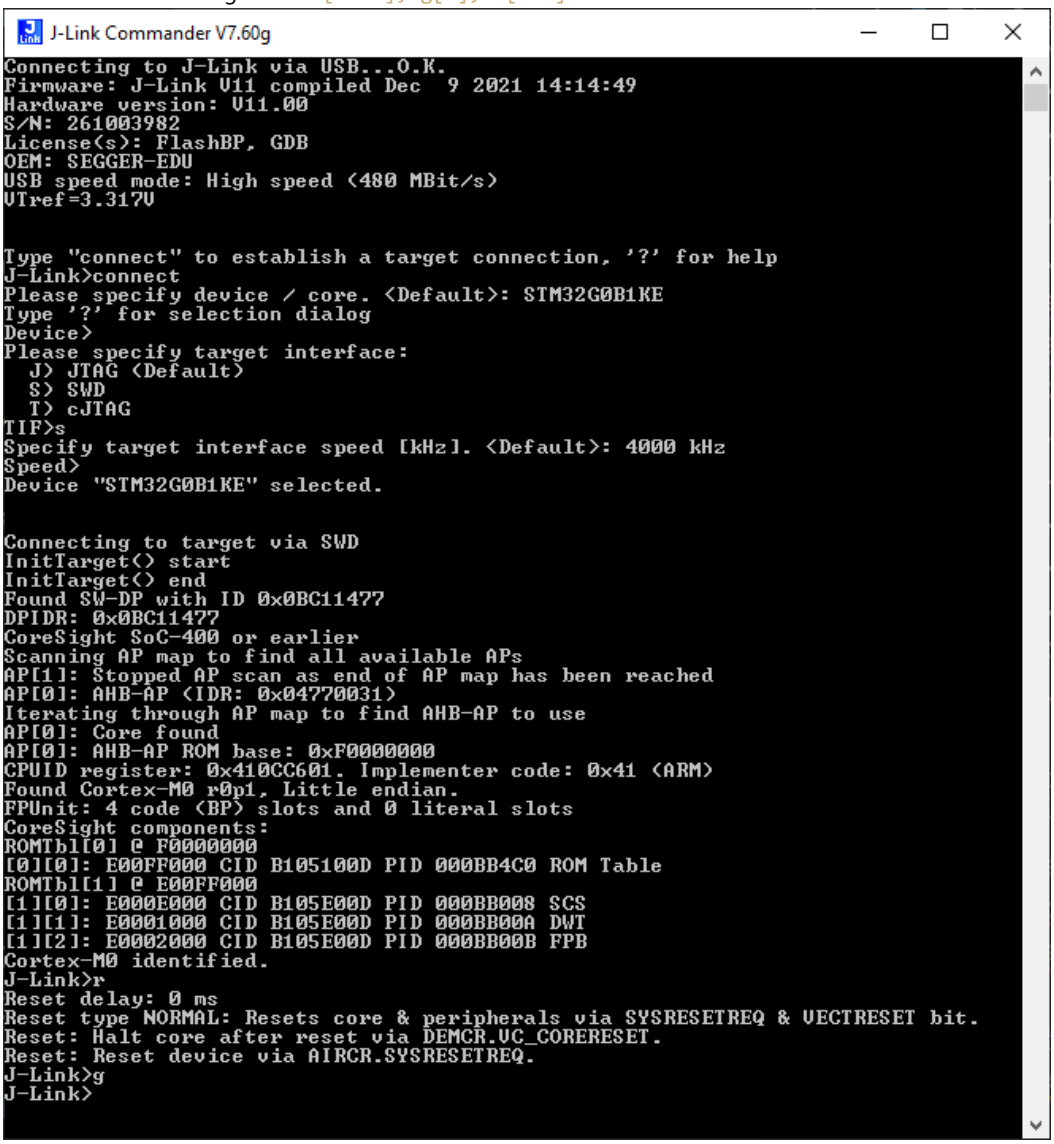
- **Possible:** Parallel usage of RTT direct mode with UART deferred mode. You can define `TRICE_UARTA_MIN_ID` and `TRICE_UARTA_MAX_ID` inside triceConfig.h to log only a specific ID range over UARTA in deferred mode for example. (#446)

(back to top)

31.1. For the impatient (2 possibilities)

The default SEGGER tools only suport RTT channel 0.

31.1.1. Start JLink commander and connect over TCP

- JLink.exe → `connect ⏎ ⏎ S ⏎` and keep it active.
  - You can control the target with `r[eset], g[o], h[alt]` and use other commands too.
  - 

- Start in Git-Bash or s.th. similar: `trice l -p TCP4 -args localhost:19021`
- You may need a Trice tool restart after firmware reload.

Setup TCP4 server providing the trace data

This is just the SEGGER J-Link server here for demonstration, but if your target device has an TCP4 interface, you can replace this with your target server.

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice (main)
$ jlink
SEGGER J-Link Commander V7.92g (Compiled Sep 27 2023 15:36:46)
DLL version V7.92g, compiled Sep 27 2023 15:35:10
```

```
Connecting to J-Link via USB...O.K.
Firmware: J-Link STLink V21 compiled Aug 12 2019 10:29:20
Hardware version: V1.00
J-Link uptime (since boot): N/A (Not supported by this model)
S/N: 770806762
VTref=3.300V


Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: STM32G0B1RE
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "STM32G0B1RE" selected.


Connecting to target via SWD
InitTarget() start
SWD selected. Executing JTAG -> SWD switching sequence.
DAP initialized successfully.
InitTarget() end - Took 36.3ms
Found SW-DP with ID 0x0BC11477
DPv0 detected
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x04770031)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xF0000000
CPUID register: 0x410CC601. Implementer code: 0x41 (ARM)
Found Cortex-M0 r0p1, Little endian.
FPUnit: 4 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl[0] @ F0000000
[0][0]: E00FF000 CID B105100D PID 000BB4C0 ROM Table
ROMTbl[1] @ E00FF000
[1][0]: E000E000 CID B105E00D PID 000BB008 SCS
[1][1]: E0001000 CID B105E00D PID 000BB00A DWT
[1][2]: E0002000 CID B105E00D PID 000BB00B FPB
Memory zones:
  Zone: "Default" Description: Default access mode
Cortex-M0 identified.
J-Link>
```

Now the TCP4 server is running and you can start the Trice tool as TCP4 client, which connects to the TCP4 server to receive the binary log data:

```
$ trice l -p TCP4 -args="127.0.0.1:19021" -til ../examples/G0B1_inst/til.json -li
../examples/G0B1_inst/li.json -d16 -pf none
```

In this **G0B1_inst** example we use the additional `-d16` and `-pf none` switches to decode the RTT data correctly.

This is a demonstration and test for the `-port TCP4` usage possibility. Using RTT with J-Link is more easy possible as shown in the next point.

31.1.2. Start using JLinkRTTLogger

- Start inside Git-Bash or s.th. similar: `trice l -p JLINK -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0"`
  - Replace CLI details with your settings.
  - For **G0B1_inst**: `trice l -p JLINK -args "-Device STM32G0B1RE -if SWD -Speed 4000 -RTTChannel 0" -d16 -pf none`
  - You can add the `-verbose` CLI switch for more details.
- You may **not** need a Trice tool restart after firmware reload.

31.1.3. JLinkRTTLogger Issue

- For some reason the RTT technique does not work well with Darwin (MacOS) and also Linux right now. The problem seems to be that the JLinkRTTLogger app cannot work correctly in the background. But there is a workaround:
  - Example 1:
    - In one terminal run `JLinkRTTLogger -Device STM32G0B1RE -if SWD -Speed 4000 -RTTChannel 0 myLogFile.bin`
    - and in an other terminal execute `trice l -p FILE -args myLogFile.bin -pf none -d16`.
  - Example 2:
    - Flash, start debugger and run to main()
    - Terminal 1: `rm ./temp/trice.bin && JLinkRTTLogger -Device STM32G0B1RE -If SWD -Speed 4000 -RTTChannel 0 ./temp/trice.bin`
    - Terminal 2: `touch ./temp/trice.bin && trice log -p FILE -args ./temp/trice.bin -prefix off -hs off -d16 -ts ms -i ../../demoTIL.json -li ../../demoLI.json -pf none`
    - Continue to run in debugger
    - Terminal 1:

```
th@P51-DebianKDE:~/repos/trice/examples/G0B1_inst$ rm ./temp/trice.bin && JLinkRTTLogger -
Device STM32G0B1RE -If SWD -Speed 4000 -RTTChannel 0 ./temp/trice.bin
SEGGER J-Link RTT Logger
Compiled Dec 18 2024 15:48:21
(c) 2016-2017 SEGGER Microcontroller GmbH, www.segger.com
        Solutions for real time microcontroller applications

Default logfile path: /home/th/.config/SEGGER

----------------------------------------------------------


----------------------------------------------------------
Connected to:
  SEGGER J-Link ST-LINK
  S/N: 779220206

Searching for RTT Control Block...OK.
1 up-channels found:
0: Terminal
Selected RTT Channel description:
  Index: 0
  Name:  Terminal
  Size:  1024 bytes.

Output file: ./temp/trice.bin

Getting RTT data from target. Press any key to quit.
----------------------------------------------------------

Transfer rate: 0 Bytes/s Data written: 15.71 KB
```

    - Terminal 2:

```
th@P51-DebianKDE:~/repos/trice/examples/G0B1_inst$ touch ./temp/trice.bin && trice log -p
FILE -args ./temp/trice.bin -prefix off -hs off -d16 -ts ms  -i ../../demoTIL.json -li
../../demoLI.json -pf none
      triceExamples.c    12        0,000  Hello! 👌😀
        ✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨
        🎈🎈🎈🎈  NUCLEO-G0B1RE    🎈🎈🎈🎈
        🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀


      triceExamples.c    61              TRICE_DIRECT_OUTPUT == 1, TRICE_DEFERRED_OUTPUT ==
1
      triceExamples.c    69              TRICE_RING_BUFFER, TRICE_MULTI_PACK_MODE
      triceExamples.c    76              _CYCLE == 1, _PROTECT == 1, _DIAG == 1, XTEA == 1
      triceExamples.c    77              _SINGLE_MAX_SIZE=104, _BUFFER_SIZE=172,
_DEFERRED_BUFFER_SIZE=2000
      triceExamples.c    29  0:00:00,003 🐔 Speedy Gonzales a  32-bit time stamp
      triceExamples.c    30  0:00:00,003 🐔 Speedy Gonzales b  32-bit time stamp
      triceExamples.c    31  0:00:00,003 🐔 Speedy Gonzales c  32-bit time stamp
      triceExamples.c    32  0:00:00,003 🐔 Speedy Gonzales d  32-bit time stamp
      triceExamples.c    33        0,310 🐔 Speedy Gonzales e  16-bit time stamp
      triceExamples.c    34        0,328 🐔 Speedy Gonzales f  16-bit time stamp
      triceExamples.c    35        0,347 🐔 Speedy Gonzales g  16-bit time stamp
      triceExamples.c    36        0,365 🐔 Speedy Gonzales h  16-bit time stamp
      triceExamples.c    42        0,394 2.71828182845904523536 <- float number as string
      triceExamples.c    43        0,436 2.71828182845904509080 (double with more ciphers
than precision)
      triceExamples.c    44        0,458 2.71828174591064453125 (float  with more ciphers
than precision)
      triceExamples.c    45        0,479 2.718282 (default rounded float)
      triceExamples.c    46        0,500 A Buffer:
      triceExamples.c    47        0,520 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35
32 33 35 33 36
      triceExamples.c    48        0,562 31372e32  31383238  34383238  34303935  35333235
      triceExamples.c    49        0,601 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)
(3438)(3935)(3430)(3235)(3533)(3633)
      triceExamples.c    50              3 times a 16 byte long Trice messages, which may
not be written all if the buffer is too small:
      triceExamples.c    52        0,664 i=44444400 aaaaaa00
      triceExamples.c    52        0,687 i=44444401 aaaaaa01
      triceExamples.c    52        0,709 i=44444402 aaaaaa02
              main.c   312  0:00:00,003 StartDefaultTask
              main.c   339  0:00:00,003 StartTask02:Diagnostics and TriceTransfer
  triceLogDiagData.c    21              RTT0_writeDepthMax=365 (BUFFER_SIZE_UP=1024)
  triceLogDiagData.c    44              triceSingleDepthMax =  96 of 172
(TRICE_BUFFER_SIZE)
  triceLogDiagData.c    75              triceRingBufferDepthMax =   0 of 2000
        triceCheck.c    57               line 57
        triceCheck.c    59  0:00:02,325 Hello World!
        triceCheck.c    61  0:00:02,405 This is a message without values and a 32-bit
stamp.
        triceCheck.c    62        0,306 This is a message without values and a 16-bit
stamp.
        triceCheck.c    63              This is a message without values and without stamp.
```

  - See also the configuration in ./examples/G0B1_inst/Core/Inc/triceConfig.h

- If you install the `tmux` command your life gets esier by using a shell script like ./examples/G0B1_inst/RTTLogTmux.sh:

```
mkdir -p ./temp
rm -f ./temp/trice.bin
touch ./temp/trice.bin
tmux new -s "tricerttlog" -d "JLinkRTTLogger -Device STM32G0B1RE -If SWD -Speed 4000 -RTTChannel 0
./temp/trice.bin"
trice log -p FILE -args ./temp/trice.bin -pf none -prefix off -hs off -d16 -ts16 "time:offs:%4d µs" -
```

```
showID "deb:%5d" -i ../../demoTIL.json -li ../../demoLI.json -stat
tmux kill-session -t "tricerttlog"
```

- Usage:

```
th@PaulPCdeb128KDE:~/repos/trice/examples/G0B1_inst$ ./RTTLogUnix.sh
    triceExamples.c    12        0,000  Hello! 👌😊
      ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
      🎈 🎈 🎈 🎈   NUCLEO-G0B1RE    🎈 🎈 🎈 🎈
      🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿


    triceExamples.c    61              TRICE_DIRECT_OUTPUT == 1, TRICE_DEFERRED_OUTPUT == 1
    triceExamples.c    69              TRICE_RING_BUFFER, TRICE_MULTI_PACK_MODE
    triceExamples.c    76              _CYCLE == 1, _PROTECT == 1, _DIAG == 1, XTEA == 1
    triceExamples.c    77              _SINGLE_MAX_SIZE=104, _BUFFER_SIZE=172,
_DEFERRED_BUFFER_SIZE=2000
    triceExamples.c    29  0:00:00,002 🐆 Speedy Gonzales a  32-bit time stamp
```

- **Hint:** If you use *RTTLogTmux.sh* with Darwin (MacOS), the "control-C" key combination seems not to work immediately. That is simply because the keyboard focus switches away after script start. Simply click into the terminal window again and then use "control-C" to terminate the Trice logging.

(back to top)

31.2. Segger Real Time Transfer (RTT)

- Prerequisite is a processor with memory background access support like ARM Cortex-M cores.
- If you can use a Segger J-Link or an STM ST-Link debug probe (ST Microelectronics eval boards have it) this is an easy and fast way to use Trice without any UART or other port.
- Detailed description can be found in document UM08001_JLink.pdf in chapter 16 which is part of https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack.
- Following examples are for Windows, but should work similar also on Linux and Darwin (MacOS).
- Trice can use the Segger RTT protocol in different ways.
  - Hardware paths:
    - Use J-Link or J-Link OB (on-board). J-Link OB can be flashed to many ST Microelectronics evaluation boards (v2.0 link hardware) and for example is also usable with NXP and Atmel. For that you can also use a spare STM32 evaluation board (10 EUR) with jumper changes and breakout wires.
    - Use ST-Link with gostlink. It uses only one USB endpoint so debugging and Trice output in parallel is not possible.
    - Use some other Debug-Probe with target memory access (support welcome)
  - RTT channel selection (on target and on host)
    - RECOMMENDED:
      - `trice l -p JLINK` or shorter `trice l` for STM32F030R8 (default port is JLINK) starts in background a `JLinkRTTLogger.exe` which connects to J-Link and writes to a logfile which in turn is read by the Trice tool. On exit the `JLinkRTTLogger.exe` is killed automatically.
        - It expects a target sending messages over RTT channel **0** (zero). Chapter 16.3.3 in UM08001_JLink.pdf refers to "Up-Channel 1" but this maybe is a typo and probably a 0 is mend. The `JLinkRTTLogger.exe` main advantage against other free available SEGGER tools is, that all bytes are transferred. Other SEGGER tools assume ASCII characters and use `FF 00` to `FF 0F` as a terminal switch command and filter that out causing Trice data disturbances.
        - It should be possible to start several instances on on different targets using `-SelectEmuBySN <SN>` inside the `-args` Trice CLI switch.
        - `JLinkRTTLogger` binaries for Linux & Darwin (MacOS) can be found at https://www.segger.com/downloads/jlink/.
      - `trice l -p STLINK` starts in background a `trice/third_party/goST/stRttLogger.exe` which connects to ST-Link and writes to a logfile which in turn is read by the Trice tool. On exit the `stRttLogger.exe` is killed automatically. It expects a target sending messages over RTT channel 0 (other channels supported too but may not work).

It is possible to start several instances on different channels as well as on different targets. The source code is in https://github.com/bbnote/gostlink and should work also at least under Linux.

- If you have the choice, prefer J-Link. It allows parallel debugging and Trice output.
- The full `-args` string is normally required and depends on the used device. Example: `trice l -args="-Device STM32F070RB -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges 0x20000000_0x1000"`. The `-RTTSearchRanges` part is mostly optional.
- Enter `trice h -log` and read info for `-args` switch:

```
        -args string
        Use to pass port specific parameters. The "default" value depends on the used port:
        port "COMn": default="", use "TARM" for a different driver. (For baud rate settings see -baud.)
        port "J-LINK": default="-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges
0x20000000_0x1000",
                The -RTTSearchRanges "..." need to be written without extra "" and with _ instead of
space.
                For args options see JLinkRTTLogger in SEGGER UM08001_JLink.pdf.
        port "ST-LINK": default="-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges
0x20000000_0x1000",
                The -RTTSearchRanges "..." need to be written without extra "" and with _ instead of
space.
                For args options see JLinkRTTLogger in SEGGER UM08001_JLink.pdf.
        port "BUFFER": default="0 0 0 0", Option for args is any byte sequence.
         (default "default")
```

(back to top)

31.3. J-Link option

- Prerequisite is a SEGGER J-Link debug probe or a development board with an on-board J-Link option.

31.3.1. Convert Evaluation Board onboard ST-Link to J-Link

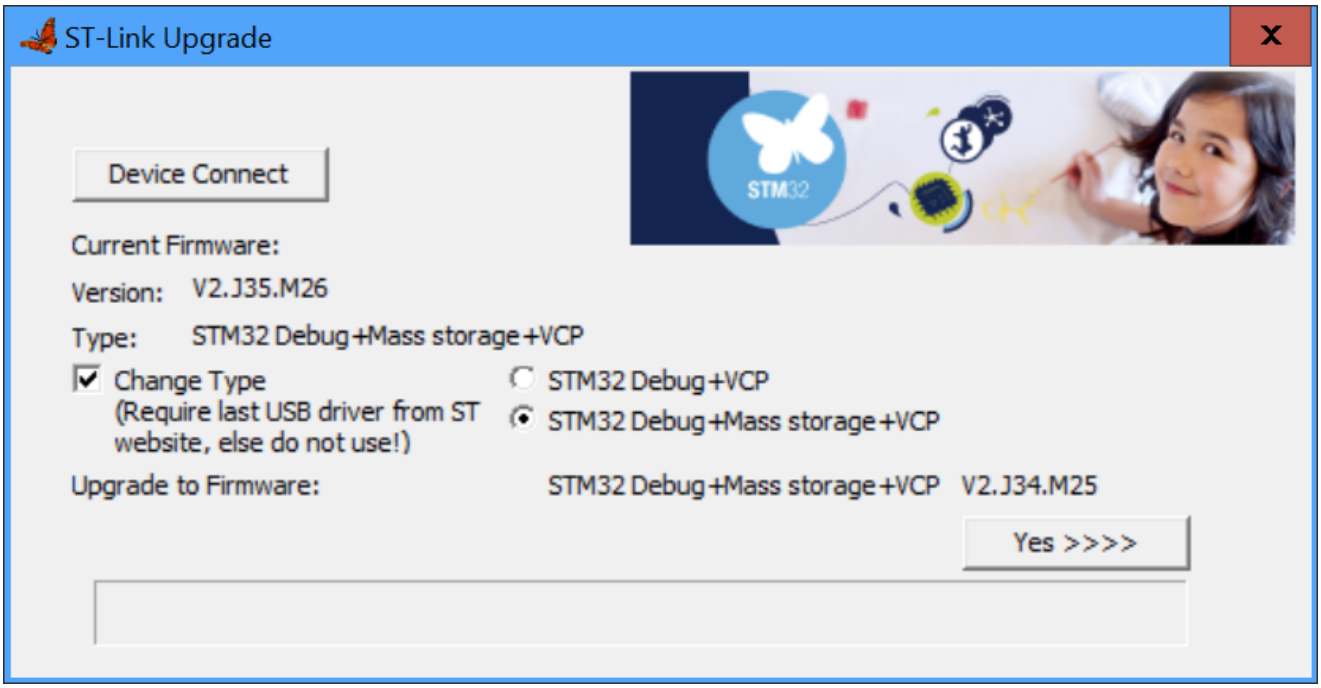
- Following steps describe the needed action for a ST Microelectronics evaluation board and windows - adapt them to your environment.
- It is always possible to turn back to the ST-Link OB firmware with the SEGGER `STLinkReflash.exe` tool but afterwards the ST-Link Upgrade tool should be used again to get the latest version.

First step (to do if some issues occur - otherwise you can skip it)

Video

See also https://github.com/stlink-org/stlink

- Get & install STM32 ST-LINK utility
- Run from default install location `"C:\Program Files (x86)\STMicroelectronics\STM32 ST-LINKUtility\ST-LINK Utility\ST-LinkUpgrade.exe"`)
- Enable checkbox `Change Type` and select radio button `STM32 Debug+Mass storage + VCP`. *The `STM32Debug+ VCP` won´t be detected by Segger reflash utility.*



Second step

- Check Converting ST-LINK On-Board Into a J-Link
- Use `STLinkReflash.exe` to convert NUCLEO from ST-Link on-board to J-Link on-board. *`STM32 Debug+ VCP` won´t be detected by Segger reflash utility.*

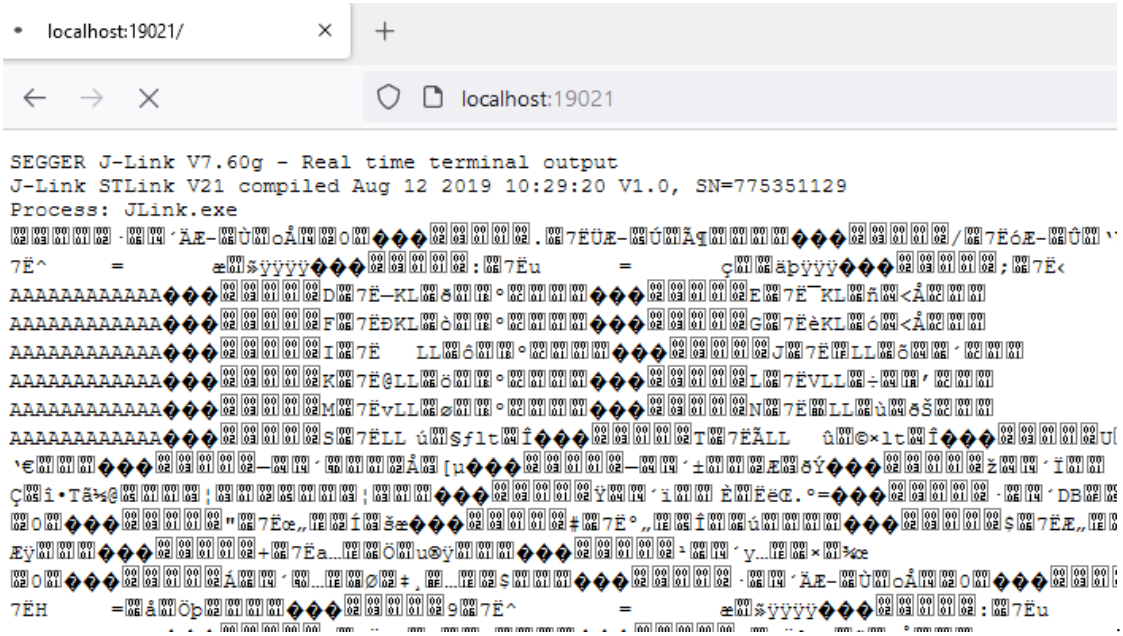
31.3.2. Some SEGGER tools in short

- Download J-Link Software and Documentation Pack and install.
  - You may need to add `C:\Program Files\SEGGER\JLink` to the %PATH% variable.
- Tested with NUCLEO64-STM32F030R8 evaluation board.
- For example: Compile and flash `../examples/F030_inst` project.
  - Check in ../examples/F030_inst/Core/Inc/triceConfig.h if `#define TRICE_RTT_CHANNEL 0` is set as output option.

JLink.exe

- `JLink.exe` is the SEGGER J-Link commander. It starts the **J-Link driver/server** and one can connect to it
- Info found here:
  - J-Link Commander can be started with different command line options for test and automation
  - purposes. In the following, the command line options which are available for J-Link
  - Commander are explained. All command line options are case insensitive.
  - Command Explanation
  - -AutoConnect Automatically start the target connect sequence
  - -CommanderScript Passes a CommandFile to J-Link
  - -CommandFile Passes a CommandFile to J-Link
  - -Device Pre-selects the device J-Link Commander shall connect to
  - -ExitOnError Commander exits after error.
  - -If Pre-selects the target interface
  - -IP Selects IP as host interface
  - -JLinkScriptFile Passes a JLinkScriptFile to J-Link
  - -JTAGConf Sets IRPre and DRPre
  - -Log Sets logfile path
  - -RTTTelnetPort Sets the RTT Telnetport
  - -SelectEmuBySN Connects to a J-Link with a specific S/N over USB
  - -SettingsFile Passes a SettingsFile to J-Link
  - -Speed Starts J-Link Commander with a given initial speed
- Documentation: https://wiki.segger.com/J-Link_Commander
- If you run successful `jlink -device STM32F030R8 -if SWD -speed 4000 -autoconnect 1` the target is stopped.
  - To let in run you need manually execute `go` as command in the open jlink window.

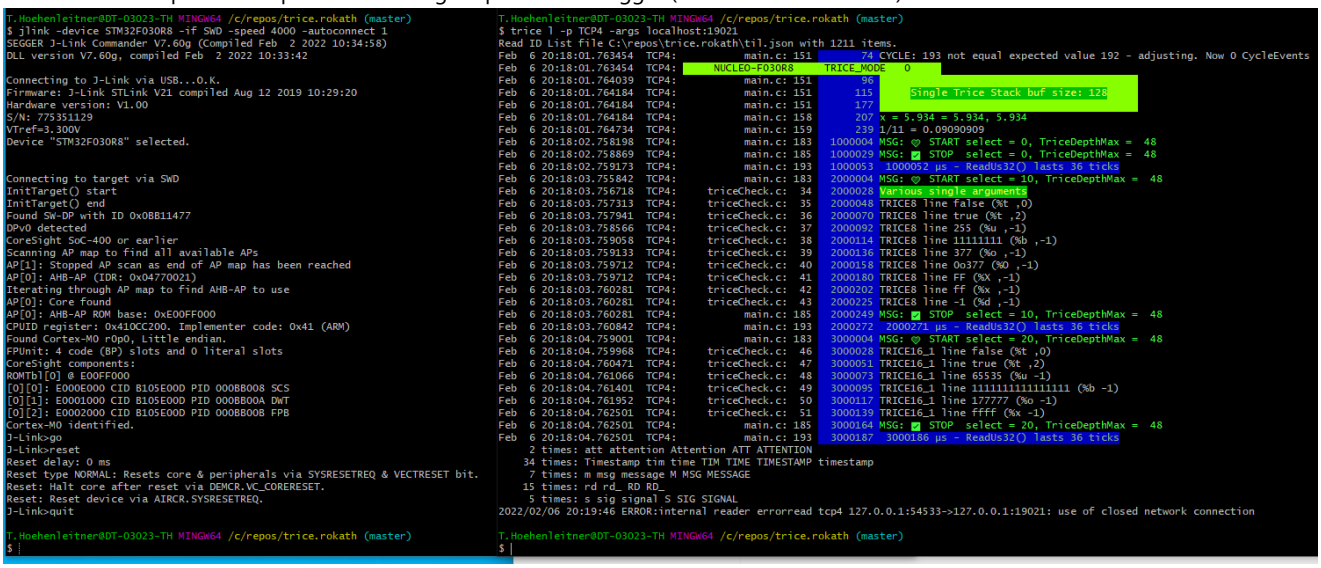
    - To automate that create a text file named for example `jlink.go` containing the `go` command: `echo go > jlink.go` and do a `jlink -device STM32F030R8 -if SWD -speed 4000 -autoconnect 1 -CommandFile jlink.go`
- It is possible to see some output with Firefox (but not with Chrome?) for example:



- After closing the Firefox the Trice tool can connect to it too:
    - Open a commandline and run:

```
trice log -port TCP4 -args localhost:19021
```

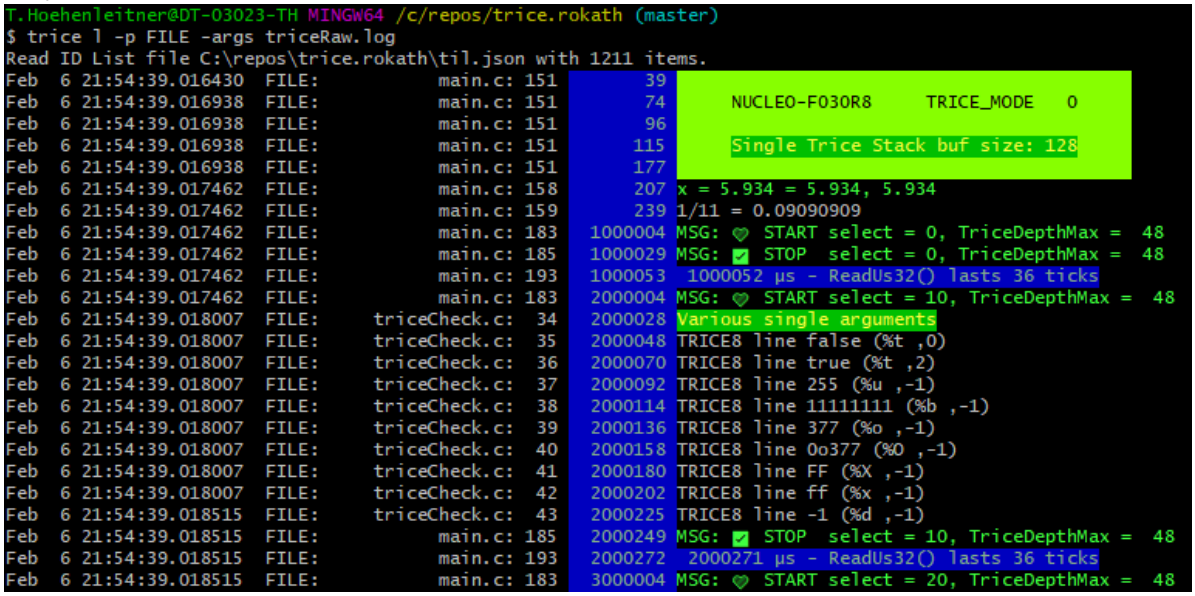
        - Trice output is visible
        - With `h`alt and `g`o inside the Jlink window the MCU can get haltes and released
        - It is possible in parallel to debug-step with a debugger (tested with ARM-MDK)



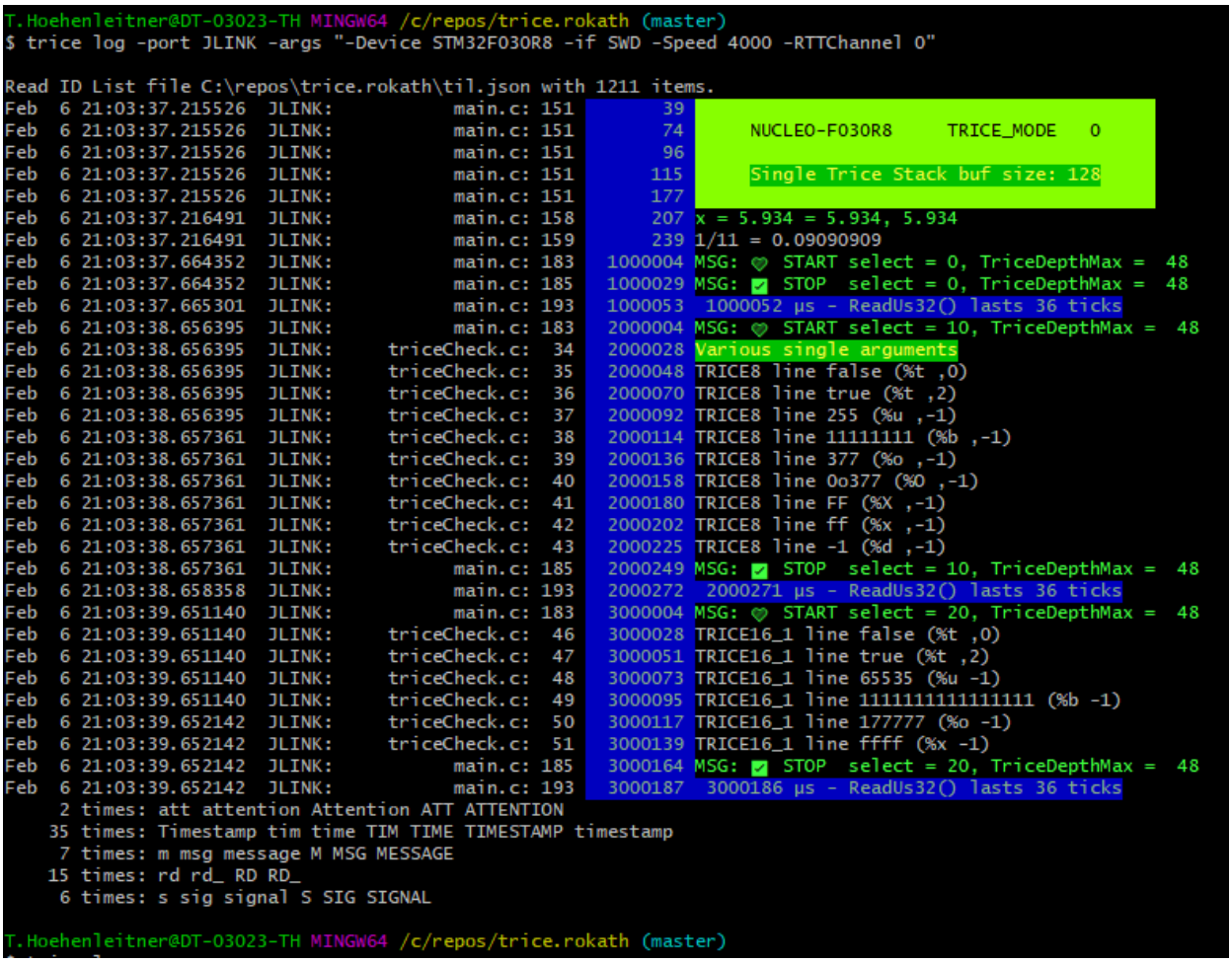
- **PLUS:**
    - Works reliable.
    - No file interface needed.
    - Trice can connect over TCP localhost:19021 and display logs over RTT channel 0.
    - The open `jlink` CLI can be handy to control the target: `[r]eset, [g]o. [h]alt`
    - No need to restart the Trice tool after changed firmware download.
- **MINUS:**
    - Uses RTT up-channel 0 and therefore RTT up-channel 0 is not usable differently.
    - No down-channel usable.
    - Needs a separate manual start of the `jlink` binary with CLI parameters.
        - I would not recommend to automate that too, because this step is needed only once after PC power on.

JLinkRTTLogger.exe

- `JLinkRTTLogger.exe` is a CLI tool and connects via the SEGGER API to the target. It is usable for writing RTT channel 0 data from target into a file.
- **PLUS:**
  - Works reliable.
  - Is automatable.
  - Create file with raw log data: `JLinkRTTLogger.exe -Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 triceRaw.log`
    - It is possible to evaluate this file offline: `trice l -p FILE -args triceRaw.log`



  - No need to restart the Trice tool after changed firmware download.
- **MINUS:**
  - Logs in a file, so the Trice tool needs to read from that file.
  - Maybe cannot write in a file as background process on Darwin (MacOS).
- The Trice tool can watch the output file and display the *Trices*: `trice log -port JLINK -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0"



31.3.3. JLinkRTTClient.exe

- `JLinkRTTClient.exe` can be used for simple text transmitting to the target, it also displays strings from target coming over channel 0. It is not used by the Trice tool.
  - **PLUS:**
    - Target stimulation with proprietary protocol over RTT down-channel 0 possible.
  - **MINUS:**
    - Unfortunately it cannot run separately parallel to stimulate the target with any proprietary protocol because it connects to localhost:19021 and therefore blockades the only one possible connection.

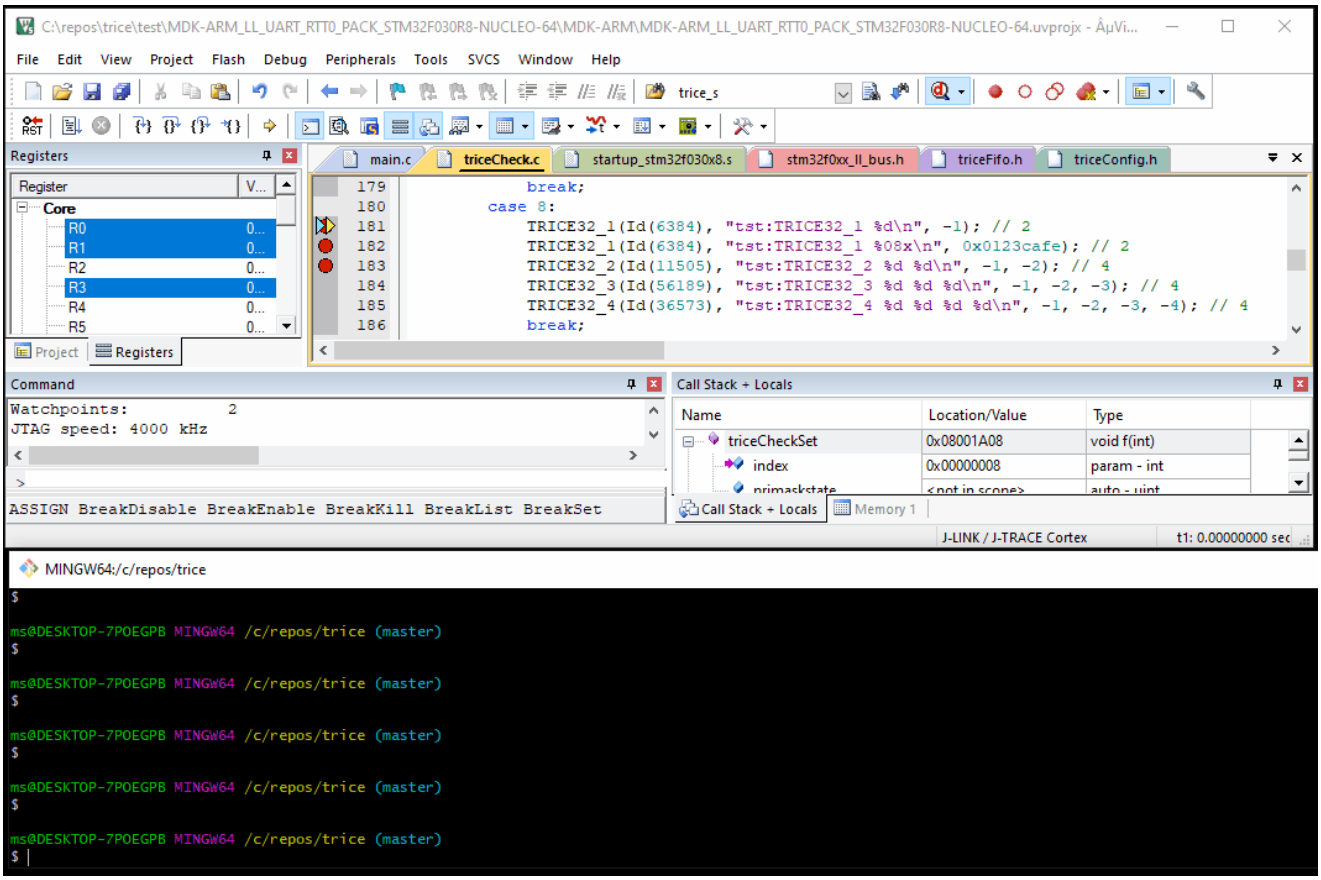
31.3.4. JLinkRTTViewer.exe

- `JLinkRTTViewer.exe` is a GUI tool and connects via the SEGGER API to the target. It expects ASCII codes and is not used by the Trice tool. The switching between the 16 possible terminals is done via `FF 00` ... `FF 0F`. These byte pairs can occur inside the Trice data.

(back to top)

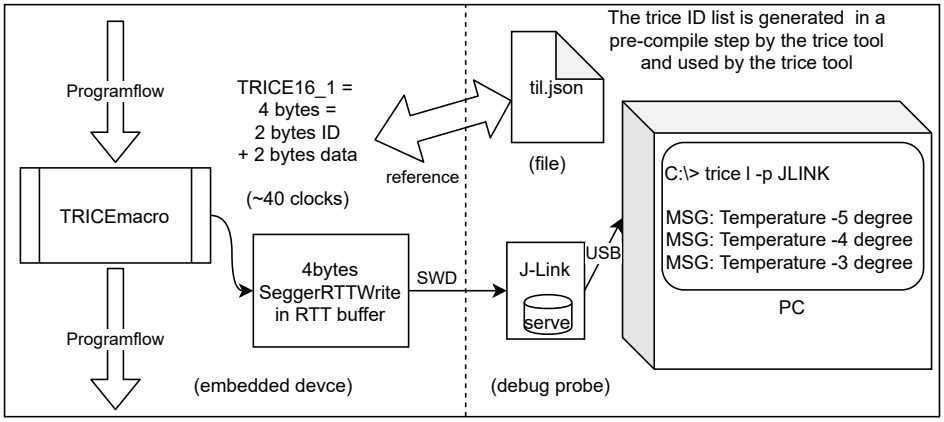
31.4. Segger RTT

- The main advantages are:
  - Speed
  - No `TriceTransfer()` nor any interrupt is needed in the background
  - No UART or other output is needed
- This is, because automatically done by SeggerRTT. This way one can debug code as comfortable as with `printf()` but with all the TRICE advantages. Have a look here:



- Avoid Trice buffering inside target and write with TRICE macro directly into the RTT buffer (direct Trice mode = `#define TRICE_MODE 0` inside triceConfig.h).

- Write the bytes per Trice directly (little time & some space overhead on target, but no changes on host side)



(back to top)

31.5. Segger J-Link SDK (~800 EUR) Option

- Segger offers a SeggerRTT SDK which allows to use more than just channel 0 and you can develop your own tooling with it.
- The `trice -port JLINK` is ok for usage **as is** right now. However if you wish more comfort check here:
- Question: How-to-access-multiple-RTT-channels
  - "Developer pack used to write your own program for the J-Link. Please be sure you agree to the terms of the associated license found on the Licensing Information tab before purchasing this SDK. You will benefit from six months of free email support from the time that this product is ordered."
- The main Segger J-Link SDK disadvantage beside closed source and payment is: **One is not allowed to distribute binaries written with the SDK.** That makes it only interesting for company internal automatization.

(back to top)

31.6. Additional Notes (leftovers)

- `Downloading RTT target package` from https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/.
- Read the manual UM08001_JLink.pdf.
- Extract `../third_party/segger.com/SEGGER_RTT_V760g.zip` to `../third_party/segger.com/SEGGER_RTT`. Check for an update @ SEGGER.
- Add `SEGGER_RTTI.c` to target project

(back to top)

31.7. Further development

- Check OpenOCD!
  - Use OpenOCD and its built-in RTT feature. OpenOCD then starts a server on localhost:17001 where it dumps all RTT messages.
- The GoST project offers a way bypassing JLINK. Used -port STLINK instead.
- Maybe `libusb` together with `libjaylink` offer some options too.
- Checkout https://github.com/deadsy/jaylink.
- `"C:\Program Files (x86)\SEGGER\JLink\JMem.exe"` shows a memory dump.
- Go to https://libusb.info/
  - -> Downloads -> Latest Windows Binaries
  - extract `libusb-1.0.23` (or later version)

```
libusb-1.0.23\examples\bin64> .\listdevs.exe
2109:2811 (bus 2, device 8) path: 6
1022:145f (bus 1, device 0)
1022:43d5 (bus 2, device 0)
0a12:0001 (bus 2, device 1) path: 13
1366:0105 (bus 2, device 10) path: 5
```

- Repeat without connected Segger JLink

```
libusb-1.0.23\examples\bin64> .\listdevs.exe
2109:2811 (bus 2, device 8) path: 6
1022:145f (bus 1, device 0)
1022:43d5 (bus 2, device 0)
0a12:0001 (bus 2, device 1) path: 13
```

- In this case `1366:0105 (bus 2, device 10) path: 5` is missing, so `vid=1366`, `did=0105` as example
- On Windows install WSL2. The real Linux kernel is needed for full USB access.

(back to top)

31.8. NUCLEO-F030R8 example

Info: https://www.st.com/en/evaluation-tools/nucleo-F030r8.html

31.8.1. RTT with original on-board ST-LINK firmware

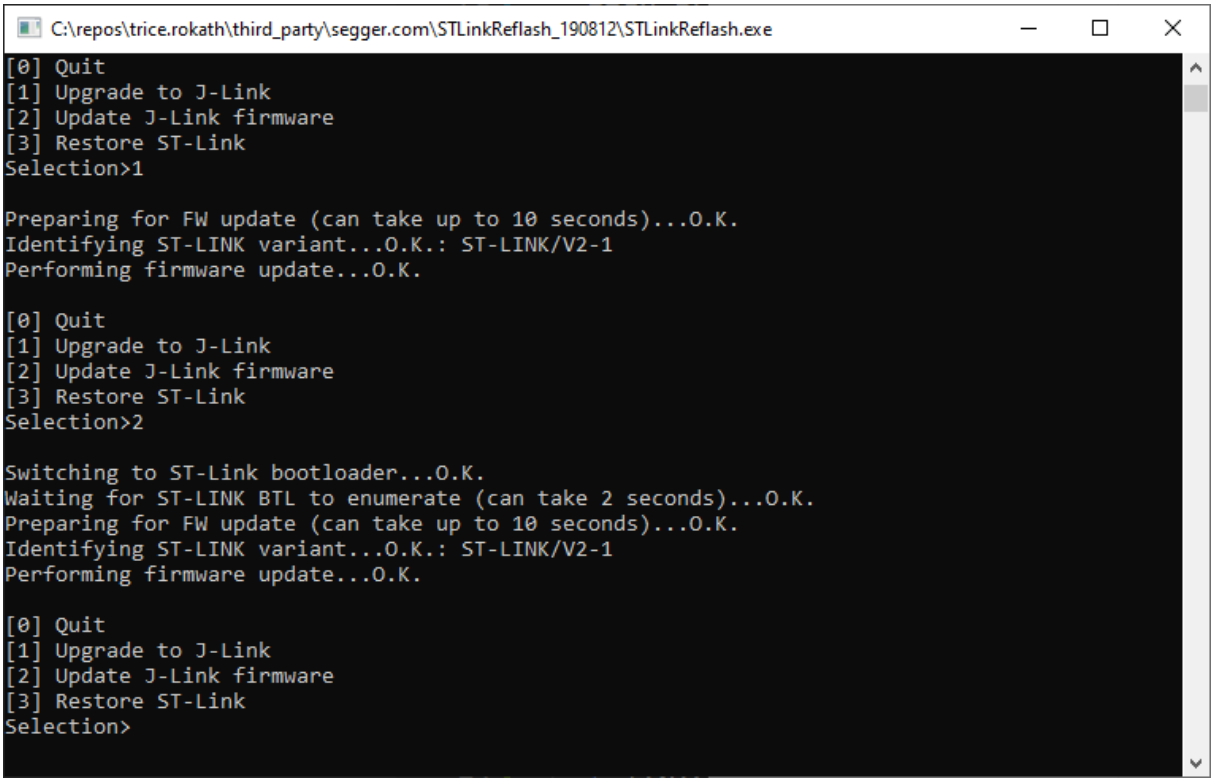
- `#define TRICE_RTT_CHANNEL 0`:
- If you use a NUCLEO-F030R8 with the original ST-Link on board after firmware download enter: `trice l -p ST-LINK -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges 0x20000000_0x2000"`. After pressing the reset

button output becomes visible:

```
T.Hoehenleitner@DT-03023-TH MINGW64 /c/repos/trice.rokath (master)
$ trice l -p ST-LINK -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges 0x20000000_0x2000"
Read ID List file C:\repos\trice.rokath\til.json with 1192 items.
Jan 23 00:56:31.883693  ST-LINK:          main.c: 151       40
Jan 23 00:56:31.883693  ST-LINK:          main.c: 151       42      NUCLEO-F030R8      TRICE_MODE 200
Jan 23 00:56:31.883693  ST-LINK:          main.c: 151       44
Jan 23 00:56:31.883693  ST-LINK:          main.c: 151       46      Trice 2x half buffer size:1000
Jan 23 00:56:31.883693  ST-LINK:          main.c: 151       51
Jan 23 00:56:31.883693  ST-LINK:          main.c: 158       67 x = 5.934 = 5.934, 5.934
Jan 23 00:56:31.883693  ST-LINK:          main.c: 159       70 1/11 = 0.09090909
Jan 23 00:56:32.864984  ST-LINK:          main.c: 183  1000007 MSG: ♡ START select = 0, TriceDepthMax = 152
Jan 23 00:56:32.865949  ST-LINK:          main.c: 185  1000012 MSG: ☑ STOP  select = 0, TriceDepthMax = 152
Jan 23 00:56:32.865949  ST-LINK:          main.c: 193  1000016  1000015 µs - ReadUs32() lasts 36 ticks
Jan 23 00:56:33.910727  ST-LINK:          main.c: 183  2000007 MSG: ♡ START select = 10, TriceDepthMax = 152
Jan 23 00:56:33.910727  ST-LINK:     triceCheck.c:  33  2000012 Various single arguments
Jan 23 00:56:33.910727  ST-LINK:     triceCheck.c:  34  2000014 TRICE8 line false (%t ,0)
Jan 23 00:56:33.910727  ST-LINK:     triceCheck.c:  35  2000016 TRICE8 line true (%t ,2)
Jan 23 00:56:33.910727  ST-LINK:     triceCheck.c:  36  2000017 TRICE8 line 255 (%u ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  37  2000019 TRICE8 line 11111111 (%b ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  38  2000021 TRICE8 line 377 (%o ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  39  2000023 TRICE8 line 0o377 (%O ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  40  2000025 TRICE8 line FF (%X ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  41  2000027 TRICE8 line ff (%x ,-1)
Jan 23 00:56:33.911729  ST-LINK:     triceCheck.c:  42  2000029 TRICE8 line -1 (%d ,-1)
Jan 23 00:56:33.911729  ST-LINK:          main.c: 185  2000031 MSG: ☑ STOP  select = 10, TriceDepthMax = 196
```

- It works with both ST-Link variants (with or without mass storage device.)

31.8.2. Change to J-LINK onboard firmware



31.8.3. RTT with J-LINK firmware on-board

```
T.Hoehenleitner@DT-03023-TH MINGW64 /c/repos/trice.rokath (master)
$ trice l -p J-LINK -args "-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -RTTSearchRanges 0x20000000_0x2000"
Read ID List file C:\repos\trice.rokath\til.json with 1192 items.
Jan 23 01:09:47.170489  J-LINK:          main.c: 151       40
Jan 23 01:09:47.170489  J-LINK:          main.c: 151       42      NUCLEO-F030R8      TRICE_MODE 200
Jan 23 01:09:47.170489  J-LINK:          main.c: 151       44
Jan 23 01:09:47.170489  J-LINK:          main.c: 151       46      Trice 2x half buffer size:1000
Jan 23 01:09:47.170489  J-LINK:          main.c: 151       51
Jan 23 01:09:47.170489  J-LINK:          main.c: 158       67 x = 5.934 = 5.934, 5.934
Jan 23 01:09:47.170489  J-LINK:          main.c: 159       70 1/11 = 0.09090909
Jan 23 01:09:47.726793  J-LINK:          main.c: 183  1000007 MSG: ♡ START select = 0, TriceDepthMax = 152
Jan 23 01:09:47.726793  J-LINK:          main.c: 185  1000012 MSG: ☑ STOP  select = 0, TriceDepthMax = 152
Jan 23 01:09:47.726793  J-LINK:          main.c: 193  1000016  1000015 µs - ReadUs32() lasts 36 ticks
Jan 23 01:09:48.725748  J-LINK:          main.c: 183  2000007 MSG: ♡ START select = 10, TriceDepthMax = 152
Jan 23 01:09:48.725748  J-LINK:     triceCheck.c:  33  2000012 Various single arguments
Jan 23 01:09:48.725748  J-LINK:     triceCheck.c:  34  2000014 TRICE8 line false (%t ,0)
Jan 23 01:09:48.725748  J-LINK:     triceCheck.c:  35  2000016 TRICE8 line true (%t ,2)
```

- Observations:
  - When pressing the black reset button, you need to restart the Trice tool.
  - When restarting the Trice tool, a target reset occurs.
  - Other channel numbers than `0` seam not to work for some reason.

(back to top)

31.9. Possible issues

- These boards seem not to work reliable with RTT over J-Link on-board firmware.
  - NUCLEO-G071RB
  - NUCLEO_G031K8
- After flashing back the ST-LINK OB firmware with the SEGGER tool, it is recommended to use the ST tool to update the ST-LINK OB firmware. Otherwise issues could occur.

(back to top)

31.10. OpenOCD with Darwin (MacOS)

- OpenOCD on MacOS works out of the box after installing it.
- When using VS code with Cortex-Debug you cannot use OpenOCD at the same time.
- The `openocd.cfg` file is taylored to the flashed on-board J-Link adapter.

Terminal 1:

```
brew install open-ocd
...
cd ./trice/examples/G0B1_inst
openocd -f openocd.cfg
Open On-Chip Debugger 0.12.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
srst_only separate srst_nogate srst_open_drain connect_deassert_srst

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link STLink V21 compiled Aug 12 2019 10:29:20
Info : Hardware version: 1.00
Info : VTarget = 3.300 V
Info : clock speed 2000 kHz
Info : SWD DPIDR 0x0bc11477
Info : [stm32g0x.cpu] Cortex-M0+ r0p1 processor detected
Info : [stm32g0x.cpu] target has 4 breakpoints, 2 watchpoints
Info : starting gdb server for stm32g0x.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : rtt: Searching for control block 'SEGGER RTT'
Info : rtt: Control block found at 0x20001238
Info : Listening on port 9090 for rtt connections
Channels: up=1, down=0
Up-channels:
0: Terminal 1024 0
Down-channels:

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

Terminal 2:

```
ms@MacBook-Pro G0B1_inst % trice l -p TCP4 -args localhost:9090  -pf none -d16
Nov 14 17:32:33.319451  TCP4:       triceExamples.c    10        0_000  Hello! 👌😀
Nov 14 17:32:33.319463  TCP4:
Nov 14 17:32:33.319463  TCP4:         ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
```

```
Nov 14 17:32:33.319463  TCP4:          🎈🎈🎈🎈  NUCLEO-G0B1RE    🎈🎈🎈🎈
Nov 14 17:32:33.319463  TCP4:          🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀🍀
Nov 14 17:32:33.319463  TCP4:
Nov 14 17:32:33.319463  TCP4:
Nov 14 17:32:33.406455  TCP4:       triceExamples.c    16        0_037 2.71828182845904523536 <- float
number as string
Nov 14 17:32:33.505116  TCP4:       triceExamples.c    17        0_087 2.71828182845904509080 (double with
more ciphers than precision)
Nov 14 17:32:33.607518  TCP4:       triceExamples.c    18        0_117 2.71828174591064453125 (float  with
more ciphers than precision)
Nov 14 17:32:33.707851  TCP4:       triceExamples.c    19        0_146 2.718282 (default rounded float)
Nov 14 17:32:33.807685  TCP4:       triceExamples.c    20        0_175 A Buffer:
Nov 14 17:32:33.908202  TCP4:       triceExamples.c    21        0_204 32 2e 37 31 38 32 38 31 38 32 38 34
35 39 30 34 35 32 33 35 33 36
Nov 14 17:32:34.007148  TCP4:       triceExamples.c    22        0_254 31372e32  31383238  34383238
34303935  35333235
Nov 14 17:32:35.007949  TCP4:       triceExamples.c    23        0_301 ARemoteFunctionName(2e32)(3137)
(3238)(3138)(3238)(3438)(3935)(3430)(3235)(3533)(3633)
Nov 14 17:32:35.112304  TCP4:       triceExamples.c    24              100 times a 16 byte long Trice
messages, which not all will be written because of the TRICE_PROTECT:
Nov 14 17:32:35.307567  TCP4:       triceExamples.c    26        0_379 i=44444400 aaaaaa00
Nov 14 17:32:35.408257  TCP4:       triceExamples.c    27    0,000_002 i=44444400 aaaaaa00
Nov 14 17:32:35.509022  TCP4:       triceExamples.c    26        0_441 i=44444401 aaaaaa01
Nov 14 17:32:35.609439  TCP4:       triceExamples.c    27    0,000_002 i=44444401 aaaaaa01
Nov 14 17:32:35.710201  TCP4:       triceExamples.c    26        0_504 i=44444402 aaaaaa02
...
```

31.11. SEGGER J-Link on Darwin (MacOS)

TODO: Working example with SEGGER_RTT J-Link and Open OCD

31.12. Links

- USB over WSL2? (Maybe intersting for OpenOCD)
- https://kickstartembedded.com/2024/03/26/openocd-one-software-to-rule-debug-them-all/?amp=1
- https://mcuoneclipse.com/2021/10/03/visual-studio-code-for-c-c-with-arm-cortex-m-part-9-rtt/

(back to top)

32. Writing the Trice logs into an SD-card (or a user specific output)

- Enable `TRICE_DEFERRED_AUXILIARY8` in your project specific *triceConfig.h* file.
- Enabling `TRICE_DEFERRED_AUXILIARY8` is possible parallel to any direct and/or deferred output.
- The `TRICE_DEFERRED_OUT_FRAMING` value is used also for the deferred auxiliary writes.
- Consider the value for `TRICE_DEFERRED_TRANSFER_MODE`. The `TRICE_SINGLE_PACK_MODE` would trigger a file write function on each single Trice message.
- Provide a self-made function like this:

```
/// mySDcardWrite performs SD-card writing by appending to file myTriceLogs.bin.
mySDcardWrite(const uint8_t* buf, size_t bufLen){
    ...
}

/// Assign this function pointer accordingly.
UserNonBlockingDeferredWrite8AuxiliaryFn = mySDcardWrite;
```

- If the SD-card write is more effective using 32-bits chunks, consider `TRICE_DEFERRED_AUXILIARY32`, what is recommended also if you use the encryption option.
    - `TRICE_DEFERRED_AUXILIARY32` is diabled / not implemented yet. It will be added on demand.

- There maybe use cases for `TRICE_DIRECT_AUXILIARY8` or `TRICE_DIRECT_AUXILIARY32`, but consider the max write time.
- Placing the files *til.json* and *li.json* anto the SD-card as well might be meaninjful.
- To decode *myTriceLogs.bin* later

```
trice log -port FILEBUFFER -args myTriceLogs.bin -hs off
```

Related issues/discussions: #253 #405 #425 #447 #537

(back to top)

33. Trice Target Code Implementation

33.1. TRICE Macro structure

33.1.1. TRICE_ENTER

- Optionally disable interrupts.
- Prepare `TriceBufferWritePosition` and keep its initial value.

33.1.2. TRICE_PUT

- Use and increment `TriceBufferWritePosition`.

33.1.3. TRICE_LEAVE

- Use `TriceBufferWritePosition` and its initial value for data transfer
- Optionally restore interrupt state.

33.2. TRICE_STACK_BUFFER

- `TRICE_ENTER`: Allocate stack
- `TRICE_LEAVE`: Call TriceDirectOut()

33.3. TRICE_STATIC_BUFFER

- This is like `TRICE_STACK_BUFFER` but avoids stack allocation, what is better for many stacks.
- `TRICE_ENTER`: Set TriceBufferWritePosition to buffer start.
- `TRICE_LEAVE`: Call TriceDirectOut().

33.4. TRICE_DOUBLE_BUFFER

- `TRICE_ENTER`: Keep TriceBufferWritePosition.
- `TRICE_LEAVE`: Optionally call TriceDirectOut().

33.5. TRICE_RING_BUFFER

- `TRICE_ENTER`: Keep or wrap TriceBufferWritePosition and add offset.
- `TRICE_LEAVE`: Optionally call TriceDirectOut().

The `TRICE_RING_BUFFER` allocates incremental ring buffer space and each trice location is read by a deferred task.

33.6. Deferred Out

33.6.1. Double Buffer

- TriceTransfer
  - TriceOut
  - TriceNonBlockingWrite(triceID, enc, encLen);

33.6.2. Ring Buffer

- TriceTransfer
  - lastWordCount = TriceSingleDeferredOut(addr);
    - int triceID = TriceIDAndBuffer(pData, &wordCount, &pStart, &Length);
    - TriceNonBlockingWrite(triceID, pEnc, encLen);

33.7. Direct Transfer

- TRICE_LEAVE
  - TriceDirectWrite(triceSingleBufferStartWritePosition, wordCount);
    - optional RTT32 with optional XTEAwithCOBS
    - optional RTT8 with optional XTEAwithCOBS
    - optional
      - triceIDAndLen
      - triceDirectEncode
      - triceNonBlockingDirectWrite

33.8. Possible Target Code Improvements

There have been 3 similar implementations for trice encode

```
static size_t triceDirectEncode(   uint8_t* enc, const uint8_t * buf, size_t len );
       size_t TriceDeferredEncode( uint8_t* enc, const uint8_t * buf, size_t len );

unsigned TriceEncryptAndCobsFraming32( uint32_t * const triceStart, unsigned wordCount ){
```

Now:

```
size_t TriceEncode( unsigned encrypt, unsigned framing, uint8_t* dst, const uint8_t * buf, size_t len ){
unsigned TriceEncryptAndCobsFraming32( uint32_t * const triceStart, unsigned wordCount ){
```

Currently there are 3 similar implementations for trice buffer reads

```
static size_t triceIDAndLen(    uint32_t* pBuf,               uint8_t** ppStart, int*      triceID );
static int    TriceNext(        uint8_t** buf,                size_t* pSize,     uint8_t** pStart,
size_t* pLen );
static int    TriceIDAndBuffer( uint32_t const * const pData, int* pWordCount,   uint8_t** ppStart,
size_t* pLength );
```

- The TriceID is only needed for routing and can go in a global variable just for speed.
- The source buffer should be `uint32_t const * const`.
- The destination should be given with `uint32_t * const` and the return value is the trice netto size. For efficiency the result should be ready encoded.

```
//! \param pTriceID is filled with ID for routing
//! \param pCount is used for double or ring buffer to advance inside the buffer
//! \param dest provides space for the encoded trice
//! \param src is the location of the trice message we want encode
//! \retval is the netto size of the encoded trice data
size_t TriceEncode(int* pTriceID, unsigned int pCount, uint32_t * const dest, uint32_t const * const src
);
```

- This function interface is used for all cases.
- First we use the existing code for implementation and then we clean the code.

(back to top)

34. Trice Similarities and Differences to printf Usage

34.1. Printf-like functions

...have a lot of things to do: Copy format string from FLASH memory into a RAM buffer and parse it for format specifiers. Also parse the variadic parameter list and convert each parameter according to its format specifier into a character sequences, what includes several divisions - costly function calls. Concatenate the parts to a new string and deliver it to the output, what often means copying again. A full-featured printf library consumes plenty space and processing time and several open source projects try to make it better in this or that way. Never ever call a printf-like function in time critical code, like an interrupt - it would crash your target in most cases. The *trice* calls are usable inside interrupts, because they only need a few MCU clocks for execution. Porting legacy code to use it with the Trice library, means mainly to replace Printf-like function calls with `trice` function calls. See also chapter Legacy User Code Option Print Buffer Wrapping and Framing.

34.2. Trice IDs

- Each Trice caries a 14-bit nuber ID as replacement for the format string.
- This ID is automatically generated (controllable) and in the source code it is the first parameter inside the Trice macro followed by the format string and optional values.
- The user can decide not to spoil the code by having the IDs permanently in its source code, by just inserting them as a pre-compile step with `trice insert` and removing them as a post-compile step with `trice clean`.
  - The Trice cache makes this invisible to the build system, allowing full translation speed.
- The format string is **not** compiled into the target code. It goes together with the ID into a project specific reference list file til.json (example).

34.3. Trice values bit width

- No need to explicit express the value bit width.
- The default parameter width for the Trice macro is 32 bit. It is changeable to 8, 16 or 64-bit:
  - Adapt `TRICE_DEFAULT_PARAMETER_BIT_WIDTH` inside `triceConfig.h`. It influences

```
238 ////////////////////////////////////////////////////////////////////////////
239 // Default TRICE macro bitwidth: 32 (optionally adapt to MCU bit width)
240 //
241
242 #define TRICE_1  TRICE32_1  //!< Default parameter bit width for 1  parameter count TRICE is 32, change for a different value.
243 #define TRICE_2  TRICE32_2  //!< Default parameter bit width for 2  parameter count TRICE is 32, change for a different value.
244 #define TRICE_3  TRICE32_3  //!< Default parameter bit width for 3  parameter count TRICE is 32, change for a different value.
245 #define TRICE_4  TRICE32_4  //!< Default parameter bit width for 4  parameter count TRICE is 32, change for a different value.
246 #define TRICE_5  TRICE32_5  //!< Default parameter bit width for 5  parameter count TRICE is 32, change for a different value.
247 #define TRICE_6  TRICE32_6  //!< Default parameter bit width for 6  parameter count TRICE is 32, change for a different value.
248 #define TRICE_7  TRICE32_7  //!< Default parameter bit width for 7  parameter count TRICE is 32, change for a different value.
249 #define TRICE_8  TRICE32_8  //!< Default parameter bit width for 8  parameter count TRICE is 32, change for a different value.
250 #define TRICE_9  TRICE32_9  //!< Default parameter bit width for 9  parameter count TRICE is 32, change for a different value.
251 #define TRICE_10 TRICE32_10 //!< Default parameter bit width for 10 parameter count TRICE is 32, change for a different value.
252 #define TRICE_11 TRICE32_11 //!< Default parameter bit width for 11 parameter count TRICE is 32, change for a different value.
253 #define TRICE_12 TRICE32_12 //!< Default parameter bit width for 12 parameter count TRICE is 32, change for a different value.
254
255 //
256 ////////////////////////////////////////////////////////////////////////////
```

  - Use `-defaultTRICEBitwidth` switch during logging when changing this value.
- The macros `trice8`, `trice16`, `trice32`, `trice64` are usable too, to define the bit width explicit.
  - This leads for the smaller bit widths to less needed space and bandwidth. But when using the default package framing TCOBS, the influence is marginal because of the implicit compression.
- The fastest Trice macro execution is, when MCU bit width matches the macro bit width.
- The implicit TCOBS compression compacts the binary Trice data during the framing.

34.4. Many value parameters

- No need to explicit express the values count.
- Up to 12 values are supported directly. Example:
  - `trice( "%p | %04x %04x %04x %04x %04x %04x %04x %04x %04x %04x | %f\n", p, p[0], p[1], p[2], p[3], p[4], p[5], p[6], p[7], p[8], p[9], aFloat(x));`
  - To support more than 12 values for each Trice macro, the Trice code on target and host is straightforward extendable up to a total payload of 32764 bytes.
- Each macro can be prolonged with the used parameter count, for example `TRICE8_3` or `TRICE_2` to intense compile time checks.
  - This length code extension can be done automatically using `trice u -addParamCount`. This is not needed anymore:
- The *Trice* tool compares the number of given format specifiers with the written parameters in a precimpile step to minimize the risk of runtime errors.
- There is no variadic values scanning during runtime. The C preprocessor does the work.

34.5. Floating Point Values

These types are mixable with integer types but need to be covered by converter function.

- *float* types use the `aFloat()` function and need a minimal value bit width of 32, to secure correct data transfer.
    - Example:

```
float x = 7.2;
trice( "%f", aFloat(x));
```

- *double* types use the `aDouble()` function and need a value bit width of 64, to secure correct data transfer.
    - Example:

```
double y = 7.2;
trice64( "float %f and double %f", aFloat(x), aDouble(y));
```

- Both functions are simple and fast:

```
// aFloat returns passed float value x as bit pattern in a uint32_t type.
static inline uint32_t aFloat( float x ){
    union {
        float f;
        uint32_t u;
    } t;
    t.f = x;
    return t.u;
}

// aDouble returns passed double value x as bit pattern in a uint64_t type.
static inline uint64_t aDouble( double x ){
    union {
        double d;
        uint64_t u;
    } t;
    t.d = x;
    return t.u;
}
```

34.6. Runtime Generated 0-terminated Strings Transfer with triceS

- The `%s` format specifier is supported by the Trice macro too but needs specific treatment.
- Strings, known at compile time should be a part of a format string to reduce runtime overhead.
- Strings created at runtime, need a special `TRICE_S` (or `triceS`, `TriceS`, `TRiceS`) macro, which accepts exactly one type `%s` format specifier. Generated strings are allowed to a size of 32764 bytes each, if the configured Trice buffer size is sufficient.
    - Example:

```
char s[] = "Hello again!";
triceS("A runtime string %20s\n", s);
```

34.7. Runtime Generated counted Strings Transfer with triceN

- It is also possible to transfer a buffer with length n using the `TRICE_N` (or `triceN`, `TriceN`, `TRiceN`) macro.

- This becomes handy for example, when a possibly not 0-terminated string in FLASH memory needs transmission: `triceN( "msg: FLASH string is %s", addr, 16 );`
- There are also specific macros like `trice32B` or `trice16F`. Please look into triceCheck.c for usage or see the following.

34.8. Runtime Generated Buffer Transfer with triceB

- A buffer is transmittable with `TRICE_B` (or `triceB`, `TriceB`, `TRiceB`) and specifying just one format specifier, which is then repeated. Example:

```
s = "abcde 12345"; // assume this as runtime generated string
triceS( "msg:Show s with triceS: %s\n", s );
len = strlen(s);
triceN( "sig:Show s with triceN:%s\n", s, len );
triceB( "dbg: %02x\n", s, len ); // Show s as colored code sequence in hex code.
triceB( "msg: %4d\n", s, len ); // Show s as colored code sequence in decimal code.
```

This gives output similar to:

```
triceCheck.c:  90   1000010 With TRICE_S:abcde 12345
triceCheck.c:  92   1000016 With TRICE_N:abcde 12345
triceCheck.c:  93   1000020 len=11:With TRICE_B:
triceCheck.c:  94   1000022   61  62  63  64  65  20  31  32  33  34  35
triceCheck.c:  96   1000027   97  98  99 100 101  32  49  50  51  52  53
```

Channel specifier within the `TRICE_B` format string are supported in Trice versions >= v0.66.0.

If the buffer is not 8 but 16, 32 or 32 bits wide, the macros `TRICE8_B`, `TRICE16_B`, `TRICE32_B` and `TRICE64_B`, are usable in the same manner.

34.9. Remote function call syntax support with triceF

The `TRICE8_F`, `TRICE16_F`, `TRICE32_F`, `TRICE64_F`, macros expect a string without format specifiers which is usable later as a function call. Examples:

```
trice8F(   "call:FunctionNameW", b8,  sizeof(b8) /sizeof(int8_t) );   //exp: time:            default:
call:FunctionNameW(00)(ff)(fe)(33)(04)(05)(06)(07)(08)(09)(0a)(0b)(00)(ff)(fe)(33)(04)(05)(06)(07)(08)(09)
(0a)(0b)
TRICE16_F( "info:FunctionNameX", b16, sizeof(b16)/sizeof(int16_t) );  //exp: time: 842,150_450default:
info:FunctionNameX(0000)(ffff)(fffe)(3344)
TRice16F(  "call:FunctionNameX", b16, sizeof(b16)/sizeof(int16_t) );  //exp: time: 842,150_450default:
call:FunctionNameX(0000)(ffff)(fffe)(3344)
Trice16F(  "call:FunctionNameX", b16, sizeof(b16)/sizeof(int16_t) );  //exp: time:       5_654default:
call:FunctionNameX(0000)(ffff)(fffe)(3344)
trice16F(  "call:FunctionNameX", b16, sizeof(b16)/sizeof(int16_t) );  //exp: time:            default:
call:FunctionNameX(0000)(ffff)(fffe)(3344)
TRICE32_F( "info:FunctionNameY", b32, sizeof(b32)/sizeof(int32_t) );  //exp: time: 842,150_450default:
info:FunctionNameY(00000000)(ffffffff)(fffffffe)(33445555)
TRice32F(  "call:FunctionNameY", b32, sizeof(b32)/sizeof(int32_t) );  //exp: time: 842,150_450default:
call:FunctionNameY(00000000)(ffffffff)(fffffffe)(33445555)
Trice32F(  "call:FunctionNameY", b32, sizeof(b32)/sizeof(int32_t) );  //exp: time:       5_654default:
call:FunctionNameY(00000000)(ffffffff)(fffffffe)(33445555)
trice32F(  "call:FunctionNameY", b32, sizeof(b32)/sizeof(int32_t) );  //exp: time:            default:
call:FunctionNameY(00000000)(ffffffff)(fffffffe)(33445555)
TRICE64_F( "info:FunctionNameZ", b64, sizeof(b64)/sizeof(int64_t) );  //exp: time: 842,150_450default:
info:FunctionNameZ(0000000000000000)(ffffffffffffffff)(fffffffffffffffe)(3344555566666666)
TRice64F(  "call:FunctionNameZ", b64, sizeof(b64)/sizeof(int64_t) );  //exp: time: 842,150_450default:
call:FunctionNameZ(0000000000000000)(ffffffffffffffff)(fffffffffffffffe)(3344555566666666)
Trice64F(  "call:FunctionNameZ", b64, sizeof(b64)/sizeof(int64_t) );  //exp: time:       5_654default:
call:FunctionNameZ(0000000000000000)(ffffffffffffffff)(fffffffffffffffe)(3344555566666666)
trice64F(  "call:FunctionNameZ", b64, sizeof(b64)/sizeof(int64_t) );  //exp: time:            default:
call:FunctionNameZ(0000000000000000)(ffffffffffffffff)(fffffffffffffffe)(3344555566666666)
```

The Trice tool displays the parameter buffer in the shown manner. There is a FunctionPointerList Generator, which generates mainly a function pointer list with associated IDs. This list can get part of the source code of a remote device. Then, when receiving a Trice message,

the remote device can execute the assigned function call using the transferred parameters. This way several devices can communicate in an easy and reliable way.

With `#define TRICE_F TRICE16_F` in the project specific *triceConfig.h* file the user can specify which should be the bitwidth (16 in this example) for `triceF` macros. The default value is 8.

Hint: If you add for example `"rpc"` as tag and call `trice log -ban "rpc"`, the Trice tool will not display the RPC Trices, but all others. That could be helpful, if you have frequent RPCs and do not wish to spoil your log output with them.

- Future extensions are possible:
  - `triceD( "dump:32", addr, 160 );` -> The Trice tool dumps in 32 byte rows.
  - An appropriate syntax is needed.

34.10. Extended format specifier possibilities

- Because the format string is interpreted by the Trice tool written in Go, the **Go** capabilities partial usable.

34.10.1. Trice format specifier

- The Trice macros are used in **C** code.
- The format strings are interpreted by the Trice tool, which is written in **Go**.
- The **C** and **Go** format specifier are not equal but similar.
- Therefore, a **T**rice adaption is internally performed.

34.10.2. Overview Table

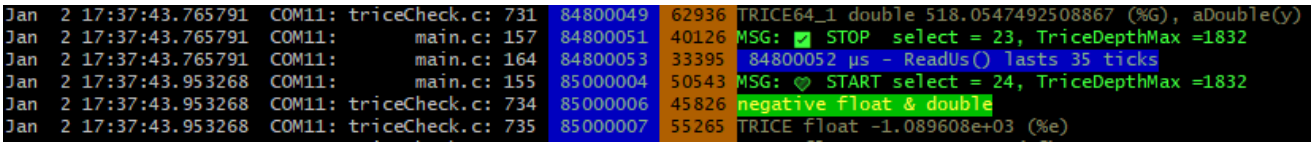
Format Specifier Type	C	Go	T	(T =Trice) | remark
signed decimal integer	d	d	d	Supported.
unsigned decimal integer	u	-	u	The Trice tool changes %u into %d and treats value as unsigned.
signed decimal integer	i	d	i	The Trice tool changes %i into %d and treats value as signed.
signed octal integer	-	o	o	With `trice log -unsigned=false` value is treated as signed.
unsigned octal integer	o	-	o	With `trice log` value is treated as unsigned.
signed octal integer with 0o prefix	-	O	O	With `trice log -unsigned=false` value is treated as signed.
unsigned octal integer with 0o prefix	-	-	O	With `trice log` value is treated as unsigned.
signed hexadecimal integer lowercase	-	x	x	With `trice log -unsigned=false` value is treated as signed.
unsigned hexadecimal integer lowercase	x	-	x	With `trice log` value is treated as unsigned.
signed hexadecimal integer uppercase	-	X	X	With `trice log -unsigned=false` value is treated as signed.
unsigned hexadecimal integer uppercase	X	-	X	With `trice log` value is treated as unsigned.
signed binary integer	-	b	b	With `trice log -unsigned=false` value is treated as signed.
unsigned binary integer	-	-	b	With `trice log` value is treated as unsigned.
decimal floating point, lowercase	f	f	f	`aFloat(value)|aDouble(value)`
decimal floating point, uppercase	-	F	F	`aFloat(value)|aDouble(value)`
scientific notation (mantissa/exponent), lowercase	e	e	e	`aFloat(value)|aDouble(value)`
scientific notation (mantissa/exponent), uppercase	E	E	E	`aFloat(value)|aDouble(value)`
the shortest representation of %e or %f	g	g	g	`aFloat(value)|aDouble(value)`

Format Specifier Type	C	Go	T	(T =Trice) | remark
the shortest representation of %E or %F	G	G	G	`aFloat(value)`|`aDouble(value)`
a character as byte	c	-	c	Value can contain ASCII character.
a character represented by the corresponding Unicode code point	c	c	c	Value can contain UTF-8 characters if the C-File is edited in UTF-8 format.
a quoted character	-	q	q	Supported.
the word true or false	-	t	t	Supported.
a string	s	s	s	Use `triceS` macro with one and only one runtime generated string.
pointer address	p	p	p	Supported.
a double %% prints a single %	%	%	%	Supported.
Unicode escape sequence	-	U	-	**Not supported.**
value in default format	-	v	-	**Not supported.**
Go-syntax representation of the value	-	#v	-	**Not supported.**
a Go-syntax representation of the type of the value	-	T	-	**Not supported.**
nothing printed	n	-	-	**Not supported.**

- ☑ Long story short: Use the `-unsigned=false` switch when you like to see hex numbers and the like as signed values.
- ☑ Look in triceCheck.c for exampe code producing this:

34.11. UTF-8 Support

This is gratis, if you edit your source files containing the format strings in UTF-8:



The target does not even "know" about that, because it gets only the Trice IDs.

34.12. Switch the language without changing a bit inside the target code

Once the til.json list is done the user can translate it in any language and exchanging the list switches to another language.

34.13. Format tags prototype specifier examples

This syntax is supported: `%[flags][width][.precision][length]`

- Because the interpretation is done inside the Trice tool written in Go these all should work:
  - `%-d`
  - `%064b`
  - `%+9.3f`
  - `%+#012.12g`
  - `%+'#012.12E`
  - `%e`
  - `%9.f`

(back to top)

35. Development Environment Setup

- Trice is usable with any C-compiler for any processor type, bit width and endianness. The example projects here are STM32 ones but illustrate how to setup Trice.
- The examples folder contains some instrumented example projects together with bare counterparts. Comparing a bare project with its intrumented counterpart gives a quick overview what needs to be done to get started.

35.1. Common Information

- All used tools are **Open Source** (despite the ARM-Keil µVision IDE, for new projects vsCode is a better choice).
- All provided information is just as example and needs adaption to your needs.
- There is no need to setup the environment in the given order.

35.2. Important to know

The ARM-Keil µVision IDE does sometimes not recognize external file modifications. That means for example: After editing `main.c` by adding a `trice( "Hi!\n" )` and executing `trice insert` as pre-compile step it could happen, that an updated `trice( iD(12345), "Hi!\n" )` was inserted and correct compiled but the update in `main.c` is not shown. Simply close and reopen `main.c` before editing again. This seems to be a ARM-Keil µVision IDE "feature" or be caused Windows not signaling a file change.

35.3. Animation

(The trice IDs occur just during the compilation.)

35.4. Setup Linux PC - Example with Debian12 - KDE Desktop

35.4.1. Basic setup

- Add yourself to the sudo group:

```
su
apt install sudo
adduser <your_user_name> sudo
exit
```

- Logout and login.
- Install and verify:

```
groups
sudo apt update
sudo apt upgrade
sudo apt install build-essential
make --version
gcc --version
git --version
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

35.4.2. Github

- Create github account.
- Create ssh pair:

```
ssh-keygen -t ed25519
```

- Add ssh key to your github account.
- Clone Trice repository:

```
cd ~
mkdir repos
cd repos
git clone git@github.com:rokath/trice.git
```

35.4.3. vsCode

- Download vsCode from https://code.visualstudio.com/download.
- Install vsCode (adapt to downloaded version) and start it inside the Trice folder:

```
sudo apt update
sudo apt upgrade
sudo apt install ~/Downloads/code_1.96.2-1734607745_amd64.deb
code .
```

35.4.4. Go

- Download the **Go** language from https://go.dev/doc/install and install:

```
cd ~/Downloads
sudo rm -rf /usr/local/go && sudo tar -C /usr/local -xzf go1.23.4.linux-amd64.tar.gz
```

Extend PATH variable with `/usr/local/go/bin:~/go/bin` for example by by adding a file like `/etc/profile.d/gopath.sh`:

```
su
sudo echo export PATH='$PATH':/usr/local/go/bin:/home/<your_user_name>/go/bin >
/etc/profile.d/gopath.sh
exit
```

- Logout, login and compile Trice:

```
th@P51-DebianKDE:~/repos$ go version
go version go1.23.4 linux/amd64
th@P51-DebianKDE:~/repos$ cd trice
th@P51-DebianKDE:~/repos/trice$ go install ./cmd/...
go: downloading github.com/spf13/afero v1.9.5
go: downloading github.com/kr/pretty v0.1.0
go: downloading go.bug.st/serial v1.6.0
go: downloading github.com/mgutz/ansi v0.0.0-20200706080929-d51e80ef957d
go: downloading github.com/rokath/cobs v0.0.0-20230425030040-4ebbe9b903b9
go: downloading github.com/rokath/tcobs v0.9.1
go: downloading golang.org/x/crypto v0.31.0
go: downloading github.com/fsnotify/fsnotify v1.6.0
go: downloading github.com/pkg/errors v0.9.1
go: downloading golang.org/x/sys v0.28.0
go: downloading golang.org/x/text v0.21.0
go: downloading github.com/kr/text v0.1.0
go: downloading github.com/mattn/go-colorable v0.1.13
go: downloading github.com/creack/goselect v0.1.2
go: downloading github.com/mattn/go-isatty v0.0.19
th@P51-DebianKDE:~/repos/trice$ trice version
version=devel, built 2025-01-04 16:29:30.51921408 +0100 CET
th@P51-DebianKDE:~/repos/trice$ go test ./...
```

```
go: downloading github.com/tj/assert v0.0.3
go: downloading github.com/stretchr/testify v1.8.4
go: downloading github.com/udhos/equalfile v0.3.0
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.1
go: downloading github.com/davecgh/go-spew v1.1.1
?       github.com/rokath/trice/internal/do     [no test files]
?       github.com/rokath/trice/internal/translator     [no test files]
?       github.com/rokath/trice/pkg/ant [no test files]
ok      github.com/rokath/trice/cmd/trice       1.014s
ok      github.com/rokath/trice/internal/args   0.009s
ok      github.com/rokath/trice/internal/charDecoder    0.005s
ok      github.com/rokath/trice/internal/com    0.005s
ok      github.com/rokath/trice/internal/decoder        0.005s
ok      github.com/rokath/trice/internal/dumpDecoder    0.006s
ok      github.com/rokath/trice/internal/emitter        0.006s
ok      github.com/rokath/trice/internal/id     2.744s
ok      github.com/rokath/trice/internal/keybcmd        0.006s
ok      github.com/rokath/trice/internal/link   0.006s
ok      github.com/rokath/trice/internal/receiver       0.007s
ok      github.com/rokath/trice/internal/trexDecoder    0.008s
ok      github.com/rokath/trice/pkg/cipher      0.006s
ok      github.com/rokath/trice/pkg/endian      0.002s
ok      github.com/rokath/trice/pkg/msg 0.005s
ok      github.com/rokath/trice/pkg/tst 0.003s
th@P51-DebianKDE:~/repos/trice$
th@P51-DebianKDE:~/repos/trice$ gcc --version
gcc (Debian 12.2.0-14) 12.2.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

th@P51-DebianKDE:~/repos/trice$ ./testAll.sh
Sa 4. Jan 16:33:57 CET 2025
This can take several minutes ...
?       github.com/rokath/trice/internal/do     [no test files]
ok      github.com/rokath/trice/cmd/trice       1.013s
ok      github.com/rokath/trice/internal/args   0.007s
ok      github.com/rokath/trice/internal/charDecoder    0.004s
ok      github.com/rokath/trice/internal/com    0.004s
ok      github.com/rokath/trice/internal/decoder        0.003s
ok      github.com/rokath/trice/internal/dumpDecoder    0.004s
ok      github.com/rokath/trice/internal/emitter        0.003s
?       github.com/rokath/trice/internal/translator     [no test files]
?       github.com/rokath/trice/pkg/ant [no test files]
ok      github.com/rokath/trice/internal/id     2.742s
ok      github.com/rokath/trice/internal/keybcmd        0.004s
ok      github.com/rokath/trice/internal/link   0.004s
ok      github.com/rokath/trice/internal/receiver       0.005s
ok      github.com/rokath/trice/internal/trexDecoder    0.005s
ok      github.com/rokath/trice/pkg/cipher      0.004s
ok      github.com/rokath/trice/pkg/endian      0.002s
ok      github.com/rokath/trice/pkg/msg 0.004s
ok      github.com/rokath/trice/pkg/tst 0.004s
ok      github.com/rokath/trice/_test/be_dblB_de_tcobs_ua       144.000s
ok      github.com/rokath/trice/_test/be_staticB_di_xtea_cobs_rtt32     143.927s
ok      github.com/rokath/trice/_test/dblB_de_cobs_ua   144.006s
ok      github.com/rokath/trice/_test/dblB_de_multi_cobs_ua     143.824s
ok      github.com/rokath/trice/_test/dblB_de_multi_nopf_ua     144.087s
ok      github.com/rokath/trice/_test/dblB_de_multi_tcobs_ua    144.100s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_cobs_ua        143.928s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_tcobs_ua       144.034s
ok      github.com/rokath/trice/_test/dblB_de_nopf_ua   142.399s
ok      github.com/rokath/trice/_test/dblB_de_tcobs_ua  142.520s
ok      github.com/rokath/trice/_test/dblB_de_xtea_cobs_ua      142.526s
ok      github.com/rokath/trice/_test/dblB_de_xtea_tcobs_ua     142.246s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_cobs_ua    281.643s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_multi_cobs_ua      281.201s
```

```
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_multi_tcobs_ua     281.518s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_tcobs_ua   281.536s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_xtea_cobs_ua       279.814s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_cobs_ua     280.173s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_multi_cobs_ua       280.095s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_multi_tcobs_ua      279.693s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_tcobs_ua    291.037s
ok      github.com/rokath/trice/_test/ringB_de_cobs_ua  140.861s
ok      github.com/rokath/trice/_test/ringB_de_multi_tcobs_ua   140.802s
ok      github.com/rokath/trice/_test/ringB_de_multi_xtea_cobs_ua       141.037s
ok      github.com/rokath/trice/_test/ringB_de_multi_xtea_tcobs_ua      149.046s
ok      github.com/rokath/trice/_test/ringB_de_nopf_ua  149.114s
ok      github.com/rokath/trice/_test/ringB_de_tcobs_ua 149.121s
ok      github.com/rokath/trice/_test/ringB_de_xtea_cobs_ua     149.089s
ok      github.com/rokath/trice/_test/ringB_de_xtea_tcobs_ua    149.164s
ok      github.com/rokath/trice/_test/ringB_di_cobs_rtt32__de_tcobs_ua  288.603s
ok      github.com/rokath/trice/_test/ringB_di_cobs_rtt8__de_tcobs_ua   288.569s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt32__de_tcobs_ua  278.409s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt32__de_xtea_cobs_ua      278.493s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt8__de_tcobs_ua   278.301s
ok      github.com/rokath/trice/_test/ringB_di_tcobs_rtt32__de_tcobs_ua 278.462s
ok      github.com/rokath/trice/_test/ringB_di_xtea_cobs_rtt32__de_xtea_cobs_ua 278.590s
ok      github.com/rokath/trice/_test/special_for_debug 0.129s
ok      github.com/rokath/trice/_test/special_protect_dblB_de_tcobs_ua  0.130s
ok      github.com/rokath/trice/_test/stackB_di_nopf_aux32      139.984s
ok      github.com/rokath/trice/_test/stackB_di_nopf_aux8       139.358s
ok      github.com/rokath/trice/_test/stackB_di_nopf_rtt32      139.366s
ok      github.com/rokath/trice/_test/stackB_di_nopf_rtt8       143.107s
ok      github.com/rokath/trice/_test/stackB_di_xtea_cobs_rtt8  141.896s
ok      github.com/rokath/trice/_test/staticB_di_nopf_aux32     141.398s
ok      github.com/rokath/trice/_test/staticB_di_nopf_aux8      141.677s
ok      github.com/rokath/trice/_test/staticB_di_nopf_rtt32     141.779s
ok      github.com/rokath/trice/_test/staticB_di_nopf_rtt8      141.609s
ok      github.com/rokath/trice/_test/staticB_di_tcobs_rtt32    141.487s
ok      github.com/rokath/trice/_test/staticB_di_tcobs_rtt8     141.559s
ok      github.com/rokath/trice/_test/staticB_di_xtea_cobs_rtt32        139.048s
Script run 1300 seconds.
th@P51-DebianKDE:~/repos/trice$
```

35.4.5. Gitkraken (or other GUI for git)

- Gitkraken download from https://www.gitkraken.com/download and Install:

```
mv ./gitkraken-amd64.deb /tmp; sudo apt install /tmp/gitkraken-amd64.deb
```

35.4.6. arm-none-eabi toolchain (or other target system compiler)

```
sudo apt install gcc-arm-none-eabi
sudo apt install binutils-arm-none-eabi
sudo apt install gdb-arm-none-eabi
sudo apt install openocd
arm-none-eabi-gcc --version
arm-none-eabi-gcc (15:12.2.rel1-1) 12.2.1 20221205
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- See installed toolchain:

```
ls -l /usr/bin/ | grep arm-none-eabi
-rwxr-xr-x 1 root root     1033504 Feb 28  2023 arm-none-eabi-addr2line
```

```
-rwxr-xr-x 2 root root     1066088 Feb 28  2023 arm-none-eabi-ar
-rwxr-xr-x 2 root root     2095024 Feb 28  2023 arm-none-eabi-as
-rwxr-xr-x 2 root root     1514496 Dec 22  2022 arm-none-eabi-c++
-rwxr-xr-x 1 root root     1032992 Feb 28  2023 arm-none-eabi-c++filt
-rwxr-xr-x 1 root root     1514496 Dec 22  2022 arm-none-eabi-cpp
-rwxr-xr-x 1 root root       43640 Feb 28  2023 arm-none-eabi-elfedit
-rwxr-xr-x 2 root root     1514496 Dec 22  2022 arm-none-eabi-g++
-rwxr-xr-x 2 root root     1514496 Dec 22  2022 arm-none-eabi-gcc
-rwxr-xr-x 2 root root     1514496 Dec 22  2022 arm-none-eabi-gcc-12.2.1
-rwxr-xr-x 1 root root       35376 Dec 22  2022 arm-none-eabi-gcc-ar
-rwxr-xr-x 1 root root       35376 Dec 22  2022 arm-none-eabi-gcc-nm
-rwxr-xr-x 1 root root       35376 Dec 22  2022 arm-none-eabi-gcc-ranlib
-rwxr-xr-x 1 root root      749664 Dec 22  2022 arm-none-eabi-gcov
-rwxr-xr-x 1 root root      585688 Dec 22  2022 arm-none-eabi-gcov-dump
-rwxr-xr-x 1 root root      610328 Dec 22  2022 arm-none-eabi-gcov-tool
-rwxr-xr-x 1 root root     1104256 Feb 28  2023 arm-none-eabi-gprof
-rwxr-xr-x 4 root root     1709968 Feb 28  2023 arm-none-eabi-ld
-rwxr-xr-x 4 root root     1709968 Feb 28  2023 arm-none-eabi-ld.bfd
-rwxr-xr-x 1 root root    24982344 Dec 22  2022 arm-none-eabi-lto-dump
-rwxr-xr-x 2 root root     1054720 Feb 28  2023 arm-none-eabi-nm
-rwxr-xr-x 2 root root     1180744 Feb 28  2023 arm-none-eabi-objcopy
-rwxr-xr-x 2 root root     1867744 Feb 28  2023 arm-none-eabi-objdump
-rwxr-xr-x 2 root root     1066120 Feb 28  2023 arm-none-eabi-ranlib
-rwxr-xr-x 2 root root      973400 Feb 28  2023 arm-none-eabi-readelf
-rwxr-xr-x 1 root root     1033280 Feb 28  2023 arm-none-eabi-size
-rwxr-xr-x 1 root root     1037504 Feb 28  2023 arm-none-eabi-strings
-rwxr-xr-x 2 root root     1180744 Feb 28  2023 arm-none-eabi-strip
```

- For some reason `sudo apt install gdb-arm-none-eabi` gives the message `Note, selecting 'gdb-multiarch' instead of 'gdb-arm-none-eabi'` and *arm-none-eabi-gdb* is not installed afterwards.
- To try the newest version, download it from https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads.
    - Setup:
        - Unpack and extend $PATH at the beginning:

```
cd ~/Downloads
tar xf arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.xz
su
echo export PATH=/home/th/Downloads/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-
eabi/bin/:'$PATH' > /etc/profile.d/arm14.2path.sh
exit
```

        - Logout and login

```
arm-none-eabi-gcc --version
arm-none-eabi-gcc (Arm GNU Toolchain 14.2.Rel1 (Build arm-14.52)) 14.2.1 20241119
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
arm-none-eabi-gdb --version
GNU gdb (Arm GNU Toolchain 14.2.Rel1 (Build arm-14.52)) 15.2.90.20241130-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

    - Remove: Delete `/etc/profile.d/arm14.2path.sh` and `~/Downloads/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi`

35.4.7. J-Link (if needed)

- Download and install from https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack

```
sudo apt install ~/Downloads/JLink_Linux_V812_x86_64.deb
```

- Logout & login & check:

```
th@P51-DebianKDE:~/Downloads$ JLinkRTTLogger -?
SEGGER J-Link RTT Logger
Compiled Dec 18 2024 15:48:21
(c) 2016-2017 SEGGER Microcontroller GmbH, www.segger.com
         Solutions for real time microcontroller applications

Default logfile path: /home/th/.config/SEGGER

------------------------------------------------------------

Available options:
-Device <devicename>
-If <ifname>
-Speed <speed>
-USB <SN>
-IP <SN>
-RTTAddress <RTTAddress>
-RTTSearchRanges "<Rangestart> <RangeSize>[, <Range1Start> <Range1Size>, ...]
"-RTTChannel <RTTChannel>
-JLinkScriptFile <PathToScript>
<OutFilename>

Shutting down... Done.th@P51-DebianKDE:~/Downloads$
```

35.4.8. Beyond Compare (if no other diff tool)

- Download and install from https://www.scootersoftware.com

35.5. Setup Windows PC Example

Setting up a PC is for Linux mostly straightforward but Windows PCs are more problematic. The steps shown here are just one example.

- Create folder `repos` in your home directory.
  - Clone all repositories here.
- Create `C:\bin` folder.
  - When installing toolchains, put them here then and avoid spaces in created paths.
- Add `C:\bin` to PATH variable at the beginning.
  - This allows to copy tools like `trice.exe` simply into `C:\bin`.
- Install "Git for windows" from https://git-scm.com/downloads to get the neat git bash.
  - Select the Standalone Installer. This gives you useful context menu entries in the Windows explorer.
- BTW: For managing git repositories I like https://www.gitkraken.com/. Its free of charge for open source programs.
- Install VS-Code
  - This is my favorite editor with many optional Add-Ons. It is used for debugging as well.
- Install Go if you wish to compile Go programs.
  - `go test ./...` should succeed in a terminal window.
  - When GCC is not installed you get some strange errors with `TestAll.sh`:

```
ms@PaulPCWin11 MINGW64 ~/repos/trice (devel)
$ ./testAll.sh
Fri Jan 10 11:19:39 WEST 2025
This can take several minutes ...
```

```
...

ok      github.com/rokath/trice/pkg/tst 0.601s
# github.com/rokath/trice/_test/dblB_de_cobs_ua
[github.com/rokath/trice/_test/dblB_de_cobs_ua.test]
dblB_de_cobs_ua\cgo_test.go:20:110: undefined: triceDir
dblB_de_cobs_ua\cgo_test.go:24:2: undefined: triceLogTest
dblB_de_cobs_ua\cgo_test.go:24:28: undefined: testLines
# github.com/rokath/trice/_test/dblB_de_tcobs_ua
[github.com/rokath/trice/_test/dblB_de_tcobs_ua.test]
dblB_de_tcobs_ua\cgo_test.go:20:92: undefined: triceDir
dblB_de_tcobs_ua\cgo_test.go:24:2: undefined: triceLogTest
dblB_de_tcobs_ua\cgo_test.go:24:28: undefined: testLines
```

    - Download and install latest GCC then, from https://winlibs.com for example.
        - Extract to `C:\bin\mingw64` and extend PATH with `C:\bin\mingw64\bin`
        - Open terminal and execute `gcc --version` to check success.
- Setup J-Link if you use this debug probe as hardware or software (see below).
    - Install SEGGER J-Link Software and Documentation Pack
- Install Make for Windows and add its installation bin folder location to the PATH variable.

35.5.1. Setup Trice

- from inside folder `repos` clone trice repo with `git clone https://github.com/rokath/trice.git`.
- Run `go install ./cmd/trice/...` from folder `repos/trice`.

OR

- Download the latest release archive and extract.
- Put trice binary into C:\bin.
- Put trice/src into `repos` if you want access the trice library code from several projects and have it only once.
    - Alternatively copy it into your project.

35.5.2. Setup ARM Environment Example

Install make

- Fast lane: Go to https://sourceforge.net/projects/ezwinports/files/, download ans extract **make-4.4.1-without-guile-w32-bin.zip**. Put the *make.exe* file somewhere in you $PATH.

- OR open the Windows Powershell and:

```
ms@PaulPCWin11 MINGW64 ~/repos/trice/examples (devel)
$ winget install ezwinports.make
The `msstore` source requires that you view the following agreements before using.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
The source requires the current machine's 2-letter geographic region to be sent to the backend service to
function properly (ex. "US").

Do you agree to all the source agreements terms?
[Y] Yes  [N] No: Y
Found ezwinports: make [ezwinports.make] Version 4.4.1
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://downloads.sourceforge.net/project/ezwinports/make-4.4.1-without-guile-w32-bin.zip
  ██████████████████████████████  383 KB /  383 KB
Successfully verified installer hash
Extracting archive...
Successfully extracted archive
Starting package install...
Path environment variable modified; restart your shell to use the new value.
Command line alias added: "make"
Successfully installed
```

```
ms@PaulPCWin11 MINGW64 ~/repos/trice/examples (devel)
```

- Check:

```
$ make --version
GNU Make 4.4.1
Built for Windows32
Copyright (C) 1988-2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Install ARM GCC

- Uninstall existing ARM GCC compilers optionally.
- Check if $PATH is clean.
  - In fact you can leave it as it is.
- Check if `C_INCLUDE_PATH` is not defined.
  - It would cause issues performing Go tests with CGO.
  - The environment variable `C_INCLUDE_PATH` is needed by Clang, but is set temporarily inside the Makefile.
- Open a console.
  - `which arm-none-eabi-gcc` should return `no arm-none-eabi-gcc` if no legacy compilers installed.
- Download latest version from https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads.
- Install exactly into `C:\bin\ArmGNUToolchain` and add `C:\bin\ArmGNUToolchain\bin` it to the **beginning** of the path variable.
  - The path is temporarily extendable inside the manually edited Makefiles as well, but for debugging the Windows path is used directly.
  - The path is used directly also in generated Makefiles.
  - Extending the path variable at the beginning avoids accidentally usage of installed legacy compilers with the same name.
  - To use exactly `C:\bin\ArmGNUToolchain` as install location allows a more easy project collaboration.
  - Check path:

```
$ echo $PATH | tr : '\n'
...
/c/bin
/c/bin/ArmGNUToolchain/bin
/c/bin/mingw64/bin
/c/Program Files/Go/bin
/c/Users/ms/go/bin
/c/Users/ms/AppData/Local/Microsoft/WinGet/Packages/ezwinports.make_Microsoft.Winget.Source_8wekyb3d8
bbwe/bin
...
```

MacOS

- In terminal `brew install arm-none-eabi-gcc`
- Restart terminal
- In teminal `arm-non-eabi-gcc --version` delivers `arm-none-eabi-gcc (GCC) 14.2.0`
- In terminal `brew install arm-none-eabi-clang`
- Restart terminal
- In teminal `clang -target arm-none-eabi --version` delivers:

```
Apple clang version 15.0.0 (clang-1500.3.9.4)
Target: arm-none-unknown-eabi
```

```
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

- In terminal `brew install arm-none-eabi-gdb`
- In terminal `brew install --cask gcc-arm-embedded`
- In terminal to get objcopy:

```
brew install binutils
echo 'export PATH="/usr/local/opt/binutils/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

Install ARM Clang (optional)

With the ARM Clang you get quicker compilation runs and smaller images.

- You need to install ARM GCC as well to use ARM Clang.
  - ARM Clang uses the GCC libraries. For that it looks for `C_INCLUDE_PATH`.
  - ARM Clang uses the GCC debugger. For that it looks into the Windows path variable directly.
- Uninstall existing ARM clang compilers or make sure they are hidden.
- Check if $PATH is clean.
  - In fact you can leave is as it is.
- Check if `C_INCLUDE_PATH` is not defined.
  - It would cause issues performing Go tests with CGO.
- Open a console.
  - `which clang` should return `no clang`.
  - If you have other clang compilers installed, do not touch them.
- Download latest version from https://github.com/llvm/llvm-project/releases.
- Install exactly into `C:\bin\ArmClang` and do **not** add it to path variable.
  - The path is extended temporarily inside the Makefile for the compiler run.

Check Project Makefile (if it already exists)

- The Makefile should start with these lines:

```
# Put ARM Clang first in path temporary to avoid compiler variants issues.
export PATH := C:\bin\ArmClang\bin:$(PATH)

# ARM Clang uses the ARM GNU toolchain libraries and finds them over C_INCLUDE_PATH.
export C_INCLUDE_PATH := C:\bin\ArmGNUToolchain\arm-none-eabi\include
```

The `C:\bin\ArmGNUToolchain\bin:` is in fact not needed, because it must be in the path anyway for debugging.

- `make version` should give output like that:

```
$ make version
/c/bin/ArmGNUToolchain/bin/arm-none-eabi-gcc
arm-none-eabi-gcc (Arm GNU Toolchain 12.3.Rel1 (Build arm-12.35)) 12.3.1 20230626
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

/c/bin/ArmClang/bin/clang
clang version 17.0.0
Target: x86_64-pc-windows-msvc
Thread model: posix
InstalledDir: C:\bin\ArmClang\bin
```

The paths must match with the installation locations.

35.5.3. Setup STM32

Generate Base Project

- Install and start STM32CubeMX code generator.
- Board-Selector -> STM32G0B1KE`or`STM32L432KC` or ...
- (Auto-)Initialize with default values.
- Clock-Generation -> Change PLL *N from "X 16" to "X 32" to get 64 MHz clocks.
  - Running at max clock speed and using `WFE` instructions in wait loops is slightly more energy efficient.
- Project Manager
  - Project
    - Set Project Name
    - Select Project Location
    - Toolchain / IDE -> Select Makefile
  - Code Generator
    - Select "Copy only the necessary library files".
  - Advanced Settings
    - Switch from HAL to LL at least for UART
- Generate Code as Makefile project

Update NUCLEO Onboard Debugger (other ST evaluation boards too)

(https://www.st.com/en/development-tools/stsw-link007.html)

This step is recommended before re-flashing with the J-Link onboard debugger software.

- Connect STM evaluation board over USB
- Start ST-Link Upgrade (trice\third_party\st.com or look for a newer version at STM.).
  - Device Connect
  - Upgrade Firmware (select version **with** mass storage option)
    - Selecting the other option, would not allow to update with the SEGGER STLinkReflash tool.
  - Close

35.5.4. Setup Onboard J-Link on NUCLEO (other ST evaluation boards too)

(https://www.segger.com/products/debug-probes/j-link/models/other-j-links/st-link-on-board/)

Using the J-Link onboard debugger software allows parallel debugging and RTT usage.

Unfortunately this is not possible with **v3** onboard debugger hardware! But you can use a J-Link hardware instead. Also it is possible to use a v2 onboard debugger from a different evaluation board or a "Bluepill" Development Board Module with ARM Cortex M3 processor".

- Start STLinkReflash (trice\third_party\segger.com)
  - Accept and Accept
  - 1: Upgrade to J-Link
  - 0: Quit
- Download, extract & start https://github.com/rokath/trice/blob/main/thomasthomasthird_party/segger.com/STLinkReflash_190812.zip
  - Re-Flash onboard debugger.
    - You can undo this step anytime.

35.5.5. Setup VS-Code

- Start VS Code
  - Install Go rich language support if you want to use Go as well (not needed for ARM debugging).
  - Install "Cortex Debug" extension.
  - Open the generated project directory.
  - Click on Run and Debug.
    - Click Generate launch.json and select "Cortex Debug"
  - Open and edit .vscode/launch.json
    - change "executable" value into: "./build/STM32G0B1KE_generated.elf" (example)

    - add lines:
        - `"device": "STM32G0B1KE",` or `"STM32L432KC"` or ...
        - `"svdFile": "./STM32G0B1KE.svd",` or `"./STM32L4x2.svd"` or ...
        - `"runToMain": true`
    - Set the commas right.
- Latest SVD Files can be found here: https://www.st.com/content/st_com/en/search.html#q=svd-t=resources-page=1
- Download file `STM32G0B1.svd` from https://www.st.com/resource/en/svd/stm32G0_svd.zip (example)
    - Alternatively copy it from `"C:\ST\STM32CubeIDE_1.13.1\STM32CubeIDE\plugins\com.st.stm32cube.ide.mcu.productdb.debug_2.1.0.202306151215\resources\cmsis\STMicroelectronics_CMSIS_SVD\STM32G0B1.svd"` if you have the STM32CubeIDE installed.
    - Download file `STM32L4x2.svd` from https://www.st.com/resource/en/svd/stm32l4_svd.zip (example)
- Installing the **Cortex Debug** extension allow you to debug the target code.

35.6. Makefile with Clang too

- After STM32 CubeMX code generation the Makefile was edited and spitted.
- STM32 CubeMX code generation accepts the edited Makefile, so re-generation is no issue.
    - It modifies the settings according to the changes.

35.7. Download Locations

35.7.1. Clang

https://releases.llvm.org/download.html -> https://github.com/llvm/llvm-project/releases/tag/llvmorg-16.0.0 (example)

35.7.2. GCC

https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain -> https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads (example))

35.8. Install Locations

Do not use locations containing spaces, like `C:\Program Files`. Take `C:\bin` for example. This avoids trouble caused by spaces inside path names.

35.9. Environment Variables

Extend the path variable:

- PATH += `C:\bin\ArmGNUToolchain\bin`
- PATH += `C:\Program Files\SEGGER\JLink`. (may be C:\Program Files\SEGGER\JLink_V812a or similar)

35.10. Build command

- Clang: `make` or to get it faster `make -j`.
- GCC: `make GCC`.

35.11. Run & Debug

- In terminal after `make` click Run&Debug & click green triangle.

35.12. Logging

- In terminal type `make log`. This executes the command in project folder:

`trice l -p JLINK -args="-Device STM32G0B1RE -if SWD -Speed 4000 -RTTChannel 0" -pf none -ts ms -d16` (example)

35.13. Setting up a new project

- Copy this project folder under a new name like `myAwesomeNewProject` or name it as you like.
- Make a temporary folder `myTemp` and generate with STM CubeMX the base project.
- Copy the *.ioc file from `myTemp` to `myAwesomeNewProject` and name it to the project name.
- Compare `myTemp\Makefile` with `myAwesomeNewProject\Makefile` and overwrite/extend in `myAwesomeNewProject\Makefile` the relevant settings, mainly the filenames, include path settings and DEFINES.
- Replace all generated files in `myAwesomeNewProject` with the ones in `myTemp`
- Replace the *.svd file if the MCU is different. You can find it in the internet.
- Run `make -j8` inside `myAwesomeNewProject` to check if all is ok.
- Open the copied *ioc file inside `myAwesomeNewProject` and re-generate and re-build to check.
- Compare the relevant files like `main.c` with the starting project and edit accordingly.
- Adapt `.vscode/launch.json` to the used MCU.
- Than the awesome new project should be ready to go for development.

(back to top)

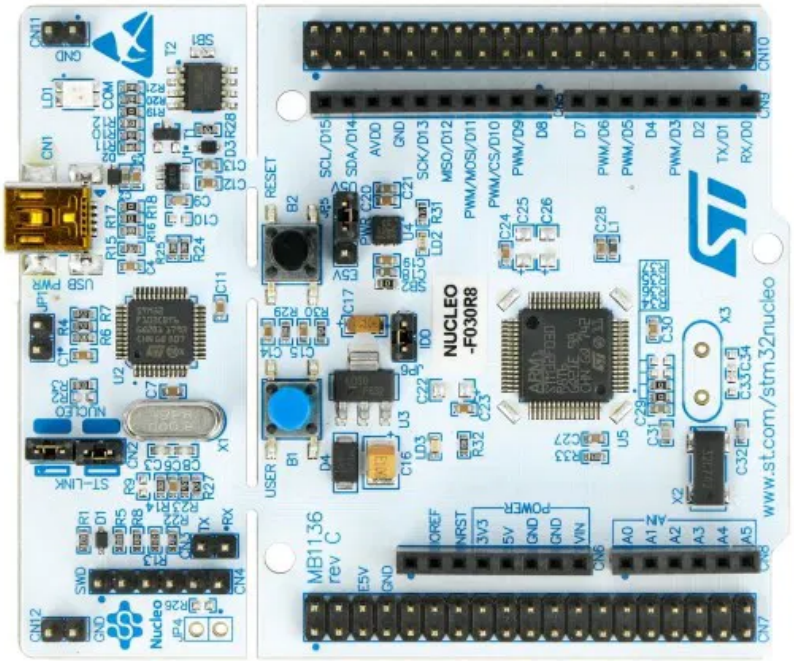
36. Example Projects without and with Trice Instrumentation

Project Name	Description
F030_bare	This is a minimal STM32CubeMX generated Makefile project adapted to Clang and GCC. It serves as a reference for diff to F030_inst so see quickly the needed instrumentation steps you need for your own project.
F030_inst	This is a minimal STM32CubeMX generated Makefile project adapted to Clang and GCC and afterward instrumented with the Trice library. Compare it with F030_bare to see quickly how to instrument your project.
G0B1_bare	This is a minimal FreeRTOS STM32CubeMX generated Makefile project adapted to Clang and GCC.
G0B1_inst	This is a minimal FreeRTOS STM32CubeMX generated Makefile project adapted to Clang and GCC and afterward instrumented with the Trice library.

Project Name	Description
L432_bare	This is a minimal FreeRTOS STM32CubeMX generated Makefile project extended to compile also with Clang trying to perform minimal changes. It produces some warnings, because it is not finetuned. The L432_inst project is then a next step performable.
L432_inst	This is a minimal FreeRTOS STM32CubeMX generated Makefile project adapted to Clang and GCC and afterward instrumented with the Trice library.

(back to top)

36.1. Nucleo-F030R8 Examples



36.1.1. F030_bare

Folder: ../examples/F030_bare/

This is a STMCubeMX generated project without Trice instrumentation for easy compare with F030_inst to figure out the needed changes to set up trice.

Steps performed as potential guide:

- Install STM32CubeMX to `C:\SMT32SubeMX`.
- Select NUCLEO-F030R8 board.
- Initialize with default values.
- Optionally set system clock to 32MHz for faster target timestamps.
- Optionally set UART baud rate to 115200.
- Mantadory set UART data bits including parity to **9**.
- Enable USART2 global interrupt.
- In Project Manager *Project*:
  - Set toolchain folder location to `E:\repos\trice\examples\F030_bare\`.
  - Set project name to `F030_bare`.
  - Set toolchain / IDE to `Makefile`.

- In Project Manager *Code Generator*:
  - Select "Copy only the necessary library files".
- In Project Manager *Advanced Settings*:
  - In Driver Selector change all to *LL*.
- Generate Code
- Start vsCode and open folder F030_bare with it.
- Start a terminal and type `make`. The output should be similar to:

```
PS E:\repos\trice\examples\F030_bare> make -j
mkdir build
arm-none-eabi-gcc -c -mcpu=cortex-m0 -mthumb   -DUSE_FULL_LL_DRIVER -DHSE_VALUE=8000000 -
DHSE_STARTUP_TIMEOUT=100 -DLSE_STARTUP_TIMEOUT=5000 -DLSE_VALUE=32768 -DHSI_VALUE=8000000 -
DLSI_VALUE=40000 -DVDD_VALUE=3300 -DPREFETCH_ENABLE=1 -DINSTRUCTION_CACHE_ENABLE=0 -DDATA_CACHE_ENABLE=0 -
DSTM32F030x8 -ICore/Inc -IDrivers/STM32F0xx_HAL_Driver/Inc -IDrivers/CMSIS/Device/ST/STM32F0xx/Include -
IDrivers/CMSIS/Include -Og -Wall -fdata-sections -ffunction-sections -g -gdwarf-2 -MMD -MP -
MF"build/main.d" -Wa,-a,-ad,-alms=build/main.lst Core/Src/main.c -o build/main.o

...

arm-none-eabi-gcc -x assembler-with-cpp -c -mcpu=cortex-m0 -mthumb   -DUSE_FULL_LL_DRIVER -
DHSE_VALUE=8000000 -DHSE_STARTUP_TIMEOUT=100 -DLSE_STARTUP_TIMEOUT=5000 -DLSE_VALUE=32768 -
DHSI_VALUE=8000000 -DLSI_VALUE=40000 -DVDD_VALUE=3300 -DPREFETCH_ENABLE=1 -DINSTRUCTION_CACHE_ENABLE=0 -
DDATA_CACHE_ENABLE=0 -DSTM32F030x8 -ICore/Inc -IDrivers/STM32F0xx_HAL_Driver/Inc -
IDrivers/CMSIS/Device/ST/STM32F0xx/Include -IDrivers/CMSIS/Include -Og -Wall -fdata-sections -ffunction-
sections -g -gdwarf-2 -MMD -MP -MF"build/startup_stm32F030x8.d" startup_stm32F030x8.s -o
build/startup_stm32F030x8.o
arm-none-eabi-gcc build/main.o build/stm32f0xx_it.o build/stm32f0xx_ll_gpio.o build/stm32f0xx_ll_pwr.o
build/stm32f0xx_ll_exti.o build/stm32f0xx_ll_usart.o build/stm32f0xx_ll_rcc.o build/stm32f0xx_ll_dma.o
build/stm32f0xx_ll_utils.o build/system_stm32f0xx.o build/sysmem.o build/syscalls.o
build/startup_stm32F030x8.o  -mcpu=cortex-m0 -mthumb   -specs=nano.specs -TSTM32F030R8Tx_FLASH.ld  -lc -lm
-lnosys  -Wl,-Map=build/F030_bare.map,--cref -Wl,--gc-sections -o build/F030_bare.elf
C:/bin/ArmGNUToolchain/bin/../lib/gcc/arm-none-eabi/13.2.1/../../../../arm-none-eabi/bin/ld.exe: warning:
build/F030_bare.elf has a LOAD segment with RWX permissions
arm-none-eabi-size build/F030_bare.elf
   text    data     bss     dec     hex filename
   2428      12    1564    4004     fa4 build/F030_bare.elf
arm-none-eabi-objcopy -O ihex build/F030_bare.elf build/F030_bare.hex
arm-none-eabi-objcopy -O binary -S build/F030_bare.elf build/F030_bare.bin
PS E:\repos\trice\examples\F030_bare>
```

- Install vsCode Cortex-Debug extension.
- Create a launch.json file inside the *.vscode* subfolder and edit it to get

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Cortex Debug",
            "cwd": "${workspaceFolder}",
            "executable": "./build/F030_bare.elf",
            "request": "launch",
            "type": "cortex-debug",
            "runToEntryPoint": "main",
            "servertype": "jlink",
            "device": "STM32F030R8",
            "svdFile": "./STM32F030R8.svd",
            "runToMain": true

        }
```

```
    ]
}
```

- Download STM32G030.svd or get it from the STMCubeIDE installation folder if you want to install this Eclipse IDE as well, but IMHO you do not need it.
- You may need to extract and install the STM32 USB drivers. You can find them also in `./third_party/st.com/en.stsw-link009_v2.0.2.zip`.
- It is assumed, that you converted the OB ST-Link to an OB J-Link already. See Convert Evaluation Board onboard ST-Link to J-Link for details.
- Press the Debug-Button or "CTRL+SHIFT+D" and start debugging.

Hint

- During the code generation, the CubeMX tool did not copy `syscalls.c` and `sysmem.c` but added them to the Makefile. This seems to be a STM32CubeMX "feature".
    - You do not need these files for the example project, but you can add them manually to avoid some warnings or extend the code with:

```
__weak int _close(void) { return -1; }
__weak int _lseek(void) { return -1; }
__weak int _read (void) { return -1; }
__weak int _write(void) { return -1; }
```

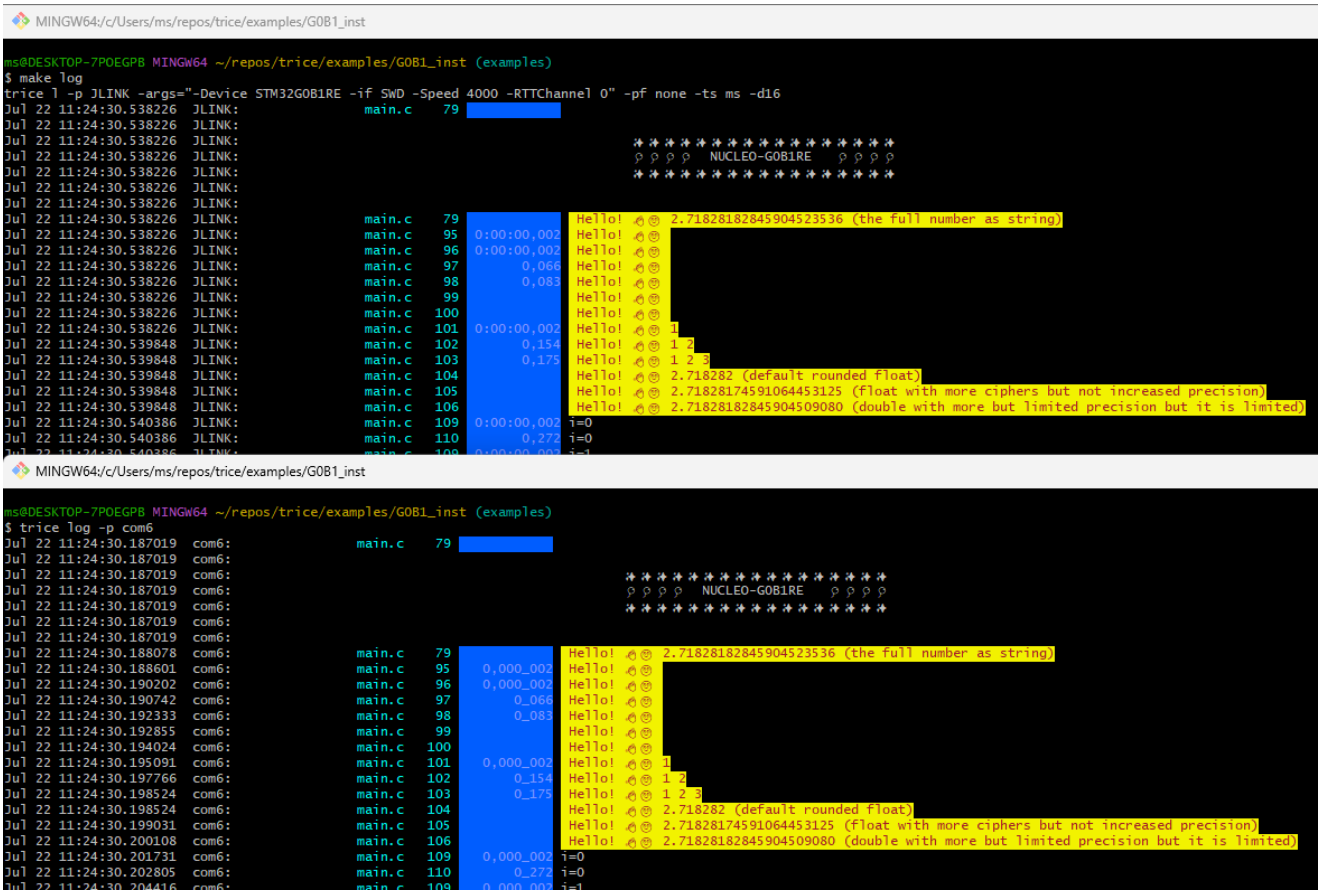
36.1.2. F030_inst

Folder: ../examples/F030_inst/

This is a working example with deferred encrypted out over UART. By uncommenting 2 lines in triceConfig.h, you get also parallel direct out over RTT. For setup see Trice over RTT and adapt steps from F030_bare.

Intrumenting:

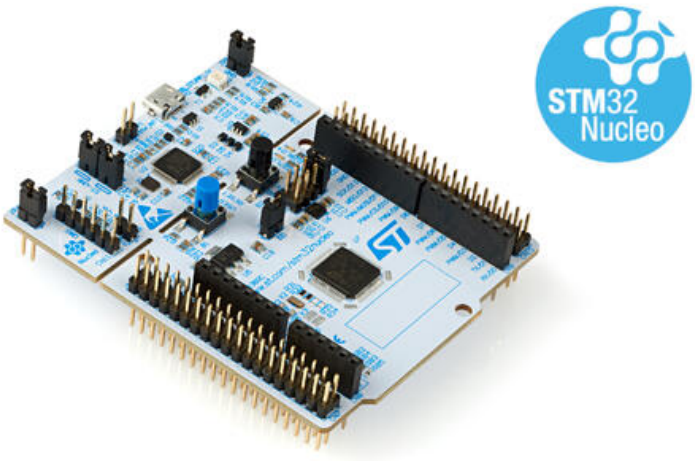
- Extend the Makefile with the information you get from comparing the *Makefile* here and in ../F030_bare/.
- Add *build.sh* and *clean.sh*.
- Copy file *SEGGER_RTT_Conf.h* from *trice/third_party/segger.com/SEGGER_RTT_V760g.zip* to ./Core/Inc/. Yu could also look for a newer version.
- Copy and adapt a file triceConfig.h to ./Core/Inc/. You can choose from an other example project or one of the test folders.
- Create 2 empty files: `touch til.json li.json`inside ./
- Run `build.sh`. This should build all.
- Add `#include "trice.h"` to *main.c* and to *stm32f0xx_it.c* and edit these files according to diff.
- Add to `int main( void )` some `Trice( "..." );` messages.
- Run `trice s` to determine the relevant comport.
- You can have this output:



- The Trices with 16-bit timestamps are about 150 clocks away from each other. @32MHz this is a time of less 5 µs.

(back to top)

36.2. Nucleo-G0B1 Examples



36.2.1. G0B1_bare

Folder: ../examples/G0B1_bare/

G0B1_bare Description

- This is a working example with CLang and also GCC.
- This is a STMCubeMX generated project. It was then manually adapted to Clang.
- It is without TRICE instrumentation for easy compare with ../G0B1_inst to figure out the needed changes to set up trice.

Setting Up G0B1_bare

- See and adapt steps from F030_bare.
- Then add/modify the files to reach this folder layot.

36.2.2. G0B1_inst

Folder: ../examples/G0B1_inst/

This is an example with direct out without framing over RTT and deferred out in TCOBS framing over UART.

Setting Up

- See and adapt steps from G0B1_bare.

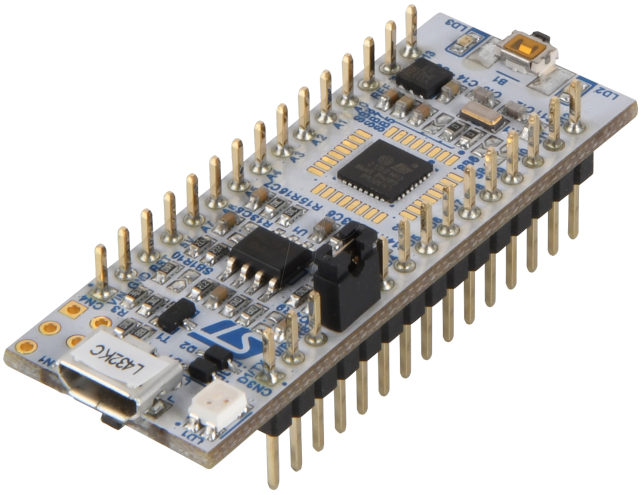
Instrumenting

- The steps are similar to the steps in F030_bare.
- See comments in triceConfig.h and commandlines in screenshot.

```
ms@LenovoP51Win11 MINGW64 /e/repos/trice/examples/F030R8_inst (master)
$ trice l -p com5 -pf cobs -pw MySecret -showID "d:%04x" -ts16 "time:     #%6d"
Jul 22 23:06:44.644126  com5:                main.c    71            1682
Jul 22 23:06:44.644725  com5:
Jul 22 23:06:44.644725  com5:                                                  ✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨✨
Jul 22 23:06:44.644725  com5:                                                  🎈🎈🎈 NUCLEO-F030R8 🎈🎈🎈🎈
Jul 22 23:06:44.644725  com5:                                                  🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿🌿
Jul 22 23:06:44.644725  com5:
Jul 22 23:06:44.644725  com5:
Jul 22 23:06:44.646768  com5:                main.c   109      # 28445 1206  Hello! 🔥😊
Jul 22 23:06:44.647324  com5:                main.c   107      # 28294 0cfe  Hello! 🔥😊
Jul 22 23:06:44.647324  com5:                main.c   106      # 28142 1205  Hello! 🔥😊
Jul 22 23:06:44.647992  com5:                main.c   104  0,000_000 0b76  Hello! 🔥😊
Jul 22 23:06:44.648559  com5:                main.c   105  0,000_000 0eff  Hello! 🔥😊
Jul 22 23:06:44.649938  com5:                main.c   106      # 27693 1205  Hello! 🔥😊
Jul 22 23:06:44.650448  com5:                main.c   107      # 27544 0cfe  Hello! 🔥😊
Jul 22 23:06:44.651792  com5:                main.c   108      # 27393 07b3  Hello! 🔥😊
Jul 22 23:06:44.652364  com5:                main.c   109      # 27244 1206  Hello! 🔥😊
Jul 22 23:06:44.653849  com5:                main.c   110            0706  Hello! 🔥😊
Jul 22 23:06:44.653849  com5:                main.c   111            0c51  Hello! 🔥😊
Jul 22 23:06:44.732464  com5:          triceCheck.c  1500            3ecc  01
Jul 22 23:06:44.833793  com5:          triceCheck.c  1500            3ecc  012
Jul 22 23:06:44.933973  com5:          triceCheck.c  1500            3ecc  0123
```

(back to top)

36.3. Nucleo-L432KC Examples



36.3.1. L432_bare

Folder: ../examples/L432_bare/

- This example is without Trice istrumentation and serves for comparing with L432_inst to see the needed instrumentation steps quickly.
- This is a STMCubeMX generated project.

- See and adapt steps from F030_bare example.
- It was then manually adapted additionally to Clang.
- It was additionally configured for FreeRTOS.

36.3.2. L432_inst

Folder: ../examples/L432_inst/

- This is the with Trice instrumented example project L432_bare.
- It is for easy compare to figure out the needed setup changes.
- See and adapt steps in F030_bare.
- Then add/modify the files to reach this folder layout.

Build:

Run `./build.sh` for configuration 0 or `./build.sh CONFIGURATION=34` for example.

Deferred Mode for max Speed

The stamps are MCU clocks here, so 🐔 `Speedy Gonzales` lasts 9 processor clocks here.

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice_wt_devel/examples/L432_inst (devel)
$ trice l -p com8 -hs off -prefix off
      triceExamples.c    10        0_272  Hello! 👌😁

        ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
        ☙ ☙ ☙ ☙  𝔑𝔘𝔈𝔏𝔈𝔒-L432KC   ☙ ☙ ☙ ☙
        🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃


        triceConfig.h   369             CONFIGURATION == 34 - UART, no cycle counter, no critical
sections.
      triceExamples.c    45             TRICE_DIRECT_OUTPUT == 0, TRICE_DEFERRED_OUTPUT == 1
      triceExamples.c    51             TRICE_DOUBLE_BUFFER, TRICE_MULTI_PACK_MODE
      triceExamples.c    60             _CYCLE == 0, _PROTECT == 0, _DIAG == 0, XTEA == 0
      triceExamples.c    61             _SINGLE_MAX_SIZE=512, _BUFFER_SIZE=580,
_DEFERRED_BUFFER_SIZE=4096
      triceExamples.c    15    0,000_731 🐔 Speedy Gonzales
      triceExamples.c    16    0,000_745 🐔 Speedy Gonzales
      triceExamples.c    17    0,000_754 🐔 Speedy Gonzales
      triceExamples.c    18    0,000_763 🐔 Speedy Gonzales
      triceExamples.c    19    0,000_772 🐔 Speedy Gonzales
      triceExamples.c    20    0,000_781 🐔 Speedy Gonzales
      triceExamples.c    21    0,000_790 🐔 Speedy Gonzales
      triceExamples.c    22    0,000_799 🐔 Speedy Gonzales
      triceExamples.c    24        0_981 2.71828182845904523536 <- float number as string
      triceExamples.c    25        1_230 2.71828182845904509080 (double with more ciphers than precision)
      triceExamples.c    26        1_268 2.71828174591064453125 (float  with more ciphers than precision)
      triceExamples.c    27        1_296 2.718282 (default rounded float)
      triceExamples.c    28        1_310 A Buffer:
      triceExamples.c    29        1_348 32 2e 37 31 38 32 38 31 38 32 38 34 35 39 30 34 35 32 33 35 33 36
      triceExamples.c    30        1_603 31372e32  31383238  34383238  34303935  35333235
      triceExamples.c    31        1_799 ARemoteFunctionName(2e32)(3137)(3238)(3138)(3238)(3438)(3935)
(3430)(3235)(3533)(3633)
      triceExamples.c    32              10 times a 16 byte long Trice messages, which not all will be
written because of the TRICE_PROTECT:
      triceExamples.c    34        2_072 i=44444400 aaaaaa00
      triceExamples.c    34        2_119 i=44444401 aaaaaa01
      triceExamples.c    34        2_166 i=44444402 aaaaaa02
```

"Hardware" Changes

- The used evaluation board is delivered with an on-board ST-Link software for debugging.

- This was changed to an on-board J-Link software for better debugging and RTT support.
- See Trice over RTT about that.

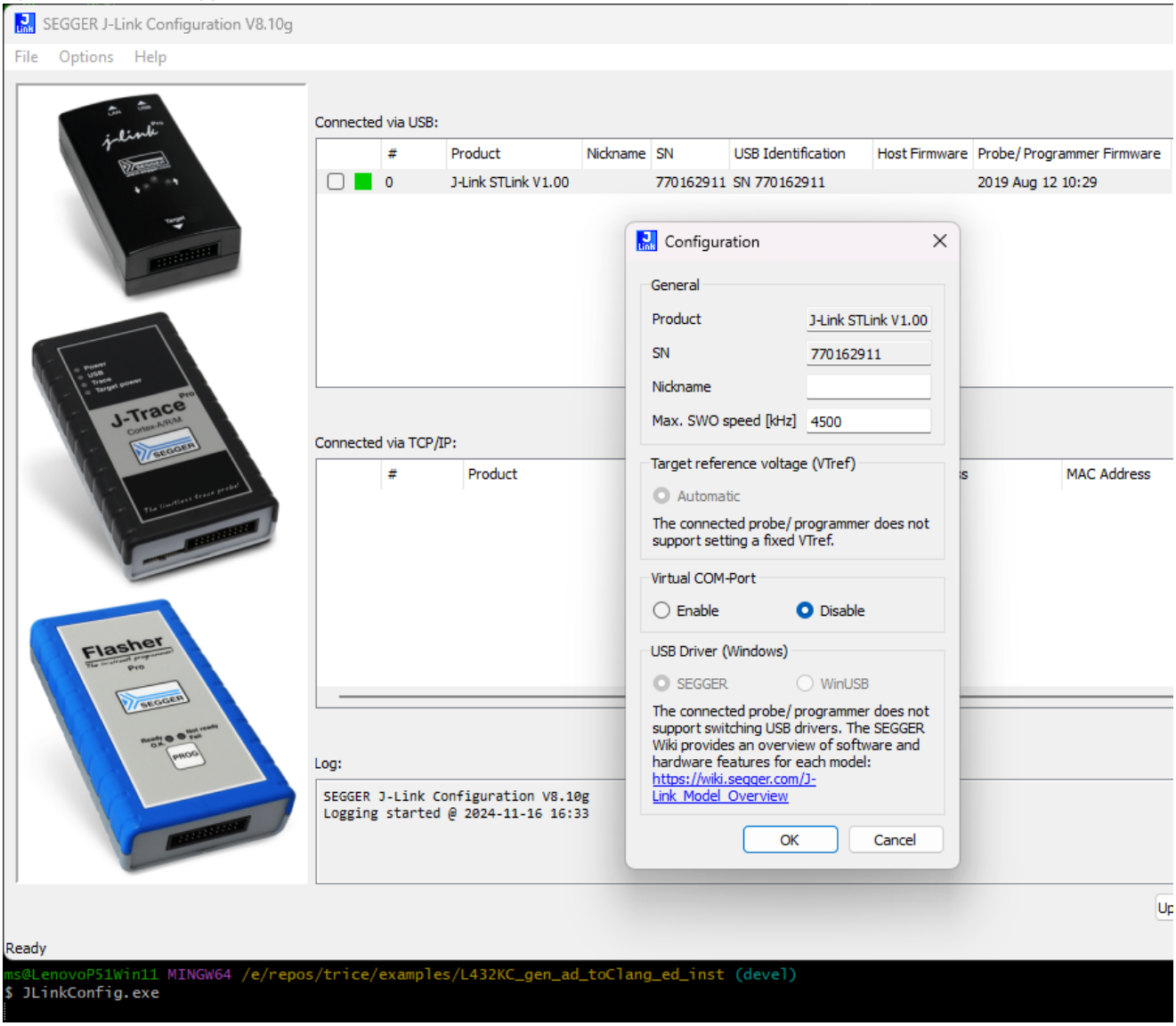
Using RTT with on-board J-Link and JLinkRTTLogger

- You need to install the "J-Link Software and Documentation pack" for yout OS.
- ./Core/Inc/triceConfig.h contains example Trice log commands.

Using RTT with on-board J-Link and OpenOCD

With Windows not possible

- OpenOCD does not support the installed JLink driver.



- Changing to the WinUSB buld device driver is here not supported 😟

Darwin (MacOS)

- See **OpenOCD with Darwin** in Trice over RTT

Using RTT with on-board ST-Link and OpenOCD

Terminal 1:

```
ms@LenovoP51Win11 MINGW64 /e/repos/trice/examples/L432_inst (devel)
$ openocd -f STLinkOpenOCD.cfg
Open On-Chip Debugger 0.12.0 (2024-09-16) [https://github.com/sysprogs/openocd]
```

```
Licensed under GNU GPL v2
libusb1 d52e355daa09f17ce64819122cb067b8a2ee0d4b
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ compared to
plain JTAG/SWD
Info : clock speed 100 kHz
Info : STLINK V2J24M11 (API v2) VID:PID 0483:374B
Info : Target voltage: 72.811768
Info : [stm32l4x.cpu] Cortex-M4 r0p1 processor detected
Info : [stm32l4x.cpu] target has 6 breakpoints, 4 watchpoints
Info : [stm32l4x.cpu] Examination succeed
Info : [stm32l4x.cpu] starting gdb server on 3333
Info : Listening on port 3333 for gdb connections
Info : rtt: Searching for control block 'SEGGER RTT'
Info : rtt: Control block found at 0x2000145c
Info : Listening on port 9090 for rtt connections
Channels: up=1, down=3
Up-channels:
0: Terminal 1024 0
Down-channels:
0: Terminal 16 0
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

Terminal2:

```
ms@LenovoP51Win11 MINGW64 /e/repos/trice/examples/L432_inst (devel)
$ trice l -p TCP4 -args localhost:9090  -pf none -d16
Nov 16 20:38:12.376056  TCP4:       triceExamples.c    10        1_595  Hello! 👌😊
Nov 16 20:38:12.376056  TCP4:
Nov 16 20:38:12.376056  TCP4:          ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨ ✨
Nov 16 20:38:12.376056  TCP4:          ♌ ♌ ♌ ♌  𝔑𝔘𝔈𝔏𝔈𝔒-L432KC    ♌ ♌ ♌ ♌
Nov 16 20:38:12.376056  TCP4:          🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃 🍃
Nov 16 20:38:12.376056  TCP4:
Nov 16 20:38:12.376056  TCP4:
Nov 16 20:38:13.891033  TCP4:       triceExamples.c    16       43_439 2.71828182845904523536 <- float
number as string
Nov 16 20:38:14.874024  TCP4:       triceExamples.c    17       44_949 2.71828182845904509080 (double with
more ciphers than precision)
Nov 16 20:38:15.692614  TCP4:       triceExamples.c    18       45_802 2.71828174591064453125 (float  with
more ciphers than precision)
Nov 16 20:38:16.323665  TCP4:       triceExamples.c    19       46_536 2.718282 (default rounded float)
```

Using On-board ST-Link and VS-Code Cortex-Debug Extension

Fail

- https://www.st.com/resource/en/user_manual/um2576-stm32cubeide-stlink-gdb-server-stmicroelectronics.pdf
- Downloaded and installed
  - en.stm32cubeprg-win64-v2-17-0.zip
  - en.st-link-server-v2-1-1.zip
    - PATH variable extended with `C:\Program Files (x86)\STMicroelectronics\stlink_server`
    - Copied
      - From: "C:\Program Files (x86)\STMicroelectronics\stlink_server\stlinkserver.exe"
      - To: "C:\Program Files (x86)\STMicroelectronics\stlink_server\ST-LINK_gdbserver.exe"

OK

- Download st-util from github.com
- Unpack to `C:\bin\stlink-1.8.0-win32` and add `C:\bin\stlink-1.8.0-win32\bin` to path
- Copy `C:\bin\stlink-1.8.0-win32\Program Files (x86)\stlink` to `C:\Program Files (x86)\stlink`

- Get `C:\bin\libusb-1.0.27`
- Copy `C:\bin\libusb-1.0.27\MinGW64\dll\libusb-1.0.dll` to `C:\bin\stlink-1.8.0-win32\bin\libusb-1.0.dll`

```
ms@LenovoP51Win11 MINGW64 /e/repos/trice/examples/L432_inst (devel)
$ st-util.exe
st-util 1.8.0
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_056A&PID_5105\5&1140C04&0&10'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_056A&PID_5105&MI_01\6&13339912&0&0001'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_058F&PID_9540\5&1140C04&0&11'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_8087&PID_0A2B\5&1140C04&0&14'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\ROOT_HUB30\4&20F1DF2E&0&0'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_0765&PID_5010\5&1140C04&0&13'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_0483&PID_374B&MI_01\6&224DEA1D&0&0001'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_5986&PID_111C&MI_00\6&104790C2&0&0000'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_046D&PID_C534\5&1140C04&0&6'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_0483&PID_374B&MI_02\6&224DEA1D&0&0002'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_0483&PID_374B\066CFF515570514867145144'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_138A&PID_0097\72FA8C531499'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_056A&PID_5105&MI_00\6&13339912&0&0000'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_046D&PID_C534&MI_01\6&C944391&0&0001'
libusb: info [get_guid] no DeviceInterfaceGUID registered for
'USB\VID_046D&PID_C534&MI_00\6&C944391&0&0000'
libusb: info [get_guid] no DeviceInterfaceGUID registered for 'USB\VID_5986&PID_111C\200901010001'
2024-11-17T22:20:05 INFO common.c: STM32L41x_L42x: 48 KiB SRAM, 256 KiB flash in at least 2 KiB pages.
2024-11-17T22:20:05 INFO gdb-server.c: Listening at *:4242...
Receive signal 0. Exiting...
```

(Last line after `CTRL-C`)

(back to top)

37. Trice Generate

37.1. Colors

Support for finding a color style:

See Check Alternatives chapter.

37.2. C-Code

If you intend to get the `trice log` functionality full or partially as a `tlog` C-Source and do not wish to parse the *til.json* file, you can run `trice generate -tilH --tilC` to create a C-file with header as starting point. That could be interesting for compiling the log functionality into a small separate microcontroller board.

```
//! \file til.c
//! ////////////////////////////////////////////////////////////////////////////

//! Trice generated code - do not edit!

#include "til.h"

//! triceFormatStringList contains all trice format strings together with id and parameter information.
//!
//! The bitWidth value is not transmitted in the binary data stream and needed for its decoding.
//! The paramCount is de-facto not needed. It is derivable from the received data, see
docs/TriceUserManual.md#binary-encoding.
//! It is recommended to check if both values are matching. A negative paramCount indicates, that its
value is unknown at compile time.
const triceFormatStringList_t triceFormatStringList[] = {
    /* Trice type (  extended  ) */  //  id, bitWidth, paramCount, format-string
    /*      trice (  trice32_9 ) */ { 14016,  32,  9, "rd:trice %d, %d, %d, %d, %d, %d, %d, %d, %d\n" },
    /*      trice (     trice0 ) */ { 14224,  32,  0, "\n" },
    /*    trice32 (  trice32_1 ) */ { 14337,  32,  1, "msg:%u (%%u)\n" },
    /*     TRICE8 (  TRICE8_10 ) */ { 15063,   8, 10, "rd:TRICE8 %d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n"
},
    /*     Trice8 (   Trice8_9 ) */ { 15124,   8,  9, "rd:Trice8 %d, %d, %d, %d, %d, %d, %d, %d, %d\n" },
    /*     TRICE8 (   TRICE8_5 ) */ { 15058,   8,  5, "rd:TRICE8 %d, %d, %d, %d, %d\n" },
    /*      TRice (     TRice0 ) */ { 14885,  32,  0, "TEST:yellow+h:black\n" },
    /*    Trice64 (  Trice64_1 ) */ { 15560,  64,  1, "rd:Trice64 %d\n" },
    /*      trice (  trice32_1 ) */ { 15860,  32,  1, "rd:TRICE float %9.f (%%9.f)\n" },
...
    /*  TRICE64_0 (  TRICE64_0 ) */ { 16157,  64,  0, "w: Hello! 👌😁 \a\n" },
    /*      TRICE (     TRICE0 ) */ { 14658,  32,  0, "interrupt:magenta+i:default+h\n" },
};

//! triceFormatStringListElements holds the compile time computed count of list elements.
const unsigned triceFormatStringListElements = sizeof(triceFormatStringList) /
```

```
sizeof(triceFormatStringList_t);
```

37.3. C#-Code

With `trice generate -tilCS` a starting point for a C-Sharp application is generated:

```
//! \file til.cs

// Trice generated code - do not edit!

// There is still a need to exchange the format specifier from C to C# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// See https://stackoverflow.com/questions/33432341/how-to-use-c-language-format-specifiers-in-c-sharp
// and https://www.codeproject.com/Articles/19274/A-printf-implementation-in-C for possible help.

namespace TriceIDList;

    public class TilItem
    {
        public TilItem(int bitWidth, int paramCount, string strg)
        {
            BitWidth = bitWidth;
            ParamCount = paramCount;
            Strg = strg;
        }

        public int BitWidth { get; init; }
        public int ParamCount { get; init; }
        public string Strg { get; init; }
    }

    //! Til contains all trice format strings together with id and parameter information.
    //!
    //! The bitWidth value is not transmitted in the binary data stream and needed for its decoding.
    //! The paramCount is de-facto not needed. It is derivable from the received data, see
docs/TriceUserManual.md#binary-encoding.
    //! It is recommended to check if both values are matching. A negative paramCount indicates, that its
value is unknown at compile time.
    public static class Til
    {
        public static readonly Dictionary<int, TilItem> TilList= new Dictionary<int, TilItem>
        { /* triceType ( extended ) */ //   id,     TilItem( bitWidth, paramCount, Strg )
        /*   TRICE_12 ( TRICE32_12 )*/ { 14991, new TilItem( 32, 12, "rd:TRICE_12 %d, %d, %d, %d, %d, %d,
%d, %d, %d, %d, %d, %d\n" ) },
        /*      TRICE (  TRICE32_1 )*/ { 15636, new TilItem( 32,  1, "WR:write        message, SysTick is
%6u\n" ) },
        /*    TRICE_S (    TRICE_S )*/ { 14178, new TilItem( 32, -1, "msg:With TRICE_S:%s\n" ) },
...
        /*    TRICE16 (  TRICE16_2 )*/ { 16056, new TilItem( 16,  2, "rd:TRICE16 %p, %p\n" ) },
    };
}
```

37.4. Generating a RPC Function Pointer List

When several embedded devices are going to communicate, `trice generate -rpcH -rpcC` could be helpful.

You will get 2 files similar to:

```
//! \file tilRpc.h
//! ////////////////////////////////////////////////////////////////////////

//! Trice generated code - do not edit!
```

```
#include <stdint.h>

typedef void (*triceRpcHandler_t)(void* buffer, int count);

typedef struct{
    int id;
    triceRpcHandler_t fn;
} triceRpc_t;

extern triceRpc_t triceRpc[];
extern int triceRpcCount;

/*  TRICE16_F */ void FunctionNameXa( int16_t* p, int cnt );
/*  TRICE32_F */ void FunctionNameYa( int32_t* p, int cnt );
/*   TRICE8_F */ void TryoutBufferFunction( int8_t* p, int cnt );
/*   TRice16F */ void FunctionNameXb( int16_t* p, int cnt );
/*   trice32F */ void FunctionNameYd( int32_t* p, int cnt );
/*    TRice8F */ void FunctionNameWb( int8_t* p, int cnt );
/*   trice64F */ void FunctionNameZd( int64_t* p, int cnt );
/*   Trice16F */ void ARemoteFunctionName( int16_t* p, int cnt );
/*    Trice8F */ void FunctionNameWc( int8_t* p, int cnt );
/*   Trice16F */ void FunctionNameXc( int16_t* p, int cnt );
/*   TRICE8_F */ void TryoutStructFunction( int8_t* p, int cnt );
/*   Trice64F */ void FunctionNameZc( int64_t* p, int cnt );
/*    trice8F */ void FunctionNameWd( int8_t* p, int cnt );
/*   TRice64F */ void FunctionNameZb( int64_t* p, int cnt );
/*  TRICE64_F */ void FunctionNameZa( int64_t* p, int cnt );
/*   trice16F */ void FunctionNameXd( int16_t* p, int cnt );
/*   TRICE8_F */ void FunctionNameWa( int8_t* p, int cnt );
/*   TRice32F */ void FunctionNameYb( int32_t* p, int cnt );
/*   Trice32F */ void FunctionNameYc( int32_t* p, int cnt );

// End of file
```

```
//! \file tilRpc.c
//! ////////////////////////////////////////////////////////////////////////////

//! Trice generated code - do not edit!

#include <stdio.h> // needed for __attribute__((weak))
#include "tilRpc.h"

//! triceRpc contains all rpc IDs together with their function pointer address.
const triceRpc_t triceRpc[] = {
    /* Trice type */  //  id, function pointer
    /*    TRice8F */ { 14227, FunctionNameWb },
    /*  TRICE32_F */ { 14234, FunctionNameYa },
    /*   TRICE8_F */ { 16179, TryoutBufferFunction },
    /*   Trice16F */ { 14232, FunctionNameXc },
    /*   Trice64F */ { 14240, FunctionNameZc },
    /*   TRice64F */ { 14239, FunctionNameZb },
    /*  TRICE16_F */ { 14230, FunctionNameXa },
    /*   TRICE8_F */ { 16178, TryoutStructFunction },
    /*    Trice8F */ { 14228, FunctionNameWc },
    /*   trice16F */ { 14233, FunctionNameXd },
    /*   trice64F */ { 14241, FunctionNameZd },
    /*   trice32F */ { 14237, FunctionNameYd },
    /*   TRICE8_F */ { 14226, FunctionNameWa },
    /*   TRice16F */ { 14231, FunctionNameXb },
    /*   TRice32F */ { 14235, FunctionNameYb },
    /*   Trice16F */ { 16337, ARemoteFunctionName },
    /*    trice8F */ { 14229, FunctionNameWd },
    /*   Trice32F */ { 14236, FunctionNameYc },
    /*  TRICE64_F */ { 14238, FunctionNameZa }
};
```

```
//! triceRpcListElements holds the compile time computed count of list elements.
const unsigned triceRpcElements = sizeof(triceRpc) / sizeof(triceRpc_t);

void TryoutBufferFunction( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameXc( int16_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameZc( int64_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameZb( int64_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameXa( int16_t* p, int cnt) __attribute__((weak)) {}
void TryoutStructFunction( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameWc( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameXd( int16_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameZd( int64_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameYd( int32_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameWa( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameXb( int16_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameYb( int32_t* p, int cnt) __attribute__((weak)) {}
void ARemoteFunctionName( int16_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameWd( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameYc( int32_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameZa( int64_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameWb( int8_t* p, int cnt) __attribute__((weak)) {}
void FunctionNameYa( int32_t* p, int cnt) __attribute__((weak)) {}

// End of file
```

Assume a project with several devices. You can add these 2 files to all targets and if a special target should execute any functions, simply implement them. These functions on their own can execute other Trice statements to transmit results. If a client executes a RPC function this way, the request is transmitted with the Trice speed. Several target devices (servers) can receive and respond and the client can wait for the first or some of them. That server receiving and client waiting functionality is not part of the Trice library.

(back to top)

38. Testing the Trice Library C-Code for the Target

38.1. General info

This folder is per default named to `_test` to avoid vsCode slow down. Also, when running `go test ./...`, the tests in the `_test` folder are excluded, because they take a long time. Run `./testAll.sh` to include them.

The main aim of these tests is to automatic compile and run the target code in different compiler switch variants avoiding manual testing this way.

`testAll.sh quick` performs just a short test. `testAll.sh full` runs all tests. That can take hours on a Windows PC because make is executed only as single thread there for stability reasons. On Darwin and Linux systems about an hour test duration should be expected.

- Partial tests:
  - In `./examples` you can translate all examples with `./buildAllTargets.sh`.
  - In `./examples/L432_inst` the script `all_configs_build.sh` translates many different configurations.

For the user it could be helpful to start with a `triceConfig.h`file from here and to adapt the Trice tool command line from the matching `cgo_test.go` if no close match in the `examples` folder was found.

38.2. How to run the tests

- In `_trice` folder first execute `go clean -cache` after editing C-files. Cleaning the **Go** cache is recommended, because the CGO tests keep pre-compiled files and when editing C-files, this can led to confusing results.
- Execute `./renewIDs_in_examples_and_test_folder.sh` after you edited files in the `./examples` or `_test` folder.
- To run the tests manually `cd` into `_test` and execute `trice insert -i ../demoTIL.json -li ../demoLI.json` and then `go test ./...` fom there. It is more handy to run `trice_insertIDs_in_examples_and_test_folder.sh` from the Trice root folder.
- It is convenient to run `testAll.sh` from the Trice root folder to perform this.
- It is possible to start the tests individually, but for some the default `-timeout 30s` maybe too short.

38.3. Tests Details

All folders despite `testdata` are test folders and the name `tf` is used as a place holder for them in this document.

To exclude a specific folder temporary, simply rename it to start with an underscore `_tf`.

The `tf` are serving for target code testing in different configuration variants on the host machine. The file ./testdata/triceCheck.c is the main file for most tests and serves also as example usage.

_test/testdata/cgoPackage.go is the common main for the `generated_cgoPackage.go` files and contains the common test code.

The folders `tf` are Go packages just for tests. They all have the same package name `cgot` and are not included into the trice tool. The different `cgot` packages are independent and could have any names. They do not see each other and are used for target code testing independently. When the tests are executed for each package, a separate test binary is build and these run parallel.

The `tf/triceConfig.h` files differ and correspondent to the `tf/cgo_test.go` files in the same folder. On test execution, the `./testdata/*.c` files are compiled into the trice test executable together with the trice sources `../src` using the `tf/triceConfig.h` file.

The individual tests collect the expected results (`//exp: result`) together with the line numbers into a slice to execute the test loop on it. The `triceLogTest` function gets the `triceLog` function as parameter.

`triceLogTest` iterates over the results slice and calls for each line the C-function `triceCheck`. Then the line specific binary data buffer is passed to the `triceLog` parameter function which "logs" the passed buffer into an actual result string which in turn is compared with the expected result.

The whole process is relatively slow because of the often passed Go - C barrier, but allows automated tests in different configuration variants in one shot.

The `testdata\cgoPackage.go` file contains a variable `testLines = n`, which limits the amount of performed trices for each test case to `n`. Changing this value will heavily influence the test duration. The value `-1` is reserved for testing all test lines.

38.4. How to add new test cases

- Choose a test folder similar to the intended test and copy it under a new descriptive name like `newTest`.
- Extend file `./renewIDs_in_examples_and_test_folder.sh` accordingly.
- Edit files `newTest/triceConfig.h` and `newTest/cgo_test.go` in a matching way.
- Run command `go test test/newTest/...`

38.5. Test Internals

The `./trice/_test/testdata/*.c` and `./trice/src/*.c` are compiled together with the actual cgot package into one singe Trice test binary, resulting in as many test binaries as there are test folders. Calling its TestFunction(s) causes the activation of the Trice statement(s) inside *triceCheck.c*. The ususally into an embedded device compiled Trice code generates a few bytes according to the configuration into a buffer. These bytes are transmitted usually in real life over a (serial) port or RTT. In the tests here, this buffer is then read out by the Trice tool handler function according to the used CLI switches and processed to a log string using the *til.json* file. This string is then compared to the expected string for the activated line.

Each `tf` is a **Go** package, which is not part of any **Go** application. They all named `cgot` and are only used independently for testing different configurations. The `tf/generated_cgoPackage.go` file is identical in all `tf`. Its master is `testdata/cgoPackage.go`. After editing the master, running the command `./renewIDs_in_examples_and_test_folder.sh` copies the master to all `tf` and renames it to `generated_cgoPackage.go`.

The test specific target code configuration is inside `tf/trice.Config.h` and the appropriate Trice tool CLI switches are in `tf/cgo_test.go`.

When running `go test ./tf`, a Trice tool test executable is build, using the Trice tool packages and the `tf` package `cgot`, and the function `TestLogs` is executed. Its internal closure `triceLog` contains the Trice tool CLI switches and is passed to the `ccgot` package function `triceLogTest` together with the number of testLines and the trice mode (`directTransfer` or `deferrerdTransfer`).

During the test, the file `triceCheck.c` is scanned for lines like

```
break; case __LINE__: TRice( iD(3537), "info:This is a message without values and a 32-bit stamp.\n" );
//exp: time: 842,150_450default: info:This is a message without values and a 32-bit stamp.
```

Some C-code lines contain Trice statements and comments starting with `//exp:` followed by the expected Trice tool output for that specific line. The **Go** testfunction collects these outputs in a slice together with the line numbers. Then for each found line number the execution of

the **Go** function `func triceCheck(n int)` takes part, which in turn calls the CGO compiled C-function `TriceCheck(n)`. The now activated Trice C-code writes the generated trice bytes in a between **C** and **Go** shared buffer using the C-function `TriceWriteDeviceCgo`. After returning from the **Go** function `func triceCheck(n int)` and optionally calling `TriceTransfer` in deferred mode the Trice tool `triceLog()` function converts the Trice buffer bytes to the log string and compares the result with the expected data. The between **Go** and **C** shared buffer limits the executed Trices per line to one, because they use the same buffer from the beginning. This could be done better with an increment to allow several trices in one single line.

Because each test runs a different configuration, all possible combinations are testable.

38.6. Test Results

```
ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice (main)
$ ./testAll.sh
Thu, Dec 12, 2024  4:51:26 PM
This can take several minutes ...
?       github.com/rokath/trice/internal/decoder        [no test files]
?       github.com/rokath/trice/internal/do     [no test files]
?       github.com/rokath/trice/internal/translator     [no test files]
?       github.com/rokath/trice/pkg/ant [no test files]
ok      github.com/rokath/trice/cmd/trice       1.392s
ok      github.com/rokath/trice/internal/args   0.415s
ok      github.com/rokath/trice/internal/charDecoder    0.298s
ok      github.com/rokath/trice/internal/com    15.845s
ok      github.com/rokath/trice/internal/dumpDecoder    0.339s
ok      github.com/rokath/trice/internal/emitter        0.326s
ok      github.com/rokath/trice/internal/id     3.088s
ok      github.com/rokath/trice/internal/keybcmd        0.233s
ok      github.com/rokath/trice/internal/link   0.196s
ok      github.com/rokath/trice/internal/receiver       0.246s
?       github.com/rokath/trice/internal/translator     [no test files]
?       github.com/rokath/trice/pkg/ant [no test files]
ok      github.com/rokath/trice/cmd/trice       1.392s
ok      github.com/rokath/trice/internal/args   0.415s
ok      github.com/rokath/trice/internal/charDecoder    0.298s
ok      github.com/rokath/trice/internal/com    15.845s
ok      github.com/rokath/trice/internal/dumpDecoder    0.339s
ok      github.com/rokath/trice/internal/emitter        0.326s
ok      github.com/rokath/trice/internal/id     3.088s
ok      github.com/rokath/trice/internal/keybcmd        0.233s
ok      github.com/rokath/trice/internal/link   0.196s
ok      github.com/rokath/trice/internal/receiver       0.246s
ok      github.com/rokath/trice/internal/trexDecoder    0.264s
ok      github.com/rokath/trice/pkg/cipher      0.230s
ok      github.com/rokath/trice/pkg/endian      0.161s
ok      github.com/rokath/trice/internal/args   0.415s
ok      github.com/rokath/trice/internal/charDecoder    0.298s
ok      github.com/rokath/trice/internal/com    15.845s
ok      github.com/rokath/trice/internal/dumpDecoder    0.339s
ok      github.com/rokath/trice/internal/emitter        0.326s
ok      github.com/rokath/trice/internal/id     3.088s
ok      github.com/rokath/trice/internal/keybcmd        0.233s
ok      github.com/rokath/trice/internal/link   0.196s
ok      github.com/rokath/trice/internal/receiver       0.246s
ok      github.com/rokath/trice/internal/trexDecoder    0.264s
ok      github.com/rokath/trice/pkg/cipher      0.230s
ok      github.com/rokath/trice/pkg/endian      0.161s
ok      github.com/rokath/trice/internal/id     3.088s
ok      github.com/rokath/trice/internal/keybcmd        0.233s
ok      github.com/rokath/trice/internal/link   0.196s
ok      github.com/rokath/trice/internal/receiver       0.246s
ok      github.com/rokath/trice/internal/trexDecoder    0.264s
ok      github.com/rokath/trice/pkg/cipher      0.230s
ok      github.com/rokath/trice/pkg/endian      0.161s
ok      github.com/rokath/trice/pkg/msg 0.157s
ok      github.com/rokath/trice/pkg/tst 0.261s
ok      github.com/rokath/trice/_test/be_dblB_de_tcobs_ua       123.142s
```

```
ok      github.com/rokath/trice/_test/be_staticB_di_xtea_cobs_rtt32     123.159s
ok      github.com/rokath/trice/internal/trexDecoder    0.264s
ok      github.com/rokath/trice/pkg/cipher      0.230s
ok      github.com/rokath/trice/pkg/endian      0.161s
ok      github.com/rokath/trice/pkg/msg 0.157s
ok      github.com/rokath/trice/pkg/tst 0.261s
ok      github.com/rokath/trice/_test/be_dblB_de_tcobs_ua       123.142s
ok      github.com/rokath/trice/_test/be_staticB_di_xtea_cobs_rtt32     123.159s
ok      github.com/rokath/trice/pkg/msg 0.157s
ok      github.com/rokath/trice/pkg/tst 0.261s
ok      github.com/rokath/trice/_test/be_dblB_de_tcobs_ua       123.142s
ok      github.com/rokath/trice/_test/be_staticB_di_xtea_cobs_rtt32     123.159s
ok      github.com/rokath/trice/_test/dblB_de_cobs_ua   122.964s
ok      github.com/rokath/trice/_test/dblB_de_multi_cobs_ua     123.308s
ok      github.com/rokath/trice/_test/be_dblB_de_tcobs_ua       123.142s
ok      github.com/rokath/trice/_test/be_staticB_di_xtea_cobs_rtt32     123.159s
ok      github.com/rokath/trice/_test/dblB_de_cobs_ua   122.964s
ok      github.com/rokath/trice/_test/dblB_de_multi_cobs_ua     123.308s
ok      github.com/rokath/trice/_test/dblB_de_cobs_ua   122.964s
ok      github.com/rokath/trice/_test/dblB_de_multi_cobs_ua     123.308s
ok      github.com/rokath/trice/_test/dblB_de_multi_cobs_ua     123.308s
ok      github.com/rokath/trice/_test/dblB_de_multi_nopf_ua     123.244s
ok      github.com/rokath/trice/_test/dblB_de_multi_nopf_ua     123.244s
ok      github.com/rokath/trice/_test/dblB_de_multi_tcobs_ua    123.109s
ok      github.com/rokath/trice/_test/dblB_de_multi_tcobs_ua    123.109s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_cobs_ua        123.213s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_tcobs_ua       123.001s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_cobs_ua        123.213s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_tcobs_ua       123.001s
ok      github.com/rokath/trice/_test/dblB_de_nopf_ua   123.092s
ok      github.com/rokath/trice/_test/dblB_de_multi_xtea_tcobs_ua       123.001s
ok      github.com/rokath/trice/_test/dblB_de_nopf_ua   123.092s
ok      github.com/rokath/trice/_test/dblB_de_tcobs_ua  122.324s
ok      github.com/rokath/trice/_test/dblB_de_nopf_ua   123.092s
ok      github.com/rokath/trice/_test/dblB_de_tcobs_ua  122.324s
ok      github.com/rokath/trice/_test/dblB_de_tcobs_ua  122.324s
ok      github.com/rokath/trice/_test/dblB_de_xtea_cobs_ua      123.149s
ok      github.com/rokath/trice/_test/dblB_de_xtea_tcobs_ua     122.883s
ok      github.com/rokath/trice/_test/dblB_de_xtea_cobs_ua      123.149s
ok      github.com/rokath/trice/_test/dblB_de_xtea_tcobs_ua     122.883s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_cobs_ua    246.703s
ok      github.com/rokath/trice/_test/dblB_de_xtea_tcobs_ua     122.883s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_cobs_ua    246.703s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_cobs_ua    246.703s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_multi_cobs_ua      247.125s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_multi_tcobs_ua     246.862s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_tcobs_ua   246.531s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt32__de_xtea_cobs_ua       247.072s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_cobs_ua     246.639s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_multi_cobs_ua       246.599s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_multi_tcobs_ua      247.114s
ok      github.com/rokath/trice/_test/dblB_di_nopf_rtt8__de_tcobs_ua    246.851s
ok      github.com/rokath/trice/_test/ringB_de_cobs_ua  123.578s
ok      github.com/rokath/trice/_test/ringB_de_multi_tcobs_ua   123.517s
ok      github.com/rokath/trice/_test/ringB_de_multi_xtea_cobs_ua       123.497s
ok      github.com/rokath/trice/_test/ringB_de_multi_xtea_tcobs_ua      123.379s
ok      github.com/rokath/trice/_test/ringB_de_nopf_ua  123.555s
ok      github.com/rokath/trice/_test/ringB_de_tcobs_ua 123.300s
ok      github.com/rokath/trice/_test/ringB_de_xtea_cobs_ua     123.487s
ok      github.com/rokath/trice/_test/ringB_de_xtea_tcobs_ua    123.846s
ok      github.com/rokath/trice/_test/ringB_di_cobs_rtt32__de_tcobs_ua  247.400s
ok      github.com/rokath/trice/_test/ringB_di_cobs_rtt8__de_tcobs_ua   247.202s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt32__de_tcobs_ua  247.204s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt32__de_xtea_cobs_ua      246.818s
ok      github.com/rokath/trice/_test/ringB_di_nopf_rtt8__de_tcobs_ua   247.006s
ok      github.com/rokath/trice/_test/ringB_di_tcobs_rtt32__de_tcobs_ua 247.000s
ok      github.com/rokath/trice/_test/ringB_di_xtea_cobs_rtt32__de_xtea_cobs_ua 246.872s
ok      github.com/rokath/trice/_test/special_protect_dblB_de_tcobs_ua  0.444s
```

```
ok      github.com/rokath/trice/_test/stackB_di_nopf_aux32      123.819s
ok      github.com/rokath/trice/_test/stackB_di_nopf_aux8       123.830s
ok      github.com/rokath/trice/_test/stackB_di_nopf_rtt32      123.912s
ok      github.com/rokath/trice/_test/stackB_di_nopf_rtt8       123.976s
ok      github.com/rokath/trice/_test/stackB_di_xtea_cobs_rtt8  123.719s
ok      github.com/rokath/trice/_test/staticB_di_nopf_aux32     123.553s
ok      github.com/rokath/trice/_test/staticB_di_nopf_aux8      123.551s
ok      github.com/rokath/trice/_test/staticB_di_nopf_rtt32     123.596s
ok      github.com/rokath/trice/_test/staticB_di_nopf_rtt8      123.618s
ok      github.com/rokath/trice/_test/staticB_di_tcobs_rtt32    123.177s
ok      github.com/rokath/trice/_test/staticB_di_tcobs_rtt8     123.353s
ok      github.com/rokath/trice/_test/staticB_di_xtea_cobs_rtt32        123.126s

real    10m31.130s
user    0m0.000s
sys     0m0.015s

ms@DESKTOP-7POEGPB MINGW64 ~/repos/trice (main)
$
```

38.7. Special tests

38.8. Test Cases

38.8.1. Folder Naming Convention

Folder Name Part	Meaning
`testdata`	This is no test folder. It contains data common to all tests.
`_...`	Folder starting with an undescore _ are excluded when `go test ./...` is executed.
`_di_`	direct mode
`_de_`	deferred mode
`special_`	a test, not using `./testdata/triceCheck.c`
`staticB_`	static buffer, direct mode only possible
`stackB_`	stack buffer, direct mode only possible
`ringB_`	ring buffer, deferred mode and optional parallel direct mode
`dblB_`	double buffer, deferred mode and optional parallel direct mode
`_rtt8_`	(simulated) SEGGER_RTT byte transfer
`_rtt32_`	(simulated) SEGGER_RTT word transfer
`__`	direct and deferred mode together
`_xtea_`	with encryption, otherwise without encryption
`_tcobs`	TCOBS package framing
`_cobs`	COBS package framing
`_nopf`	no package framing
`_multi_`	Usually each Trice is handled separately. In multi mode, groups of available Trices are framed together.
`_ua`	simulated UART A output (for deferred modes)

(back to top)

39. Test Issues

Test folders starting with `ERROR_` have issues. These cases are **usable** on the target. These tests fail for an unknown reason. Probably it is a test implementation issue. Especially when XTEA is used in one output but not in the other, the tests fail.

(back to top)

40. Add-On Hints

40.1. Trice on LibOpenCM3

- This is a OpenCM3_STM32F411_Nucleo Contribution from kraiskil.
- See also pull request #269.
- It is here because the code need some re-work to be compatible with Trice version 1.0.

LibOpenCM3 is a hardware abstraction library for many microcontrollers.

This is an exampe using STM's STM32F411 Nucleo board.

```
--> This code uses a legacy Trice version and needs adaption!
```

40.1.1. Prerequisites

- Suitable ARM GCC cross compiler (`arm-none-eabi-gcc`) found in your system's PATH
- GNU Make, or compatible
- Environment variable `OPENCM3_DIR` points to the base install of libopencm3. This is e.g. the libopencm3 source directory, if you also built it in the source directory.
- OpenOCD

40.1.2. triceConfig.h

```
/*! \file triceConfig.h
\author Thomas.Hoehenleitner [at] seerose.net
LibOpenCM3 adapatation by Kalle Raiskila.
*******************************************************************************/

#ifndef TRICE_CONFIG_H_
#define TRICE_CONFIG_H_

#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <libopencm3/cm3/cortex.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/usart.h>

// Local (to this demo) time keeping functions
#include "time.h"

#define TRICE_UART USART2 //!< Enable and set UART for serial output.
// The alternative, TRICE_RTT_CHANNEL is not available with OpenCM3
// #define TRICE_RTT_CHANNEL 0

// Timestamping function to be provided by user. In this demo from time.h
#define TRICE_TIMESTAMP wallclock_ms() // todo: replace with TRICE_TREX_ENCODING stuff

// Enabling next 2 lines results in XTEA TriceEncryption  with the key.
// #define TRICE_ENCRYPT XTEA_KEY( ea, bb, ec, 6f, 31, 80, 4e, b9, 68, e2, fa, ea, ae, f1, 50, 54 ); //!<
-password MySecret
// #define TRICE_DECRYPT //!< TRICE_DECRYPT is usually not needed. Enable for checks.

// #define TRICE_BIG_ENDIANNESS //!< TRICE_BIG_ENDIANNESS needs to be defined for TRICE64 macros on big
endian devices. (Untested!)

//
////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
// Predefined trice modes: Adapt or creeate your own trice mode.
//
#ifndef TRICE_MODE
#error Define TRICE_MODE to 0, 200 or 201
#endif

//! Direct output to UART or RTT with cycle counter. Trices inside interrupts forbidden. Direct TRICE
macro execution.
//! This mode is mainly for a quick tryout start or if no timing constrains for the TRICE macros exist.
//! Only a putchar() function is required - look for triceBlockingPutChar().
//! UART Command line similar to: `trice log -p COM1 -baud 115200`
//! RTT needs additional tools installed - see RTT documentation.
//! J-LINK Command line similar to: `trice log -args="-Device STM32G071RB -if SWD -Speed 4000 -RTTChannel
0 -RTTSearchRanges 0x20000000_0x1000"`
//! ST-LINK Command line similar to: `trice log -p ST-LINK -args="-Device STM32G071RB -if SWD -Speed 4000
-RTTChannel 0 -RTTSearchRanges 0x20000000_0x1000"`
#if TRICE_MODE == 0                     // must not use TRICE_ENCRYPT!
#define TRICE_STACK_BUFFER_MAX_SIZE 128 //!< This  minus TRICE_DATA_OFFSET the max allowed single trice
size. Usually ~40 is enough.
#ifndef TRICE_ENTER
#define TRICE_ENTER                                                                         \
    {                                                 /*! Start of TRICE macro */           \
        uint32_t co[TRICE_STACK_BUFFER_MAX_SIZE >> 2]; /* Check TriceDepthMax at runtime. */ \
        uint32_t* TriceBufferWritePosition = co + (TRICE_DATA_OFFSET >> 2);
#endif
#ifndef TRICE_LEAVE
#define TRICE_LEAVE                                                                 \
    { /*! End of TRICE macro */                                                    \
        unsigned tLen = ((TriceBufferWritePosition - co) << 2) - TRICE_DATA_OFFSET; \
        TriceOut(co, tLen);                                                         \
    }                                                                               \
    }
#endif
#endif // #if TRICE_MODE == 0

//! Double Buffering output to RTT or UART with cycle counter. Trices inside interrupts allowed. Fast
TRICE macro execution.
//! UART Command line similar to: `trice log -p COM1 -baud 115200`
//! RTT Command line similar to: `trice l -args="-Device STM32F030R8 -if SWD -Speed 4000 -RTTChannel 0 -
RTTSearchRanges 0x20000000_0x1000"`
#if TRICE_MODE == 200
#ifndef TRICE_ENTER
#define TRICE_ENTER TRICE_ENTER_CRITICAL_SECTION //! TRICE_ENTER is the start of TRICE macro. The TRICE
macros are a bit slower. Inside interrupts TRICE macros allowed.
#endif
#ifndef TRICE_LEAVE
#define TRICE_LEAVE TRICE_LEAVE_CRITICAL_SECTION //! TRICE_LEAVE is the end of TRICE macro.
#endif
#define TRICE_HALF_BUFFER_SIZE 1000 //!< This is the size of each of both buffers. Must be able to hold
the max TRICE burst count within TRICE_TRANSFER_INTERVAL_MS or even more, if the write out speed is small.
Must not exceed SEGGER BUFFER_SIZE_UP
#define TRICE_SINGLE_MAX_SIZE 100   //!< must not exeed TRICE_HALF_BUFFER_SIZE!
#endif                              // #if TRICE_MODE == 200

//! Double Buffering output to UART without cycle counter. No trices inside interrupts allowed. Fastest
TRICE macro execution.
//! Command line similar to: `trice log -p COM1 -baud 115200`
#if TRICE_MODE == 201
#define TRICE_CYCLE_COUNTER 0       //! Do not add cycle counter, The TRICE macros are a bit faster. Lost
TRICEs are not detectable by the trice tool.
#define TRICE_ENTER                 //! TRICE_ENTER is the start of TRICE macro. The TRICE macros are a
bit faster. Inside interrupts TRICE macros forbidden.
#define TRICE_LEAVE                 //! TRICE_LEAVE is the end of TRICE macro.
#define TRICE_HALF_BUFFER_SIZE 2000 //!< This is the size of each of both buffers. Must be able to hold
the max TRICE burst count within TRICE_TRANSFER_INTERVAL_MS or even more, if the write out speed is small.
Must not exceed SEGGER BUFFER_SIZE_UP
```

```
#define TRICE_SINGLE_MAX_SIZE 800   //!< must not exeed TRICE_HALF_BUFFER_SIZE!
#endif                              // #if TRICE_MODE == 201

//
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Headline info
//

#ifdef TRICE_HALF_BUFFER_SIZE
#define TRICE_BUFFER_INFO                                                   \
    do {                                                                    \
        TRICE32(Id(0), "att: Trice 2x half buffer size:%4u ", TRICE_HALF_BUFFER_SIZE); \
    } while (0)
#else
#define TRICE_BUFFER_INFO
\
    do {
\
        TRICE32(Id(0), "att:Single Trice Stack buf size:%4u", TRICE_SINGLE_MAX_SIZE + TRICE_DATA_OFFSET);
\
    } while (0)
#endif

//! This is usable as the very first trice sequence after restart. Adapt and use it or ignore it.
#define TRICE_HEADLINE                                                  \
    TRICE0(Id(0), "s:                                       \n");       \
    TRICE8(Id(0), "s:     NUCLEO-F411RE     TRICE_MODE %3u  \n", TRICE_MODE); \
    TRICE0(Id(0), "s:                                       \n");       \
    TRICE0(Id(0), "s:     ");                                           \
    TRICE_BUFFER_INFO;                                                  \
    TRICE0(Id(0), "s:     \n");                                         \
    TRICE0(Id(0), "s:                                       \n");

//
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Compiler Adaption
//

#if defined(__GNUC__) /* gnu compiler ######################################## */

#define TRICE_INLINE static inline //! used for trice code

#define ALIGN4                                  //!< align to 4 byte boundary preamble
#define ALIGN4_END __attribute__((aligned(4))) //!< align to 4 byte boundary post declaration

//! TRICE_ENTER_CRITICAL_SECTION saves interrupt state and disables Interrupts.
#define TRICE_ENTER_CRITICAL_SECTION              \
    {                                             \
        uint32_t old_mask = cm_mask_interrupts(1); \
        {

//! TRICE_LEAVE_CRITICAL_SECTION restores interrupt state.
#define TRICE_LEAVE_CRITICAL_SECTION \
    }                                \
    cm_mask_interrupts(old_mask);    \
    }

#else
#error unknown compliler
#endif // compiler adaptions ###################################################

//
////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
// Optical feedback: Adapt to your device.
//

TRICE_INLINE void ToggleOpticalFeedbackLED(void) {
    // The only user controllable LED available on the
    // Nucleo is LD2, on port A5. This is set up in main.c
    gpio_toggle(GPIOA, GPIO5);
}

//
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// UART interface: Adapt to your device.
//

#ifdef TRICE_UART

//! Check if a new byte can be written into trice transmit register.
//! \retval 0 == not empty
//! \retval !0 == empty
//! User must provide this function.
TRICE_INLINE uint32_t triceTxDataRegisterEmpty(void) {
    uint32_t reg = USART_SR(TRICE_UART);
    return (reg & USART_SR_TXE);
}

//! Write value v into trice transmit register.
//! \param v byte to transmit
//! User must provide this function.
TRICE_INLINE void triceTransmitData8(uint8_t v) {
    usart_send_blocking(TRICE_UART, v);
    ToggleOpticalFeedbackLED();
}

//! Allow interrupt for empty trice data transmit register.
//! User must provide this function.
TRICE_INLINE void triceEnableTxEmptyInterrupt(void) {
    usart_enable_tx_interrupt(TRICE_UART);
}

//! Disallow interrupt for empty trice data transmit register.
//! User must provide this function.
TRICE_INLINE void triceDisableTxEmptyInterrupt(void) {
    usart_disable_tx_interrupt(TRICE_UART);
}

#endif // #ifdef TRICE_UART

////////////////////////////////////////////////////////////////////////////
// Default TRICE macro bitwidth: 32 (optionally adapt to MCU bit width)
//

#define TRICE_1 TRICE32_1   //!< Default parameter bit width for 1  parameter count TRICE is 32, change
for a different value.
#define TRICE_2 TRICE32_2   //!< Default parameter bit width for 2  parameter count TRICE is 32, change
for a different value.
#define TRICE_3 TRICE32_3   //!< Default parameter bit width for 3  parameter count TRICE is 32, change
for a different value.
#define TRICE_4 TRICE32_4   //!< Default parameter bit width for 4  parameter count TRICE is 32, change
for a different value.
#define TRICE_5 TRICE32_5   //!< Default parameter bit width for 5  parameter count TRICE is 32, change
for a different value.
#define TRICE_6 TRICE32_6   //!< Default parameter bit width for 6  parameter count TRICE is 32, change
for a different value.
#define TRICE_7 TRICE32_7   //!< Default parameter bit width for 7  parameter count TRICE is 32, change
for a different value.
```

```
#define TRICE_8 TRICE32_8   //!< Default parameter bit width for 8  parameter count TRICE is 32, change
for a different value.
#define TRICE_9 TRICE32_9   //!< Default parameter bit width for 9  parameter count TRICE is 32, change
for a different value.
#define TRICE_10 TRICE32_10 //!< Default parameter bit width for 10 parameter count TRICE is 32, change
for a different value.
#define TRICE_11 TRICE32_11 //!< Default parameter bit width for 11 parameter count TRICE is 32, change
for a different value.
#define TRICE_12 TRICE32_12 //!< Default parameter bit width for 12 parameter count TRICE is 32, change
for a different value.

//
////////////////////////////////////////////////////////////////////////////

#ifdef __cplusplus
}
#endif

#endif /* TRICE_CONFIG_H_ */
```

40.1.3. main.c

```
/*
 * Demo to test/show TRICE usage in a libopencm3
 * environment.
 */

#include <libopencm3/cm3/systick.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/exti.h>
#include <libopencm3/stm32/usart.h>
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/cm3/nvic.h>

#include <stdint.h>
void msleep(uint32_t delay);
uint32_t wallclock_ms(void);

#include "trice.h"

static void hardware_setup(void)
{
    /* Set device clocks from opencm3 provided preset.*/
    const struct rcc_clock_scale *clocks = &rcc_hsi_configs[RCC_CLOCK_3V3_84MHZ];
    rcc_clock_setup_pll( clocks );

    /* Set up driving the LED connected to port A, pin 5. */
    rcc_periph_clock_enable(RCC_GPIOA);
    gpio_mode_setup(GPIOA, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO5);

    /* User-button is connected to port C, pin 13. Set up button push
     * to cause an interrupt. */
    gpio_mode_setup(GPIOC, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO13);
    rcc_periph_clock_enable(RCC_SYSCFG);  // clock for the EXTI handler
    nvic_enable_irq(NVIC_EXTI15_10_IRQ);
    exti_select_source(EXTI13, GPIOC);
    exti_set_trigger(EXTI13, EXTI_TRIGGER_FALLING);
    exti_enable_request(EXTI13);

    /* USART2 is connected to the nucleo's onboard ST-Link, which forwards
     * it as a serial terminal over the ST-Link USB connection.
     * This UART is given to the Trice data. */
    rcc_periph_clock_enable(RCC_USART2);
    usart_set_baudrate(USART2, 115200);
```

```
    usart_set_databits(USART2, 8);
    usart_set_stopbits(USART2, USART_STOPBITS_1);
    usart_set_mode(USART2, USART_MODE_TX);
    usart_set_parity(USART2, USART_PARITY_NONE);
    usart_set_flow_control(USART2, USART_FLOWCONTROL_NONE);

    // Enable UART2 interrupts in the system's interrupt controller
    // but do NOT enable generating interrupts in the UART at this
    // time. Let Trice enable them with triceEnableTxEmptyInterrupt()
    nvic_enable_irq(NVIC_USART2_IRQ);
    //usart_enable_tx_interrupt(USART2);
    usart_enable(USART2);

    /* Configure USART TX pin only. We don't get any input via the TRICE
     * channel, so the RX pin can be left unconnected to the USART2 */
    gpio_mode_setup(GPIOA, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO2);
    gpio_set_af(GPIOA, GPIO_AF7, GPIO2);

    /* Enable systick at a 1mS interrupt rate */
    systick_set_reload(84000);
    systick_set_clocksource(STK_CSR_CLKSOURCE_AHB);
    systick_counter_enable();
    systick_interrupt_enable();
}

//////////////////////////
// Time handling utilities
static volatile uint32_t system_millis;

/* "sleep" for delay milliseconds */
void msleep(uint32_t delay)
{
    uint32_t wake = system_millis + delay;
    while (wake > system_millis);
}

uint32_t wallclock_ms(void)
{
    return system_millis;
}

//////////////////////////
// Interupt handlers
// These are weak symbols in libopencm3
// that get overridden here.

// Trice USART
void usart2_isr(void)
{
    #if TRICE_MODE == 200
    triceServeTransmit();
    #endif
}

// External interrupts on pins 10-15, all ports.
// Only PC13 (User button on Nucleo) is enabled in this program.
void exti15_10_isr(void)
{
    exti_reset_request(EXTI13);
    #if TRICE_MODE == 200
    TRICE(Id(0), "Button press at, %d\n", system_millis);
    #endif
}

// Systick timer set to 1ms
void sys_tick_handler(void)
{
    system_millis++;
```

```
    #if TRICE_MODE == 200
    // Start sending what is currently in the Trice transmit buffer
    triceTriggerTransmit();
    #endif
}

int main(void)
{
    hardware_setup();
    TRICE_HEADLINE;
    while (1) {
        msleep(1000);

        // Depending on mode, either print this string to
        // UART (mode 0), or the Trice write buffer (mode 200).
        TRICE(Id(0), "Hello, TRICE, %d\n", 42);

        // TRICE("") with a string parameter only is problematic.
        // See discussion on https://github.com/rokath/trice/issues/279
        // TRICE0("") works in either case
        #ifdef __STRICT_ANSI__
        // if compiled with e.g. --std=c99
        TRICE0(Id(0), "Hello, TRICE\n");
        #else
        TRICE(Id(0), "Hello, TRICE\n");
        TRICE0(Id(0), "Hello, TRICE0()\n");
        #endif

        #if TRICE_MODE == 200
        // Swap Trice transmit/write ping-pong buffers.
        // Stuff printed with TRICE() since the last
        // call to TriceTransfer() will be sent once
        // triceTriggerTransmit() is called.
        TriceTransfer();
        #endif
    }

    return 0;
}
```

40.1.4. nucleo-f411re.ld

```
/* Use the LibOpenCM3-provided defaults for the linker details.
 */
MEMORY
{
    rom (rx)  : ORIGIN = 0x08000000, LENGTH = 512K
    ram (rwx) : ORIGIN = 0x20000000, LENGTH = 128K
}

INCLUDE cortex-m-generic.ld
```

40.1.5. Makefile

```
# Makefile for compiling the Trice demo on LibOpenCM3
# for STM32F411-Nucleo boards
CC=arm-none-eabi-gcc
C_FLAGS=-O0 -std=c99 -ggdb3
C_FLAGS+=-mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16
C_FLAGS+=-Wextra -Wshadow -Wimplicit-function-declaration -Wredundant-decls -Wmissing-prototypes -Wstrict-
prototypes
C_FLAGS+=-fno-common -ffunction-sections -fdata-sections  -MD -Wall -Wundef
```

```
C_FLAGS+=-DSTM32F4 -I/home/kraiskil/stuff/libopencm3/include
# These two are for trice.h and triceConfig.h
C_FLAGS+=-I../../pkg/src/ -I.

LFLAGS=-L${OPENCM3_DIR}/lib -lopencm3_stm32f4 -lm -Wl,--start-group -lc -lgcc -lnosys -Wl,--end-group
LFLAGS+=-T nucleo-f411re.ld
LFLAGS+=--static -nostartfiles
LFLAGS+=-Wl,-Map=memorymap.txt

all: direct_mode.elf irq_mode.elf
.PHONY: flash clean

# Annotate Trice-enabled code.
# trice does this annotation in-place, so here we take
# a copy before running trice.
# I.e. write TRICE macros in foo.c, and this will generate
# the TRICE( Id(1234) .. ) macros into foo.trice.c
%.trice.c: %.c til.json
    cp -f $< $<.bak
    trice update
    cp -f $< $@
    cp -f $<.bak $<

# trice expects this file to exist, can be empty.
til.json:
    touch til.json

direct_mode.elf: main.trice.c ../../pkg/src/trice.c
    ${CC} ${C_FLAGS} $^ -o $@ ${LFLAGS} -DTRICE_MODE=0

flash_direct_mode: direct_mode.elf
    openocd -f interface/stlink-v2.cfg -f target/stm32f4x.cfg -c "program direct_mode.elf verify reset
exit"

irq_mode.elf: main.trice.c ../../pkg/src/trice.c
    ${CC} ${C_FLAGS} $^ -o $@ ${LFLAGS} -DTRICE_MODE=200

flash_irq_mode: irq_mode.elf
    openocd -f interface/stlink-v2.cfg -f target/stm32f4x.cfg -c "program irq_mode.elf verify reset exit"


clean:
    @rm -f *.elf til.json main.trice.c
```

40.1.6. Usage

- Run `make direct_mode.elf` to compile with Trice mode 0.
- Run `make flash_direct_mode` to program the board.
- Run trice: `trice l -p /dev/ttyACM0`.

40.2. Get all project files containing Trice messages

We check the location information file. Every Trice is registered here.

```
cat demoLI.json | grep '"File":' | sort | uniq
        "File": "_test/_ringB_protect_de_tcobs_ua/TargetActivity.c",
        "File": "_test/special_dblB_de_tcobs_ua/TargetActivity.c",
        "File": "_test/special_for_debug/TargetActivity.c",
        "File": "_test/special_protect_dblB_de_tcobs_ua/TargetActivity.c",
        "File": "_test/testdata/triceCheck.c",
        "File": "examples/F030_inst/Core/Src/stm32f0xx_it.c",
        "File": "examples/G0B1_inst/Core/Src/main.c",
        "File": "examples/G0B1_inst/Core/Src/stm32g0xx_it.c",
        "File": "examples/L432_inst/Core/Inc/triceConfig.h",
        "File": "examples/L432_inst/Core/Src/main.c",
```

```
          "File": "examples/L432_inst/Core/Src/stm32l4xx_it.c",
          "File": "examples/exampleData/triceExamples.c",
          "File": "examples/exampleData/triceLogDiagData.c",
```

40.3. Building a trice library?

The triceConfig.h is mandatory for the trice code. It controls which parts of the trice code are included. There is no big advantage having a trice library, because it would work only with unchanged settings in the project specific triceConfig.h. Once the trice source files are translated, their objects are rebuilt automatically and only when the triceConfig.h is changed. So only the linker has a bit less to do when it finds a trice library compared to a bunch of trice objects. But does that influence the build time heavily?

The triceConfig.h is the only part of the trice sources which should be modified by the users. It is ment to be a individual part of the user projects. The examples folder shows the usage.

40.4. Possible Compiler Issue when using Trice macros without parameters on old compiler or with strict-C settings

If you encounter a compilation error on `trice( "hi");` for example, but not on `trice( "%u stars", 5 );`, this is probably caused by the way your compiler interprets variadic macros. Simply change to `trice0( "hi");` or change your compiler settings. See issue #279 for more details. If your project needs to be translated with strict-C settings for some reason, you have to use the `trice0` macros when no values exist for the Trice macros.

(back to top)

41. Trice And Legacy User Code

When it comes to use legacy sources together with Trice, there are several ways doing so, which do not exclude each other:

- Legacy User Code Option: Separate Physical Output Channel
- Legacy User Code Option: Trice Adaption Edits
- Legacy User Code Option: Print Buffer Wrapping and Framing
- Legacy User Code Option: Trice Aliases Adaption

41.1. Legacy User Code Option Separate Physical Output Channel

Advantages:

- No user code adaption at all needed.
- Code can mix user prints and Trices.

Disadvantages:

- A 2nd physical output is needed.
- The log output goes to one or the other app, what may result in a partial sequence information loss.
- Suboptimal result for target image size and speed, because the legacy user code still prints and transmits strings.

Details:

- The legacy user code output drives a terminal app and the Trice output feeds the Trice binary data into the Trice tool.

41.2. Legacy User Code Option Trice Adaption Edits

Advantages:

- This is the most straight forward method.
- Optimal result for target image size and speed.

Disadvantages:

- No mixed user prints and Trices.
- Legacy code gets changed, needs new testing and is not usable parallel in other existing projects anymore.
- Error prone, even when done KI supported.
    - Max 12 integers/floats **OR** a single runtime generated string in one Trice possible, otherwise splitting into several Trices is needed.
    - `float x` values need wrapping with `aFloat(x)`.

    - `double x` needs wrapping with `aDouble(x)`.
    - `int64` and `double` need `trice64` instead of `trice` or generally use 64-bit width `trice`.
- Not applicable for a large legacy code basis.

Details:

- All exising user prints are replaced with appropriate Trice macros according chapter Trice Similarities and Differences to printf Usage.
- When using 64-bit as default Trice bit width, more RAM is used compared to 32-bit, but in combination with the default TCOBS compressing framing the transmitted Trice packets do not increase much compared to 32-bit width.

41.3. Legacy User Code Option Print Buffer Wrapping and Framing

> **Trice >= v1.1 feature**, see also issue #550

Advantages:

- Code can mix user prints and Trices.
- Legacy code stays unchanged and is usable parallel in other existing projects.

Disadvantages:

- Suboptimal result for target image size and speed, because the legacy user code still prints and transmits strings.
- The reserved case, both Binary Encoding stamp selector bits are 0, is not available anymore for additional use cases.
- The log output may have a partial sequence information loss.

Details:

The Trice binary encoding uses states 1, 2, 3 of the 4 states, the 2 Binary Encoding stamp selector bits can have. They located in the starting `uint16_t` ID value to encode the Trice (time) stamp size. If both bits are zero (state 0), the Trice tool can interpret the incoming data buffer according to a passed CLI switch; in this special case just printing it as string.

If the Trice library and the user print both write to the same output, an easy modification would be, to prepend the user print output with a 2-byte count as long its size is < 16383, so that the 2 most significant bits are zero. Additionally, the this way counted buffer needs the same buffer framing as the Trice binary data.

41.4. Legacy User Code Option Trice Aliases Adaption

> **Trice >= v1.1 feature**, see also accepted pull requests #533 and #536

Advantages:

- Code can mix user prints and Trices.
- Legacy code stays unchanged or mainly unchanged and is usable parallel in other existing projects.
- Nearly optimal result for target image size and speed.
- No special wrapping and need to use the Binary Encoding stamp selector bits state 0 for this.
- Especially, when adapting user specific ASSERT macros with `-salias` (see below), even their strings are compiled into the Target image, only in error cases the strings are printed and transmitted.

Disadvantages:

- The legacy user code could *partially* still print and transmit strings, especially when `float` or `double` are used.

Details:

This cool functionality was contributed by @srgg in pull requests (PR) #533 and #536 (to be considered as one PR only). It allows code integration containing user specific log statements into Trice instrumented projects without the need to rename the user specific log statements.

In the assumption, most user `printi` statements having only up to 12 integers, those user prints could get covered by adding `-alias printi` to the `trice insert` and `trice clean` commands.

The user `printi` statements containing floats, doubles, strings could get simply renamed into user `prints` and then `-salias prints` will cover them too. That, of course, is a legacy user code change, but it allows to use this slightly modified legacy user code parallel in other projects.

Yes, user `printi` and user `prints` need to be defined too. See ./_test/alias_dblB_de_tcobs_ua/triceConfig.h/triceConfig as a simple example and its usage in ./_test/alias_dblB_de_tcobs_ua/TargetActivity.c

This technique allows also to cover legacy user code specific ASSERT macros, as shown in ./_test/aliasassert_dblB_de_tcobs_ua/triceConfig.h and used in the tests ./_test/aliasassert_dblB_de_tcobs_ua/TargetActivity.c.

Despite of these 2 CGO tests the real-world example ./examples/G0B1_inst shows the usage too.

The following sub-chapters are mainly written by @srgg as accompanying documentation to its pull requests.

41.4.1. PR533 Doc

41.4.2. PR533 Summary

This PR introduces support for treating user-defined macros as aliases to trice and triceS within the Trice CLI toolchain. The goal is to enable project-specific logging macros to be processed just like built-in Trice macros — including ID generation, decoding, and binary format support — without requiring projects to directly call `trice()` or `triceS()` in their source code.

PR leverages the `-exclude` source feature added in #529.

41.4.3. PR533 Motivation

Trice uses a source-scanning and ID generation approach, where the toolchain scans for `trice(...)` and `triceS(...)` calls, injects numeric trace IDs, and builds a mapping database. However, it currently only supports built-in(hardcoded) macros and allows only global on/off control via compile-time flags.

This makes it difficult to:

- Adopt custom naming conventions (`DBG()`, `APP_LOG()`, `MON()`, etc.).
- Redirect trace/logging behavior to other backends (e.g., MicroSD, raw printf, no-op).
- Change behavior per module or configuration without losing Trice tooling support.

41.4.4. What This PR533 Adds

CLI-level aliasing: Developers can now declare custom macros to be treated as `trice` or `triceS` equivalents. These user-defined macros will be recognized during scanning, ID injection, and decoding.

41.4.5. PR533 Example

print_macro.h:

```
#ifndef TRICE_OFF
  #define DEBUG_PRINT(...)  trice(__VA_ARGS__)
  #define DEBUG_PRINT_S(...)  triceS(__VA_ARGS__)
#else
  #define DEBUG_PRINT(...)  Serial.println(__VA_ARGS__)
  #define DEBUG_PRINT_S(...)  Serial.printf(__VA_ARGS__)
#endif
```

example.c:

```
#include "trice.h"
#include "print_macro.h"

void setup() {
    Serial.begin(115200);

    while (!Serial) {
        delay(10);
    }

   // Add code here to initialize whatever Trice sender TCP/UDP/UART, etc.
```

```
  // No argument
  DEBUG_PRINT("DEBUG_PRINT test: no args\n");

  char* str = "Test string";
  DEBUG_PRINT_S("DEBUG_PRINT_S test: %s\n", str);
 }
```

PR533 Check with Trice

Insert trice IDs:

```
trice insert -alias DEBUG_PRINT -salias DEBUG_PRINT_S  -exclude ./print_macro.h -v
```

Flash the MCU and run the trice receiver on your host machine to receive probes (cli command is config and receiver dependent), for UDP4, it can be:

```
   trice log -p UDP4 -v -pf none
```

PR533 Check without Trice:

Clean trice IDs, if any:

```
trice clean -alias DEBUG_PRINT -salias DEBUG_PRINT_S  -exclude ./print_macro.h -v
```

Flash with `-DTRICE_OFF`.

41.4.6. PR536 Doc

What This PR536 Adds

This is a follow-up to #533. It **enforces the consistent use of the "%s" format in all triceS aliases** and fixes that behavior in the newly added test case.

The following simplified example reflects a real use case where custom macros wrap formatting logic:

```
#define CUSTOM_PRINT_S(id, fmt, ....) triceS(id, "%s", format_message(fmt, ##__VA_ARGS__))
```

PR536 - The Problem Statement

Determining the Strg argument reliably is challenging due to:

- The unrestricted flexibility of macros (*which I would like to preserve and utilize*).
- Trice core's limited parsing, which relies on regular expressions.

For instance, custom macros like these show the variability:

```
CUSTOM_ASSERT(false, "Message: %s", msg);
CUSTOM_ASSERT(false, "Message without args");
CUSTOM_ASSERT(false);
```

Improving this would likely require Clang integration—adding complexity (e.g., full build flags, complete source context)—whereas Trice's current regex-based approach remains lightweight and simple to use.

PR536 Implementation Details:

`matchTrice()` was re-implemented to improve robustness. It now:

- Locates the macro or alias name followed by (.
- Finds the matching closing parenthesis, correctly handling nested structures.
- Parses the arguments within.

This approach simplifies the logic and allows the parser to skip invalid or partial matches without aborting, enabling continued scanning of the file for valid constructs.

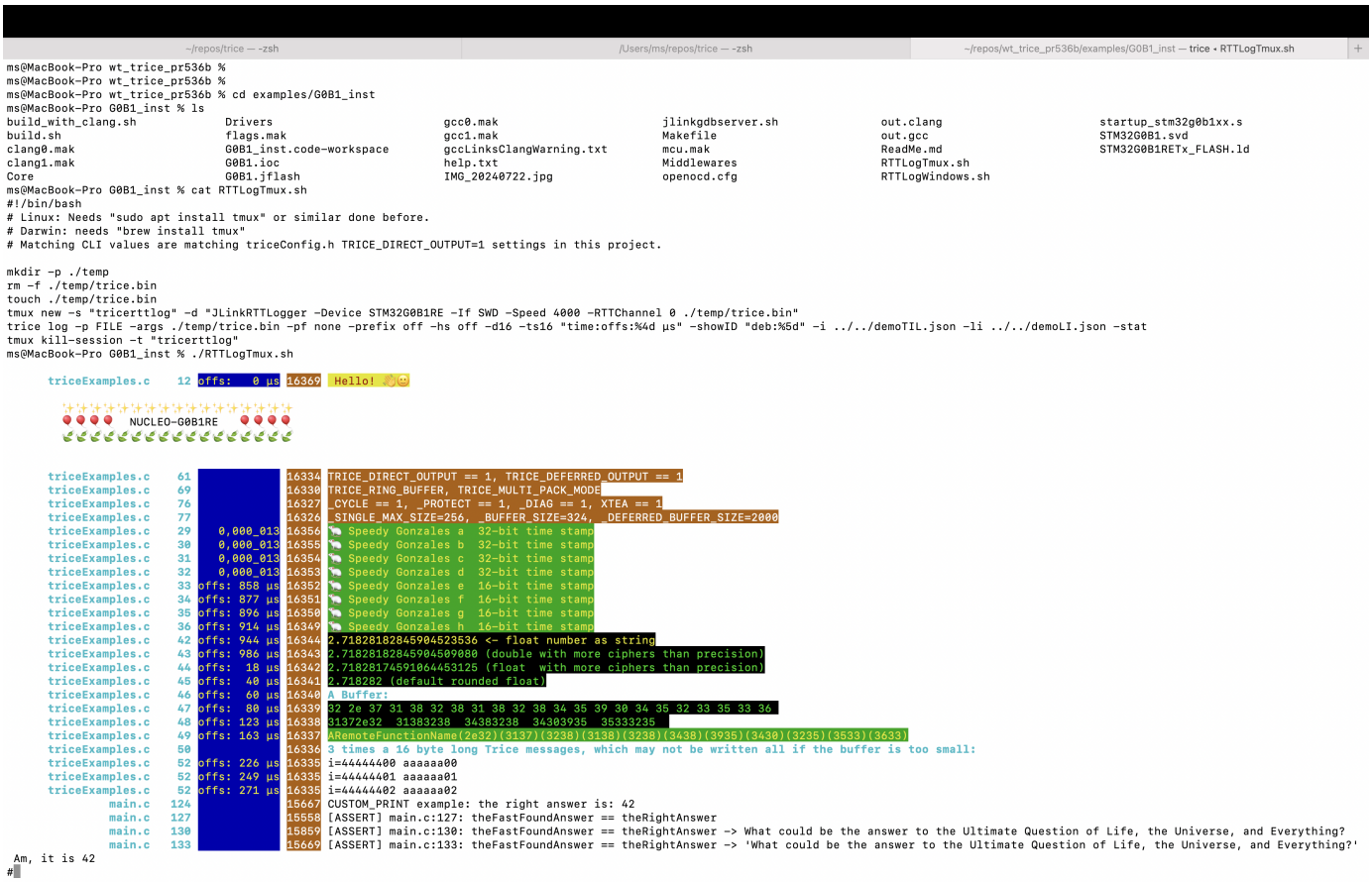
41.4.7. Alias Example Project

To use the Alias technique with `examples/G0B1_inst` the following adaptions where made:

- Copied file ./examples/G0B1_inst/Core/Inc/nanoprintf.h from https://github.com/charlesnicholson/nanoprintf
- Created file ./examples/G0B1_inst/Core/Inc/triceCustomAliases.h to cover the user code specific `CUSTUM_PRINT` and `CUSTUM_ASSERT`.
- *Core/Src/main.c*:

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
- #include "trice.h"
+ #include "triceCustomAliases.h"
#include <limits.h> // INT_MAX
/* USER CODE END Includes */

...

  /* USER CODE BEGIN 2 */
#if !TRICE_OFF
  LogTriceConfiguration();
  SomeExampleTrices(3);
- #endif
+   /* Some Custom Trice Alias Examples */
+   const int theRightAnswer = 42;
+   const int theFastFoundAnswer = 24;
+   const char* theQuestion = "What could be the answer to the Ultimate Question of + Life, the
Universe, and Everything?";
+
+   // Some Trice custom alias examples
+   CUSTOM_PRINT("CUSTOM_PRINT example: the right answer is: %d\n", theRightAnswer);
+
+   // Assert with condition
+   CUSTOM_ASSERT(theFastFoundAnswer == theRightAnswer);
+
+   // Assert with condition and a message: This works too, but triggers a clang + compiler warning,
we cannot suppress.
+   CUSTOM_ASSERT(theFastFoundAnswer == theRightAnswer, (char*)theQuestion ); // +
https://stackoverflow.com/questions/52692564/+ how-can-i-disable-format-security-error-with-clang
+
+   // Assert with condition and a message and some extra message arguments
+   CUSTOM_ASSERT(theFastFoundAnswer == theRightAnswer, (char*)"'%s' Am, it is %d", +
(char*)theQuestion, theRightAnswer);
+ #endif
  /* USER CODE END 2 */
```

- After flashing:



(back to top)

42. Future Development

42.1. Trice Log-level Control Specification Draft

> Specification Draft

```
--> IMPORTANT: No breaking changes!
```

42.1.1. What log levels exist in general, including exotic ones, and what is their exact weighting relative to each other?

🧭 Basic principle

- Lower level → more noise / diagnostic detail
- Higher level → more severe / critical condition

🔢 Common standardized levels (by severity)

Level	Name	Weight	Meaning
0	TRACE	lowest	Finest-grained details — e.g., every function call or variable change. Used for deep debugging only.
1	DEBUG	low	Developer-level details about execution flow. No failure.
2	INFO	medium	Normal operational messages — startup, config loaded, connection established.
3	NOTICE	slightly higher	Significant but expected events (e.g., user login). Exists in syslog.
4	WARN / WARNING	rather high	Something unexpected but not yet a failure. System continues running.
5	ERROR	high	A problem occurred — operation failed, but program still runs.

Level	Name	Weight	Meaning
6	CRITICAL	very high	A subsystem failure. Urgent attention required.
7	ALERT	extremely high	Immediate human intervention needed.
8	EMERGENCY / FATAL / PANIC	highest	System unusable. Shutdown or restart required.

Rare or exotic variants

Name	Origin / Context	Severity	Description
VERBOSE	Windows, Android, C/C++ loggers	Between TRACE and DEBUG	Very detailed, but not quite as deep as TRACE.
SUCCESS / OK / PASS	Test frameworks	Between INFO and NOTICE	Indicates successful operations.
FAIL	Test frameworks	ERROR	Failed test but not system error.
SECURITY / AUDIT	Compliance systems	Variable	Logs security or compliance events separately.
CONFIG / INIT	Embedded / frameworks	INFO	Configuration or initialization messages.
DEPRECATION	Compilers, frameworks	WARN	Deprecated feature warnings.
ASSERT	Debuggers, C/C++	CRITICAL	Assertion failure, usually aborts program.
NOTICE / IMPORTANT / EVENT	Various	Between INFO and WARN	Events worth attention but not errors.
OFF	Logging frameworks	none	Turns off all logging.
ALL	Logging frameworks	lowest	Enables every log level.

Example comparison across systems

Severity	Syslog	Log4J / Java	Python	.NET	Meaning
0	debug	TRACE	NOTSET	Trace	Internal details
1	info	DEBUG	DEBUG	Debug	Developer info
2	notice	INFO	INFO	Information	Normal ops
3	warning	WARN	WARNING	Warning	Unexpected
4	err	ERROR	ERROR	Error	Operation failed
5	crit	FATAL	CRITICAL	Critical	Severe
6	alert	—	—	—	Immediate action
7	emerg	—	—	—	System crash

Suggested numeric scale

Level	Weight	Meaning
TRACE	10	Ultra-detailed
VERBOSE	20	Very detailed
DEBUG	30	Developer info
INFO	40	Normal operation
NOTICE	50	Significant event

Level	Weight	Meaning
WARN	60	Warning
ERROR	70	Error
CRITICAL	80	Serious problem
ALERT	90	Urgent
EMERGENCY / FATAL / PANIC	100	Total failure

For The Trice project (an embedded logging tool), logging must be:

- lightweight,
- memory-efficient (Flash/RAM),
- but still expressive enough for both developers and customers.

Here's a 7-level scheme, embedded-friendly yet compatible with syslog/log4j conventions:

🔧 Recommended Trice Log Level Scale

Macro/Level	Name	Weight	Meaning	Typical Use
0 – trice_SILENT	**OFF / NONE**	0	No output at all.	Disable logging in release builds.
1 – trice_FATAL	**FATAL / PANIC**	100	System unusable, restart required.	Watchdog reset, hard fault, stack overflow.
2 – trice_ERROR	**ERROR**	80	Recoverable error.	CRC failure, timeout, file missing.
3 – trice_WARN	**WARN**	60	Unexpected but tolerable.	Retry, threshold exceeded.
4 – trice_INFO	**INFO**	40	Regular operation messages.	Init complete, connection established.
5 – trice_DEBUG	**DEBUG**	30	Developer-level diagnostics.	Variable states, state transitions.
6 – trice_VERBOSE	**VERBOSE / TRACE**	10	Deepest trace level.	Function calls, ISR entry, timings.

🎯 Advantages

- ☑ Compatible with Syslog conventions
- ☑ Filtered by one threshold (if(level <= currentLevel))
- ☑ Backward-compatible (old log constants still valid)
- ☑ Easily extendable (e.g., add NOTICE or ASSERT later)

42.1.2. Compile-time Log-level Control

In Trice Structured Logging Compile-time Information we see, how `trice insert ...` could modify (temporarily) the source code. With an additional *insert* switch like `-loglevel` the shown example could get changed in this way:

User may have written inside *val.c*:

```
void doStuff( void ){
    // ...
    trice("info:The answer is %d.\n", 42);
    // ...
}
```

and a `trice insert -loglevel` command could change that into (revertable with `trice clean`):

```
void doStuff( void ){
    // ...
    trice_INFO(iD(123), "info:The answer is %d.\n", 42);
    // ...
}
```

The idea here is to modify also the `trice` macro name into `trice_INFO`, when a Trice tag "info" or "inf" was found. We could define this way:

```
#define TRICE_LOG_LEVEL TRICE_LEVEL_INFO

#if TRICE_LOG_LEVEL >= TRICE_LEVEL_INFO
#define trice_INFO(...) trice( __VA_ARGS__)
#else
#define trice_INFO(...) ((void)0)
#endif

#if TRICE_LOG_LEVEL >= TRICE_LEVEL_DEBUG
#define trice_DEBUG(...) trice(__VA_ARGS__)
#else
#define trice_DEBUG(...) ((void)0)
#endif
```

That results in no code generation for `trice("info:The answer is %d.\n", 42);` for TRICE_LOG_LEVEL < TRICE_LEVEL_INFO. What we get this way is:

- A fine-granular compile-time log-level control.
- The user is free to add its own log-levels.
- No run-time costs at all.

42.1.3. Run-time Log-level Control

Trice logs are very light-weight and usually is no need for their run-time control. Nevertheless there could be a need for that. The very first we need, is a control channel to tell the target device about a changing log-level. See for example chapter Stimulate target with a user command over UART.

When we are able to set a value *LogLevel* in the target device, we can use this value as an ID threshold in combination with the `-IDRange` switch. More in detail as an example:

```
trice insert -loglevel -IDRange debug:1,999 -IDRange info:2000,2999 -IDMin 4000 -IDMax 9999 -IDRange
err:10000,10999
```

It is important to understand, that all other Trice messages get IDs in the range `-IDMin` and `-IDMax` and that no range overlapping is allowed.

LogLevel	Result
16384	no output
10000	only error messages
4000	normal messages and error messages
2000	all output except info and debug messages
1000	all output except debug messages
0	all output

That implies a small Trice library extension, which gets active only with a `LOGLEVELS` switch. In that case we get a small additional run-time overhead. What we cannot achieve this way is a tag specific target-side selection, but that would be no big deal to add as well.

(back to top)

42.2. Trice Structured Logging

> **Specification Draft**

Structured logging, in contrast to unformatted logging, automatically adds compile time and runtime data to logs as well as log level information. The user should be able to configure, which data get added and also should have control about the data formatting. The generic data insertion allows later an automatic log file analysis and frees the developer from manually typing defaults, what is also error-prone.

Trice is considerable already a bit as a (very limited) structured logger, if we look at the file and line insertion capability and the timestamp options. The following is about how Trice could get full structured logging capability without making a breaking change.

42.2.1. Trice Structured Logging Compile-time Information

file, line, function, compiler version, module, build time, firmware version, machine name, user name, locale, host OS version, log level, an (unstructured) format string, compiler flags, (locally) defined values...

These data can be strings or numbers.

42.2.2. Trice Structured Logging Runtime Information

uptime, timestamp, hw serial, task ID, stack depth, event count, core ID, position, variables values, parameter values ...

In an initial approach we assume, these data do not contain runtime generated strings. If really needed, a derived hash is usable instead for now. Despite of this, runtime generated strings are an important feature and therefore Trice supports `triceS`, capable to transmit a single string up to 32KB long, and `triceS` relatives like `triceB`. We could add compile-time data (as inserted fixed strings) but runtime information can only get as an additional part of the runtime generated string into the structured log. This should be acceptable and we will deal with this later.

42.2.3. Trice Structured Logging Limitations and Special Cases

For performance reasons, Trice was designed to only transmit 0-12 (straight forward extendable) numbers of equal bit-width **OR** a single runtime generated string. Firstly we look at only "normal" Trice macros `trice`, `Trice`, `TRice` and exclude the special cases `triceS`, `TriceS`, `TRiceS`. Also we consider just trices without specified bit-width, assume 32-bit and exlude cases like `trice32_4` firstly.

42.2.4. A Trice Structured Logging Example

User may have written inside *val.c*:

```
void doStuff( void ){
    // ...
    trice("info:The answer is %d.\n", 42);
    // ...
}
```

and (as we know) a `trice insert` command would change that into (revertable with `trice clean`):

```
void doStuff( void ){
    // ...
    trice(iD(123), "info:The answer is %d.\n", 42);
    // ...
}
```

But a `trice insert` command with context option will, for example, change that line into (revertable with `trice clean`):

```
void doStuff( void ){
    // ...
    trice(iD(456), "[level=info][file=\"val.c\"][line=321][func=doStuff][taskID=%x][fmt=\"The answer is
%d.\"][uptime=%08us][temperature=%3.1f°C]\n", getTaskID(), 42, uptime(), aFloat(sensorValue));
    // ...
}
```

42.2.5. Trice Structured Logging CLI Switches and Variables

To achieve that, 2 structured logging CLI switches `-stf` and `-stv` on `trice insert` and `trice clean` are usable:

CLI switch	meaning
`-stf`	structured logging format
`-stv`	structured logging values

Additionally the Trice tool uses these internal variables (no bash variables!) as replacements during `trice insert` and `trice clean`:

Variable	Example	Comment
`$level`	`info`	The bare trice format string part until the first colon (`:`), if known as channel/tag value.
`$file`	`val.c`	The file name, where the Trice log occures.
`$line`	`321`	The file line, where the Trice log occures.
`$func`	`doStuff`	The function name, where the Trice log occures.
`$fmt`	`The asnwer is %d.`	The bare Trice format string stripped from the channel/tag specifier including the colon (`:`) according to the Trice rule (lowercase-only ones)
`$values`	`42`	The bare Trice statement values.
`$usr0`	`abc` || `xyz`	A predefined string value with location dependent values (see below).

42.2.6. Trice Structured Logging User Defined Values

This use case is not expected for most cases, but mentioned here to show the possibilities. Adding user specific values like `$usr0` can be done in this way:

- File *main.c*:

```
 88 | ...
 89 | #define XSTR(x) STR(x)
 90 | #define STR(x) #x
 91 |
 92 | trice("info:hi");
 93 |
 94 | #define TRICE_ETC "xyz"
 95 | #pragma message "$usr0=" XSTR(TRICE_ETC)
 96 | trice("info:hi");
 97 |
 98 | #undef TRICE_ETC
 99 | #pragma message "$usr0=" XSTR(TRICE_ETC)
100 | trice("info:hi");
101 |
102 | #define TRICE_ETC "abc"
103 | #pragma message "$usr0=" XSTR(TRICE_ETC)
104 | trice("info:hi");
105 | ...
```

This is just a demonstration. The `#pragma message "$usr0=" XSTR(TRICE_ETC)` line probably is needed only on a few lines in the project. A pre-compile output

```
$ ./build.sh 2>&1 | grep "pragma message:"
Core/Src/main.c:95:9: note: '#pragma message: $usr0="xyz"'
Core/Src/main.c:99:9: note: '#pragma message: $usr0=""'
Core/Src/main.c:103:9: note: '#pragma message: $usr0="abc"'
```

could get transferred automatically to the Trice tool, with a user generator script to tell, that normally `$usr0=""`, but `$usr0="xyz"` for Trices in file *main.c* from line 95 to 99, that `$usr0="abc"` is valid for *main.c* after line 103.

Those things are compiler and user specific and not part of the Trice tool design. But on demand a CLI multi switch `-stu` can get invented, to inject such information into the `trice insert` process automatically. With

```
STF='{"level":"%s","loc":"%s:%d","fmt":"$fmt","etc":"%s"}'
STV='$level, $file, $line, $values, $usr0'

# user script generated begin ##################################################
ST0='usr0="xyz":main.c:95'                      # user script generated line
ST1='usr0="":main.c:99'                         # user script generated line
ST2='usr0="abc":main.c:103'                     # user script generated line
STU="-stu $ST0 -stu $ST1 -stu $ST2"             # user script generated line
# user script generated end ####################################################

trice insert $STU -stf $STF -stv $STV
```

The structured log output would be:

```
{...}
{"level":"info","loc":"main.c:92","fmt":"hi","etc":""}
{"level":"info","loc":"main.c:96","fmt":"hi","etc":"xyz"}
{"level":"info","loc":"main.c:100","fmt":"hi","etc":""}
{"level":"info","loc":"main.c:104","fmt":"hi","etc":"abc"}
{...}
```

42.2.7. Trice Structured Logging CLI Switches Usage Options

The in A Trice Structured Logging Example shown `trice insert` result is possible with

```
trice insert \
-stf='[level=$level][file=$file][line=$line][func=$func][taskID=%x][fmt=$fmt][uptime=%08us]
[temperature=%3.1f°C]' \
-stv='getTaskID(), $values, uptime(), aFloat(sensorValue)'
```

The raw string syntax is mandatory here, to pass the internal Trice tool variables names.

Adding variable values like `$line` as strings has performance advantages, but on each such value change a new Trice ID is generated then. Those variables are better inserted as values, if the code is under development. A `$line` value insertion looks like this:

```
trice insert \
-stf='[level=$level][file=$file][line=%5d][func=$func][taskID=%04x][fmt=$fmt][uptime=%08us]
[temperature=%3.1f°C]' \
-stv='$line, getTaskID(), $values, uptime(), aFloat(sensorValue)'
```

It is also possible to use string format specifiers to allow somehow aligned values. For example:

```
trice insert \
-stf='[level=%-6s][file=%24s][line=%5d][func=%-16s][taskID=%04x][fmt=$fmt][uptime=%08us]
[temperature=%3.1f°C]' \
-stv='$level, $file, $line, $func, getTaskID(), $values, uptime(), aFloat(sensorValue)'
```

Or, if you like alignment after the format string, even:

```
trice insert \
-stf='[level=%-6s][file=%24s][line=%5d][func=%-16s][taskID=%04x][fmt=%64s][uptime=%08us]
[temperature=%3.1f°C]' \
-stv='$level, $file, $line, $func, getTaskID(), $fmt, $values, uptime(), aFloat(sensorValue)'
```

The user has full control and could also use any other syntax like a JSON format. Only the format specifiers are requested to match the passed values after the Trice tool internal variables replacement during `trice insert`, so that the Trice tool can perform a printf during logging.

To achieve a log output in compact JSON with line as string we can use:

```
trice insert \
-stf='{"level":"$level","file":"$file","line:"$line","taskID":"%04x","fmt":$fmt,"uptime":%08u us"}' \
-stv='getTaskID(), $values, uptime()'
```

To put things together: Any structured format string design is possible and the user can insert the $line (example) value:

- directly as string (fastest execution, straight forward)
- indirectly as formatted string (fastest execution alignment option)
- indirectly as formatted number (recommended when often changing)

After `trice insert` we get this (compact JSON) log line according to `-stf` and `-stv`:

```
void doStuff( void ){
    // ...
    trice(iD(789), "
{\"level\":\"info\",\"file\":\"val.c\",\"line\":\"321\",\"taskID\":\"%04x\",\"fmt\":\"The answer is
%d.\",\"uptime\":\"%08u us\"}\n', getTaskID(), 42, uptime());
    // ...
}
```

All compile time strings are part of the Trice format string now, which is registered inside the *til.json* file. The needed Trice byte count stays 4 bytes only plus the 3 times 4 bytes for the runtime parameter values taskID, 42, uptime. The default TCOBS compression will afterwards reduce these 16 bytes to 12 or 13 or so.

A `trice clean` command will remove the context information completely including the ID. Please keep in mind, that with `trice insert` as a pre-compile and `trice clean` as post-compile step, the user all the time sees only the original written code:

```
void doStuff( void ){
    // ...
    trice("info:The answer is %d.\n", 42);
    // ...
}
```

The optional `-cache` switch makes things blazing fast.

The appropriate Trice tool log line output would be similar to

```
{...}
{"level":"info","file":"val.c","line":"321","taskID":"0123","fmt":"The answer is 42.","uptime":"12345678
us"}
{...}
```

When *stf* and *stv* are empty strings (default), `trice insert` and `trice clean` commands will work the ususal way. If they are not empty, the `trice insert` command will on each Trice statement use a heuristic to check if the context information was inserted already and update it or otherwise insert it. **ATTENTION:** That will work only, if *stf* and *stv* where not changed by the user inbetween. In the same way

`trice clean` would remove the context information only, if *stf* and *stv* kept unchanged. If the user wants to change *stf* and *stv* during development, first a `trice clean` is needed. Use a `build.sh` script like this:

```
#!/bin/bash

# Run "rm -rf ~/.trice/cache/*" automatically after changing this file !!!

STF='{"level":"$level","file":"$file","line:"$line","taskID":"%04x","fmt":$fmt,"uptime":%08u us"}'
STV='getTaskID(), $values, uptime()'

trice insert -cache -stf="$STF" -stv="$STV"
# make
trice clean  -cache -stf="$STF" -stv="$STV"
```

The `-cache` switch is still experimental - to stay safe, use (here again with `$line` as string):

```
#!/bin/bash
STF='{"level":"$level","file":"$file","line:"$line","taskID":"%04x","fmt":$fmt,"uptime":%08u us"}'
STV='getTaskID(), $values, uptime()'

trice insert -stf="$STF" -stv="$STV"
# make
trice clean  -stf="$STF" -stv="$STV"
```

42.2.8. Trice Structured Logging Level Specific Configuration

Configure the Trice Structured Logging selectively in a way, to provide as much helpful diagnostic info as possible on `ERROR` level for example. Example script:

```
#!/bin/bash

# Specify `-stf` and `-stv` differently for different channels/tags.

STL="" # Trice Structured Logging configuration

# Trices with an `ERROR:` tag `trice("err:...", ...);`:
STF_ERROR='ERROR:{"log
level":"%-6s","file":"%24s","line:"%5d","func":"%-16s","taskID":"%x","fmt":"$fmt","uptime":"%08u us"}'` #
(with location)
STV_ERROR='ERROR:$level, $file, $line, $func, getTaskID(), $values, uptime()'`
STL+=" -stf $STF_ERROR -stv $STV_ERROR "

# Trices with an underscore tag, like `trice("_DEBUG:...", ...);` or `trice("_info:...", ...);`:
STF_underscoreTagStart='_*:{"log
level":"%-6s","file":"%24s","line:"%5d","func":"%-16s","fmt":"$fmt","uptime":"%08u us"}'` # (no task ID)
STV_underscoreTagStart='_*:$level, $file, $line, $func, $values, uptime()'`
STL+=" -stf $STF_underscoreTagStart -stv $STV_underscoreTagStart "

# Tices with any other tag:
STF_anyTag='*:{"log level":"%-6s","file":"%24s","line:"%5d","func":"%-16s","fmt":"$fmt"}'` # (no task ID,
no uptime)
STV_anyTag='*:$level, $file, $line, $func, $values'`
STL+=" -stf $STF_anyTag -stv $STV_anyTag "

# Trices with no tag at all:
STF_noTag='{"file":"%24s","line:"%5d","fmt":"$fmt"}'` # (only location information)
STV_noTag='$file, $line, $values'`
STL+=" -stf $STF_noTag -stv $STV_noTag "

trice insert $STL ...
source make.sh # build process
trice clean  $STL ...
```

42.2.9. Trice Structured Logging Assert Macros (TODO)

Configure `TriceAssert` like macros and this works also with the `-salias` switch.

(back to top)

42.3. Improving the Trice Tool Internal Parser

42.3.1. Trice Internal Log Code Short Description

Trice v1.0 Code

> **Hint:** To follow this explanation with the debugger, you can open in VSCode the trice folder, klick the Run & Debug Button or press CTRL-SHIFT-D, select `trice l -p DUMP` and set a breakpoint at `func main()` in ./cmd/trice/main.go or directly in `translator.Translate` ./internal/translator/translator.go.

- In file ./internal/args/handler.go function `logLoop` calls `receiver.NewReadWriteCloser` and passes the created `rwc` object to `translator.Translate`.
- There `rwc` is used to create an appropriate decode object `dec` passed to `decodeAndComposeLoop`, which uses `func (p *trexDec) Read(b []byte) (n int, err error)` then, doing the byte interpretation.
    - Finally `n += p.sprintTrice(b[n:]) // use param info` is called doing the conversion.
- Read returns a **single** Trice conversion result or a **single** error message in b[:n] or 0 and is called again and again.
- The returned Trice conversion result is the Trice format string with inserted values, but no timestamp, color or location information. The target timestamp, for this single Trice is hold in `decoder.TargetTimestamp`. It is used only when a new log line begins, what is the normal case. If a Trice format string ends not with a newline, the following Trice gets part of the same log line and therefore its target time stamp is discarded. Also additional data like the location information only displayed for the first Trice in a log line containing several Trices. Because the color is inherent to the Trice tag and needs no display space it is attached to the following Trices in a log line as well.
- The after `Read` following `emitter.BanOrPickFilter` could remove the Read result.
- If something is to write, the location information is generated if a new line starts and passed to the with `sw := emitter.New(w)` created object `sw.WriteString` method which internally keeps a slice of type `[]string` collecting all parts of an output line.
- The line `p.Line = append(p.Line, ts, p.prefix, sx)` adds host timestamp `ts`, the build `p.prefix` and the "printed Trice" (`sx` is here just the location information) to the line slice.
- In the next step the stored target timestamp `decoder.TargetTimestamp` is printed into a string and added to the Trice line.
- Optionally the Trice ID follows, if desired.
- The in a string printed Trice follows now and if the `sw.WriteString` method detects a `\n` at its end, the configured line suffix (usually `""`) follows and `p.completeLine()` is called then. It passes the line (slice of strings) to `p.lw.WriteLine(p.Line)`, which adds color, prints to the output device and clears the sw.Line slice for the next line.

Disadvantages of Trice v1.0 Implementation

- The Reader can only return a **single** Trice, because its byte interface cannot distinguish between Trices anymore.
- The field order (prefix, host stamp, location information, target stamp optional ID, format string, suffix) is hardcoded.
- Trices with several newlines inside the format string cannot deal with tags after an (internal) line break (newline)
- Binary parser needs to hold internal location after one Trice was decoded.
- Only one target timestamp value in a global variable.
- Line writing is not straight forward understandable.

Aims for a better implementation

- Read should parse the binary data only and return a Trice struct slice.
- Cycle errors?
- Each Trice struct gets printed in a string.
- The printed string then is split into several strings, which all get the same stamp information or space fields.
- Finally we have a slice of such structs: hs, ts, loc, idString, format string.
- The format string has no newline inside anymore, but has one at the end (usually) or not.
- The struct slice is cyclically passed to a line writer, which writes one line if it can find a format string ending with a newline.

42.4. Using Trice on Servers

- The internet traffic causes many megabytes logfiles, which need storage and are also often transferred by themselves.
- Of course it is possibe to compress them to save space and traffic.
- But if a server generates binary Trice log data directly:
  - The log generation is much faster and demands less energy.
  - An additional compression is not needed, because the Trice internal TCOBS already does it in a reasonable way.
  - Less log data are to be transferred.
- To read the (binary) log data, the matching Trice-Id-List *til.json* is needed. Because this has a size of only a few kilobytes, the server can transmit it on request.
- The ~16000 usable IDs may be not enough for big systems. Options:
  1. Change the TREX binary format to
  - use 32- or 48- or 64-bit IDs. The many zeroes are efficiently compressed internally with TCOBS.
  - use the full 16-bit IDs for ~65000 IDs
  2. Use up to 2^16 or 2^32 different *til.json* files and transmit their index in the optional Trice stamp field. This requests minimal code adaptions.

(back to top)

43. Working with the Trice Git Repository

Action	Command
Get a local repository copy.	`git clone github.com/rokath/trice.git trice`
Show current folder	`pwd`
Show repository status.	`git status`
Clean the repo, if needed.	`git stash push`
Show all branches.	`git branch -a`
Switch to main.	`git switch main`
Fetch a pull request as new branch PRIDa.	`git fetch origin pull/ID/head:PRIDa`
List worktree.	`git worktree list`
Add to worktree.	`git worktree add ../trice_wt_PRIDa PRIDa`
Add branch dev to worktree	`git worktree add ../trice-dev dev`
Rstore the repo if needed.	`git stash pop`
Change to new folder.	`cd ../trice_wt_PRIDa`
Show repository status.	`git status`
Test pull request.	`./testAll.sh full`
Show repository status.	`git status`
Clean pull request.	`git restore .`
Change to previous folder.	`cd -`

Action	Command
Delete worktree branch.	`git worktree remove ../trice_wt_PRIDa`
Delete git branch.	`git branch -d PRIDa`
Log last 3 commits in branch maste	`git log -3 main`
Checkout by hash	`git checkout <hash>`
One Liner Log until shortly before v1.0.0	`git log --graph --decorate --all --pretty=format:'%C(bold yellow)%h%Creset %C(bold green)%ad%Creset %C(bold cyan)%d%Creset %C(white)%s%Creset' --date=format:'%Y-%m-%d %H:%M' --since 2025-04-01`
One Liner Log for branch `devel`	`git log --graph --decorate devel --pretty=format:'%C(bold yellow)%h%Creset %C(bold green)%ad%Creset %C(bold cyan)%d%Creset %C(white)%s%Creset' --date=format:'%Y-%m-%d %H:%M'`
One Liner Log with author	`git log --graph --decorate --all --pretty=format:'%C(bold yellow)%h%Creset %C(bold green)%ad%Creset %C(bold blue)%an%Creset %C(bold cyan)%d%Creset %C(white)%s%Creset' --date=format:'%Y-%m-%d %H:%M'`
New worktree detached branch for compare	`git worktree add --detach ../trice_9995fdc4b 9995fdc4b`
Add a special commit worktree	`./AddWorktreeFromGitLogLineData.sh <commit-hash> <YYYY-MM-DD> <HH:MM>`
Create a bunch of worktrees	`./AddWorktreesBetween.sh "<since-date>" "<until-date>"` or `./AddWorktreesBetween.sh <older-hash> <newer-hash>`
Delete all `trice_*` worktrees	`cd ~/repos && rm trice_* && cd trice && git worktree prune && git worktree list`
Delete all `trice_*` branches	`git branch -D `git branch | grep -E 'trice_'`
Show all opencommit parameter	`oco config describe`
Show some config settings	`oco config get OCO_MODEL && oco config get OCO_PROMPT_MODULE && oco config get OCO_EMOJI`

43.1. Install `opencommit` on MacOS

Prerequisites

Before you begin, make sure you have:

- Install Homebrew first if you don't have it.
- **Git**: Check if Git is installed: `git --version` If not, install the Xcode Command Line Tools:
  - `xcode-select --install`
- **Node.js and npm**
  OpenCommit runs on Node.js. Check with: `node -v` and `npm -v` If not installed: `brew install node`
- **An OpenAI API key** (or compatible provider like OpenRouter, see below).

Step 1 — Get Your OpenAI API Key

- Go to **https://platform.openai.com**
- Log in (or sign up) using your email, Google, Microsoft, or Apple account.
- Navigate to your API Keys page: https://platform.openai.com/account/api-keys
  - Click **"Create new secret key"**.

- Give it a name (e.g., `opencommit-mac`) and copy it immediately.
  It will look like this: `sk-proj-1a2b3c4d5e6f...`

> ⚠ You'll only see it once — store it securely (e.g., 1Password, Bitwarden).

Step 2 — Install OpenCommit

- Run the following in your Terminal: `npm install -g opencommit` You may see warnings about deprecated dependencies — those are safe to ignore.
- Once installed, check: `opencommit --version`

Step 3 — Set Up Your API Key on macOS

- Add your key as an environment variable: `export OPENAI_API_KEY="sk-proj-your-key-here"`
- To make it **permanent**, add it to your shell configuration file (`~/.zshrc`): `echo 'export OPENAI_API_KEY="sk-proj-your-key-here"' >> ~/.zshrc source ~/.zshrc`
- Verify that it's active: `echo $OPENAI_API_KEY` If it prints your key (or the beginning of it), you're good.

Step 4 — Optional Configuration

- You can customize OpenCommit's behavior by setting additional environment variables, for example:

```
export OCO_LANG="en"           # or "de", "fr", etc.
export OCO_MODEL="gpt-5"       # or another model like "gpt-4-turbo"
export OCO_PROMPT_MODULE="conventional"
export OCO_EMOJI=true
```

Add these to your `~/.zshrc` for persistence.

Step 5 — Use OpenCommit

- Go to a Git repository: `cd /path/to/your/repo`
- Stage your changes: `git add .`
- Run OpenCommit: `opencommit`
- It will analyze your staged changes and automatically generate a commit message.
  Example: `🔍 Analyzing changes... ☑ Commit message generated: feat(api): add endpoint for user authentication`

Step 6 — (Optional) Install Git Hook

- To have OpenCommit run automatically every time you commit: `npx opencommit install-hook`
- Now every time you run `git commit`, OpenCommit will propose a commit message for you.

Step 7 — Troubleshooting

If OpenCommit says:

- **"Missing API key"** → Check that `$OPENAI_API_KEY` is set (`echo $OPENAI_API_KEY`).
- **"Unauthorized"** → Verify your key is valid or not expired.
- **"Cannot find opencommit command"** → Reinstall globally with `npm install -g opencommit`.

43.2. Install `opencommit` on Windows

Overview

OpenCommit is a tool that uses AI (like GPT models) to automatically generate meaningful Git commit messages based on your code changes.

This guide explains how to install and configure OpenCommit on Windows step by step.

⚙ Prerequisites

- Before installing OpenCommit, make sure you have:
  - Git installed 👉 Download and install from https://git-scm.com/download/win
- Verify installation by running in PowerShell or CMD: `git --version`
- Node.js and npm (Node Package Manager) installed 👉 Download from https://nodejs.org (LTS version recommended).
  - Verify:

```
node -v
npm -v
```

- An OpenAI API key (for GPT access) 👉 Get it from https://platform.openai.com/account/api-keys

---

🪄 Installation Steps

1. Install OpenCommit Globally

- Open a terminal (PowerShell or CMD) and run: `npm install -g opencommit`
- Verify installation: `opencommit --version` If you get an error like “opencommit not recognized”, restart your terminal or ensure your npm global path is in your system PATH environment variable.

2. Configure the API Key

- Set your OpenAI API key as an environment variable permanently:
  - Press Win + R, type SystemPropertiesAdvanced, and press Enter.
  - Click Environment Variables...
  - Under User variables, click New, and add:
    - Variable name: OPENAI_API_KEY
    - Variable value: your_api_key_here
    - You can also customize OpenCommit’s behavior by setting additional environment variables, for example:

```
OCO_LANG="en"            # or "de", "fr", etc.
OCO_MODEL="gpt-4o"       # or another model like "gpt-4-turbo"
OCO_PROMPT_MODULE="conventional"
OCO_EMOJI=true
```

    - Click OK on all windows.
- Then restart PowerShell.

3. (Optional) Configure Defaults

- You can set up default parameters in your Git repository or globally: `opencommit config`
  - Follow the interactive prompts to define:
    - Default model (e.g., gpt-3.5-turbo)
    - Commit style
    - Language

🚀 Usage

- Once installed, simply run: `opencommit`. This will:
  - Analyze your staged changes (git diff --cached)
  - Generate an AI-powered commit message
  - Ask for confirmation before committing
- You can also use: `opencommit --no-verify` to skip confirmation and commit directly.

🔧 Troubleshooting

- Issue Solution
  - opencommit: command not found Ensure npm global bin path is added to your Windows PATH (e.g., C:\Users<YourUser>\AppData\Roaming\npm).
  - Error: Missing OPENAI_API_KEY Set your OpenAI API key as shown above.
  - Model too slow / API errors Try setting a smaller model: opencommit config --model gpt-3.5-turbo.

☑ Example

```
git add .
oco
```

Output:

```
Generated commit message:
...
```

(back to top)

44. Trice Maintenance

44.1. Trice Project structure (Files and Folders)

Trice Root Folder File	Details
.clang-format	See GitHub Action clang-format.yml - Check C Code Formatting
.clang-format-ignore	See GitHub Action clang-format.yml - Check C Code Formatting
.code_snippets	Some legacy helper code for copying where to use
.editorconfig	See GitHub Action clang-format.yml - Check C Code Formatting
.git	version control data base
.gitattributes	See GitHub Action clang-format.yml - Check C Code Formatting
.github/	🗀 The .github Folder — Purpose and Contents
.gitignore	git ignores these files
.goreleaser.yaml	goreleaser configuration
.idea/	GoLand settings
lychee.toml	GitHub Action link-check.yml - Broken Links Check
.markdownlinkcheck.json	GitHub Action link-check.yml - Broken Links Check
.markdownlint.yaml	Cleaning the Sources
.markdownlintignore	Cleaning the Sources
.vscode/	vsCode settings

Trice Root Folder File	Details
AUTHORS.md	contributors
CHANGELOG.md	History
CODE_OF_CONDUCT.md	How to communicate
CONTRIBUTING.md	Helper
LICENSE.md	MIT
README.md	Github first page
_config.yml	jekyll configuration
_test	automatic target code tests
buildTriceTool.sh	Build Trice tool from Go sources
build_environment.sh	see inside
clang-format.sh	See GitHub Action clang-format.yml - Check C Code Formatting
clean-dsstore.sh	Ru to remove MacOS artifacts
`coverage.out`	Go test coverage output
cmd/_cui/	(do not use) command user interface tryout code
cmd/_stim/	(do not use) target stimulation tool tryout code
cmd/clang-filter	ReadMe
cmd/trice	Trice tool command Go sources
demoLI.json	location information example
demoTIL.json	Trice ID list example
`dist/`	local distribution files folder created by goreleaser
docs	documentation folder with link forwarding
examples/	example target projects
format-dumeng-toc.sh	Trice User Manual Maintenance (or any `*.md` file)
gitAddWorktreeFromGitLogLineData.sh	helper to get easy a git worktree folder from any git hash for easy folder compare, see inside
gitAddWorktreesBetween.sh	helper to get easy git worktree folders from any time range
gitLogWithBranches.sh	helper to get easy a history view
go.mod	Go modules file
go.sum	Go modules sums
index.md	Jekyll index site for RERADME.md
internal/	Trice tool internal Go packages
pkg/	Trice tool common Go packages
renewIDs_in_examples_and_test_folder.sh	renew all ID data
src/	C sources for trice instrumentation -> Add to target project
`temp/`	binary logfiles could appear here
testAll.log	Accumulated output of last `./testAll.sh` run
testAll.sh	run all tests
third_party/	external components

Trice Root Folder File	Details
trice_cleanIDs_in_examples_and_test_folder.sh	Cleaning the Sources Activating the Trice Cache
trice_environment.sh	Cleaning the Sources Activating the Trice Cache
trice_insertIDs_in_examples_and_test_folder.sh	Cleaning the Sources Activating the Trice Cache

(back to top)

44.2. 🗀 The .github Folder — Purpose and Contents

GitHub automatically recognizes and uses everything contained inside the .github/ directory. This folder defines how the project behaves on GitHub: issue templates, automated workflows, labels, code scanning, greetings, and release automation. Details:

44.2.1. 🗀 `.github` Root

It contains issue templates, labels, workflow automation, code scanning, linting, and the CI/CD release pipeline.

- ISSUE_TEMPLATE/ Used to create structured bug reports and feature requests.
- FUNDING.yml Enables the GitHub “Sponsor” button.
- labeler.yml configuration file consumed by actions/labeler. It defines the labeling rules.
  - Because labeler.yml is configuration and not an executable workflow, it is placed directly under .github/, not under .github/workflows/.
  - The actions/labeler action expects .github/labeler.yml as the default configuration path, which is why this layout is standard and correct.
  - This is intentional and follows GitHub Actions conventions.
    - .github/workflows/label.yml is a workflow. It defines when and how the GitHub Action runs (triggers, permissions, runner, action reference).
    - .github/labeler.yml is not a workflow. It is a configuration file consumed by actions/labeler. It defines the labeling rules.
- [workflows/]](https://github.com/rokath/trice/blob/main/workflows/) Contains GitHub Actions automation

44.2.2. 🗁 `.github/workflows` — GitHub Actions Workflows

The .github/workflows/ folder contains YAML descriptions for various actions, which will be triggered automatically on certain events or are started manually. Every *yml* file in this directory defines an automated process. These processes run on GitHub’s servers (CI/CD).

- README.md: Documentation specifically for the /workflows folder.
  - Can be useful for contributors who want to understand or modify CI behaviors.
- **Additional Subdirectories**
  - icons/: Stores custom icons used inside workflows (e.g., for badges or reporting).
    - Example: go.svg — used in the Go workflow or README badges.
  - properties/: Contains metadata files used by GitHub for configuration purposes such as:
    - enabling/disabling features
    - controlling workflow permissions
    - defining workflow categories for the Actions UI

These workflows run automatically on pushes and pull requests to main, and can also be triggered manually via the GitHub Actions UI. These files are not executed; they simply inform GitHub how certain workflows behave or should be displayed.

Github Action	About
clang-format.yml	GitHub Action clang-format.yml - Check C Code Formatting
codeql.yml	GitHub Action codeql.yml - Static Code Analysis
coverage.yml	GitHub Action coverage.yml - Test Coverage and Coveralls Integration
go.yml	GitHub Action go.yml - Building and Testing Go Code
goreleaser.yml	GitHub Action goreleaser.yml - Build & Pack Trice Distribution
greetings.yml	GitHub Action greetings.yml - Greeting Message
label.yml	GitHub Action label.yml - Automatic Labeling Rules

Github Action	About
learn-github-actions.yml	GitHub Action learn-github-actions.yml - Instructional Workflow
link-check.yml	GitHub Action link-check.yml - Broken Links Check
manual.yml	GitHub Action manual.ym - To Be Triggered Manually
shellcheck.yml	GitHub Action shellcheck.yml - Catching Common Bash Scripts Bugs
shfmt.yml	GitHub Action shfmt.yml - Ensure Consistent Shell Scripts Formatting
stale.yml	GitHub Action stale.yml - Automatic Stale Issue Handling
superlinter.ym	GitHub Action superlinter.yml - Ensure Consistent YAML and Markdown Formatting
pages.yml	Github Action pages.yml - Creates The Trice Github Pages
test_goreleaser.yml	Github Action test_goreleaser.yml - Checks If Goreleaser Would Succeed

44.2.3. GitHub Action clang-format.yml - Check C Code Formatting

- **Local Action (developer machine):** ./clang-format.sh - adjust all C files excluding .clang-format-ignore according rule set in .clang-format.

> The file ./clang-format.sh is used to auto-format the Trice code.
>
> File .clang-format
>
> *Contributor: Sazerac4*
>
> Sazerac4 commented Aug 29, 2024: I have a code formatter when I make changes to my application but I would like to keep the style of the library when modifying. I couldn't find a code formatter, is there a tool used? If not, I propose this to provide one as an example by using clang-format.
>
> ```
> ## I have created a default style :
> clang-format -style=llvm -dump-config > .clang-format
> ## Then format the code:
> find ./src  -name '*.c' -o  -name '*.h'| xargs clang-format-18 -style=file -i
> ```
>
> The style of the example does not correspond to the original one. Configurations are necessary for this to be the case. Tags can be placed to prevent certain macros from being formatted
>
> ```
> int formatted_code;
> // clang-format off
>     void    unformatted_code  ;
> // clang-format on
> void formatted_code_again;
> ```
>
> I have tuned some settings for clang-format :
>
> ```
> * IndentWidth: 4  // original code size indentation
> * ColumnLimit: 0  // avoid breaking long line (like macros)
> * PointerAlignment: Left  // like original files (mostly)
> ```
>
> With preprocessor indentation, the result is a bit strange in some cases. It's possible with the option IndentPPDirectives (doc).
>
> Staying as close as possible to a default version (LLVM in this case) makes it easier to regenerate the style if necessary.
>
> See also: https://github.com/rokath/trice/pull/487#issuecomment-2318003072
>
> File .clang-format-ignore:

> *Contributor: Sazerac4*
>
> Sazerac4 commented Aug 30, 2024: I have added .clang-format-ignore to ignore formatting for specific files
>
> File .editorconfig:
>
> *Contributor: Sazerac4*
>
> The `.editorconfig` file allows to better identify the basic style for every files. (endline, charset, ...). It is a file accepted by a wide list of IDEs and editors : link This addition is motivated by forgetting the end of line in the .gitattributes file.
>
> File .gitattributes
>
> *Contributor: Sazerac4*
>
> With the `.gitattributes` file avoid problems with "diff" and end of lines. Here is an article that presents the problem.
>
> To fill the `.gitattributes`, I used the command below to view all the extensions currently used.
>
> ```
> git ls-tree -r HEAD --name-only | perl -ne 'print $1 if m/\.([^.\/]+)$/' | sort -u
> ```

- **Github Action (Continuous Integration):** .github/workflows/clang-format.yml does not format, it only checks.

44.2.4. GitHub Action codeql.yml - Static Code Analysis

- This workflow runs CodeQL, GitHub's static code analysis tool. Purpose:
  - scan the codebase for potential security vulnerabilities
  - detect unsafe code patterns
  - provide security alerts in the "Security" tab
  - Runs automatically on pushes and pull requests.
- **Github Action (Continuous Integration):** .github/workflows/codeql.yml
  - This workflow configures **GitHub CodeQL code scanning** for the repository.
  - This workflow continuously scans the codebase for security and quality problems using CodeQL.
  - It performs **static code analysis** on source code.
  - It looks for:
    - Security vulnerabilities (e.g. injection flaws, unsafe API usage)
    - Common programming errors
    - Code quality issues
  - It runs automatically:
  - On pushes to `main`
  - On pull requests targeting `main`
  - Manually via the GitHub Actions UI
  - The results are uploaded to GitHub and appear under **Security → Code scanning alerts**.
  - The "QL" is the same concept as SQL, but specialized for code analysis.

44.2.5. GitHub Action coverage.yml - Test Coverage and Coveralls Integration

Trice uses Go's built-in coverage tooling to measure how much of the Go codebase is exercised by automated tests.

- **Local Action (developer machine):**

Action	Command
Generate a coverage profile locally	`go test ./... -covermode=atomic -coverprofile=coverage.out`
Show results as list in terminal	`go tool cover -func=coverage.out`
Show results colored file specific in browser	`go tool cover -html=coverage.out`

- **Github Action (Continuous Integration):**
  On GitHub, the workflow `.github/workflows/coverage.yml` runs automatically for every pull request and also on a monthly schedule. The workflow:
  1. executes `go test` with coverage enabled,

2. prints a coverage summary in the CI logs,
3. uploads the raw `coverage.out` file as a workflow artifact, and
4. publishes the coverage results to Coveralls (if configured).

Coverage badge:
The README displays the current coverage status for the default branch using the Coveralls badge:

```
[![Coverage Status](https://coveralls.io/repos/github/rokath/trice/badge.svg?branch=master)]
(https://coveralls.io/github/rokath/trice?branch=master)
```

This badge is updated whenever the CI workflow successfully uploads a new coverage report for the `master` branch.

44.2.6. GitHub Action go.yml - Building and Testing Go Code

- A workflow for building and testing Go code. It runs GitHub CodeQL for Go and C (cpp language pack), detecting security vulnerabilities and code issues.
  - Typically includes steps such as:
  - setting up the Go toolchain
  - checking out the repository
  - compiling the project
  - running unit tests
- **Local Action (developer machine):** `go test ./...` or better `./testAll.sh full` (takes long)
- **Github Action (Continuous Integration):** .github/workflows/go.yml

44.2.7. GitHub Action goreleaser.yml - Build & Pack Trice Distribution

This workflow runs GoReleaser, the tool that builds and packages Trice for distribution.

- Purpose:
  - create release binaries for all supported platforms
  - generate archives (ZIP, tar.gz, etc.)
  - compute checksums
  - create a GitHub Release with all artifacts
- Triggers:
  - manually from the GitHub UI (“Run workflow”)
  - automatically when pushing a tag matching v* (e.g., v0.44.0)
  - This is the workflow responsible for generating official Trice releases.
- **Local Action (developer machine):** `goreleaser`
- **Github Action (Continuous Integration):** .github/workflows/goreleaser.yml

See also Trigger a **real** Trice release via CI (with `git tag`)

44.2.8. GitHub Action greetings.yml - Greeting Message

A small automation that posts a friendly greeting message when somebody:

- opens their first issue
- opens their first pull request
- Used to make new contributors feel welcome.

44.2.9. GitHub Action label.yml - Automatic Labeling Rules

- Defines automatic labeling rules for issues and PRs.
  - For example, files in certain directories may automatically get category labels.
  - This helps maintainers classify submissions more easily.
- **Github Action (Continuous Integration):** .github/workflows/label.yml

44.2.10. GitHub Action learn-github-actions.yml - Instructional Workflow

An instructional workflow provided by GitHub. Purpose:

- demonstrate basic GitHub Actions usage
- serve as documentation or a teaching example for new contributors

- It does not affect the Trice build or release process.

44.2.11. GitHub Action link-check.yml - Broken Links Check

- **Local Action (developer machine):** (deprechiated) `markdown-link-check ./docs/TriceUserManual.md`
  - Ignore patterns: .markdownlinkcheck.json
  - Info: `[/] #%F0%9F%93%82-%60.github/workflows%60-%E2%80%94-github-actions-workflows` = skipped check
- **Local Action (developer machine):** `lychee .`
  - Uses .lychee.toml as configuration
- **Github Action (Continuous Integration):** .github/workflows/link-check.yml

44.2.12. GitHub Action manual.ym - To Be Triggered Manually

A workflow that is designed to be triggered manually (similar to *workflow_dispatch* workflows). Common use cases:

- executing maintenance tasks
- running scripts on demand
- testing workflow behavior without making a commit
- This workflow does not run automatically.

44.2.13. GitHub Action shellcheck.yml - Catching Common Bash Scripts Bugs

Runs ShellCheck on all *.sh files, catching common bugs in Bash scripts.

- **Github Action (Continuous Integration):** .github/workflows/shellcheck.yml

44.2.14. GitHub Action shfmt.yml - Ensure Consistent Shell Scripts Formatting

Runs shfmt in diff mode on pull requests to ensure consistent formatting of shell scripts.

- **Local Action (developer machine):** `go test ./...` or better `./testAll.sh full` (takes long)
- **Github Action (Continuous Integration):** .github/workflows/shfmt.yml

44.2.15. GitHub Action stale.yml - Automatic Stale Issue Handling

Automates stale issue handling. Function:

- marks inactive issues or PRs as “stale”
- closes them after a configurable time period if there is no further activity
- eps the issue tracker manageable.

Mark stale issues and pull requests

- **Github Action (Continuous Integration):** .github/workflows/stale.yml

44.2.16. GitHub Action superlinter.yml - Ensure Consistent YAML and Markdown Formatting

- **Local Action (developer machine):** `markdownlint .`
- Runs GitHub Super Linter, a powerful linting suite. Purpose:
  - ensure consistent code formatting
  - detect stylistic issues
  - catch potential errors in supported languages
  - Helps maintain code quality across the entire repository.
- **Github Action (Continuous Integration):** .github/workflows/superlinter.yml
  - Checks YAML and Markdown files

44.2.17. Github Action pages.yml - Creates The Trice Github Pages

This workflow creates the Trice github pages avaliable under rokath.github.io/trice/.

- **Github Action (Continuous Integration):** .github/workflows/pages.yml

44.2.18. Github Action test_goreleaser.yml - Checks If Goreleaser Would Succeed

Test Goreleaser

- **Local Action (developer machine):** `goreleaser release --clean --snapshot --skip=publish`
- **Github Action (Continuous Integration):** .github/workflows/test_goreleaser.yml
  - Checks if Goreleaser would succeed.

44.3. Trice User Manual Maintenance (or any `*.md` file)

- Recommended Tool: VS Code with some extensions:
  - Markdown All in One (Yu Zhang)
  - Markdown Table Prettifier (Kristin Daroczi)
    - Click in mouse context menu the entry "Format Document" to adjust Markdown tables.
  - Markdown Preview Enhanced
    - Click the preview button in the top right.
  - Markdown TOC (dumeng)
    - Make sure to have `numbering=true` and place the following code to the place where the table of contend should be generated.

```
<!-- vscode-markdown-toc-config
  numbering=true
  autoSave=true
  /vscode-markdown-toc-config -->
<!-- /vscode-markdown-toc -->
```

    - Use Shift-Command-P and select `markdownTOC:generate` to get the TOC with automatic numbering.
      - If you do not place any configuration, the TOC is build at the document start.
    - There are other TOC generators, but this one creates numbers for all headlines.
    - We need to tweak the TOC a bit afterwards with `./format-dumeng-toc.sh`, which also exhanges `<a name` with `<a id`.
  - Markdown Paste (telesoho)
    - Helpful to get web site content preformatted as Markdown. Use mouse context menu.
  - markdownlint (David Anson)
    - Uses .markdownlint.yaml as rule set and .markdownlintignore to ignore files.
    - You can also run `markdownlint .` in a terminal after installing the CLI tool. It uses the same control files.
    - The Github action superlinter.yml uses .markdownlint.yaml and the same config files.
  - Markdown PDF (yzane)
    - Use Shift-Command-P "markdown PDF:export" to generata a PDF
    - page break for PDF generation: `<div style="page-break-before: always;"></div>`

44.4. Cleaning the Sources

In Github are some Actions defined. Some of them get triggered on a `git push` and perform some checks. To get no fail, some scripts should run before committing:

- `npx markdownlint *.md` (or just `markdownlint`) - uses .markdownlint.yaml to allow exeptions.
  - `npx markdownlint ./docs/TriceUserManual.md 2>&1 | awk '!seen[$2]++'` for example to reduce message count in case of errors.
- ./clean-dsstore.sh - remove MacOS maintenance data.
- ./trice_cleanIDs_in_examples_and_test_folder.sh removes all IDs.

(back to top)

45. Build and Release the Trice Tool

45.1. Build Trice tool from Go sources

- Install Go.
- Run:

```
ms@DESKTOP-7POEGPB MINGW64 /c/repos/trice (main)
$ bash ./buildTriceTool.sh # does internal do go install ./cmd/trice/...
```

- Afterwards you should find an executable `trice` inside `~/go/bin`.
- Extend PATH variable with `~/go/bin` **OR** copy the Trice binary from there into a folder of your path.
- Check:

```
ms@PaulPCWin11 MINGW64 ~/repos/trice (main)
$ ./buildTriceTool.sh
----------------------------------------
Building trice with embedded Git metadata:
  origin:     git@github.com:rokath/trice.git
  branch:     main
  version:    branch dirty
  commit:     f7edcc51
  date:       2025-11-27T13:59:50+01:00
  git_state:  dirty
  git_status:  M .vscode/launch.json  M docs/TriceUserManual.md  M internal/emitter/lineComposer.go
----------------------------------------
Build complete.

ms@PaulPCWin11 MINGW64 ~/repos/trice (main)
$ trice version
no version, branch=git@github.com:rokath/trice.git - main (local modifications at build time),
commit=f7edcc51,   built at 2025-11-27T13:59:50+01:00

ms@PaulPCWin11 MINGW64 ~/repos/trice (main)
```

- Hints
  - Use only the main branch. Other branches may be inconsistent.
  - When using ./buildTriceTool.sh, the generated Trice image is significant smaller (about 30%), because the build script removes debugging information from the Trice binary. The Trice release images contain this additionally information for a more verbose error reporting, just in case.
  - Give each Trice binary its own name when using different images. Otherwise you always get what is found first in the **$PATH**.
  - Use Goreleaser if you wish to create releases on your forked Trice repository.
  - On Windows you need to install TDM-GCC if you wish to execute the CGO tests as well.
    - Take the 64-bit variant when Go is 64-bit or take the 32-bit variant when Go is 32-bit. If mixed installations work I doubt.
    - Recommendation: Minimal online installer.
    - GCC is only needed to test the target C-code on the host.
    - Make sure TDM-GCC is found first in the path, if you have several compilers installed.
    - Other gcc variants could work also but not tested.
  - Open a console inside the Trice directory, recommended is the git-bash, when using Windows.
  - Tests:

```
ms@DESKTOP-7POEGPB MINGW64 /c/repos/trice (main)
$ go clean -cache

ms@DESKTOP-7POEGPB MINGW64 /c/repos/trice (main)
$ go vet ./...

ms@DESKTOP-7POEGPB MINGW64 /c/repos/trice (main)
$ go test ./...
?       github.com/rokath/trice/cmd/cui [no test files]
ok      github.com/rokath/trice/cmd/stim        0.227s
ok      github.com/rokath/trice/cmd/trice       0.577s
ok      github.com/rokath/trice/internal/args   0.232s
ok      github.com/rokath/trice/internal/charDecoder    0.407s
ok      github.com/rokath/trice/internal/com    1.148s
```

```
ok      github.com/rokath/trice/internal/decoder        0.412s [no tests to run]
?       github.com/rokath/trice/internal/do     [no test files]
ok      github.com/rokath/trice/internal/dumpDecoder    0.388s
ok      github.com/rokath/trice/internal/emitter        0.431s
ok      github.com/rokath/trice/internal/id     0.421s
ok      github.com/rokath/trice/internal/keybcmd        0.431s
ok      github.com/rokath/trice/internal/link   0.404s
ok      github.com/rokath/trice/internal/receiver       0.409s
ok      github.com/rokath/trice/internal/tleDecoder     0.398s
?       github.com/rokath/trice/internal/translator     [no test files]
ok      github.com/rokath/trice/internal/trexDecoder    0.391s
ok      github.com/rokath/trice/pkg/cipher      0.377s
ok      github.com/rokath/trice/pkg/endian      0.302s
ok      github.com/rokath/trice/pkg/msg 0.299s
ok      github.com/rokath/trice/pkg/tst 0.406s
```

To execute the target code tests, you can run `testAll.sh` or `cd` into `_test` and run `go test ./...` from there. ATTENTION: These tests run a significant long time (many minutes depending on your machine), because the **Go** - **C** border is crossed very often. The last tests can last quite a while, depending on your machine.

```
ms@DESKTOP-7POEGPB MINGW64 /c/repos/trice (main)
$ go install ./cmd/trice/
```

Afterwards you should find an executable `trice` inside $GOPATH/bin/ and you can modify its source code.

After installing Go, in your home folder should exist a folder ./go/bin. Please add it to your path variable. OR: Copy the Trice binary from there into a folder of your path after creating it with `go install ./cmd/trice/...`. There is now a remommended script `./buildTriceTool.sh`. Using it, depending on your system you may need to enter `bash ./buildTriceTool.sh`, includes the actual Trice repository state into the Trice binary, which is shown with `trice version` then - useful in case of issues.

45.2. Prepare A Release

Prerequisite: Installed `goreleaser`.

45.2.1. Check a Goreleaser Release before Publishing

By cloning the Trice repo into an empty folder, you make sure no other files exist in the Trice folder.

```
mkdir ./tmp
cd /tmp
git clone https://github.com/rokath/trice.git
cd trice
goreleaser release --clean --snapshot --skip=publish
```

This just generates the artifacts locally in `/tmp/trice/dist` using the ./trice/.goreleaser.yaml copy in `./temp/trice`.

Alternatively you can do in your local Trice clone directly, by removing everything not a part of the Trice repo, if you are sure not to loose any data:

```
git status
git clean -xfd
goreleaser release --clean --snapshot --skip=publish
```

Explanation:

- `release` – run the normal release pipeline (builds, archives, checksums, changelog, etc.).
- `--clean` – delete the `dist/` folder first so no old artefacts are reused.
- `--snapshot` – build as a “snapshot” version, not a real tagged release.
- `--skip=publish` – **do not upload** anything (no GitHub Releases, no Homebrew, no Docker, etc.).

What you should see:

- GoReleaser building all your `builds:` targets from `.goreleaser.yaml`.
- Creating archives in `dist/`.
- Generating checksums.
- No attempts to call the GitHub API for a real release.

If this **succeeds**, you've already tested 90% of what CI will do for a real release. If it **fails**, fix the problem locally first (missing files, bad paths, etc.) – it would fail the same way in CI.

45.3. Trigger a **real** Trice release via CI (with `git tag`)

Letting CI build and publish an **official release**.

45.3.1. Make sure your workflow reacts to tags

In .github/workflows/goreleaser.yml, you need `on: workflow_dispatch: push: tags: - 'v*'`.

- `workflow_dispatch` = you can still run it manually from the Actions tab.
- `push -> tags: 'v*'` = whenever you push a tag like `v0.44.0`, this workflow will start automatically.

Commit & push this change (if you haven't already):

```
git add .github/workflows/goreleaser.yml git commit -m "Configure GoReleaser workflow to run on tags" git push origin main
```

45.3.2. Final checks before tagging

In your local `trice` repo:

- Update to latest main:
  - `git checkout main git pull origin main`
  - Run your tests:
  - `go test ./...` or better `./testAll.sh full`
  - Optional but recommended: **run the snapshot dry run** again, just to be safe:
- `goreleaser release --clean --snapshot --skip=publish`

If all of that is green, you're ready to "bless" a version.

45.3.3. Choose a version and create a `git tag`

Decide on a version, for example:

- `v0.44.0`
- `v1.0.0`
  (Important: GoReleaser expects **SemVer-style tags** like `vX.Y.Z`.)

Create an **annotated tag**:

```
git tag -a v0.44.0 -m "Trice v0.44.0"
```

Check your tags:

```
git tag
```

You should see `v0.44.0` in the list.

> The **tag** is what GoReleaser uses as the release version (`.Tag`, `.Version`, etc.) in your `.goreleaser.yaml`.
> Your `ldflags` like `-X main.version={{ .Version }}` will use this.

45.3.4. Push the tag to GitHub (this triggers CI)

Now push the tag:

```
git push origin v0.44.0
```

This does **not** push all tags, only `v0.44.0`.

Because of your workflow's `on: push: tags: 'v*'`, this **automatically starts** the GoReleaser workflow in GitHub Actions.

45.3.5. Watch the CI release run on GitHub

1. Open your browser and go to your repo:

   `https://github.com/rokath/trice`

2. Click the **"Actions"** tab at the top.

3. In the list of workflows, click on **"goreleaser"**.

4. You should see a new run with something like:

- Event: `push`
- Ref: `refs/tags/v0.44.0`
- Status: in progress → green (hopefully)

5. Click on that run, then on the job (e.g. `goreleaser`):

- Step "Check out repository"
- Step "Set up Go"
- Step "Run GoReleaser"

If "Run GoReleaser" is green ✔, the CI release has succeeded.

45.3.6. Check the GitHub Release

Finally, verify the published release:

1. In your repo, click the **"Releases"** section (right side or under "Code").

2. You should see a new release **`v0.44.0`** created by GoReleaser.

3. Inside it you'll find:

- the generated archives (tar.gz/zip),
- checksums,
- etc., as defined in your `.goreleaser.yaml`.

This is now your **official Trice release built by CI**.

(back to top)

46. Scratch Pad

(back to top)