

# Percepio Detect™ Demo Guide

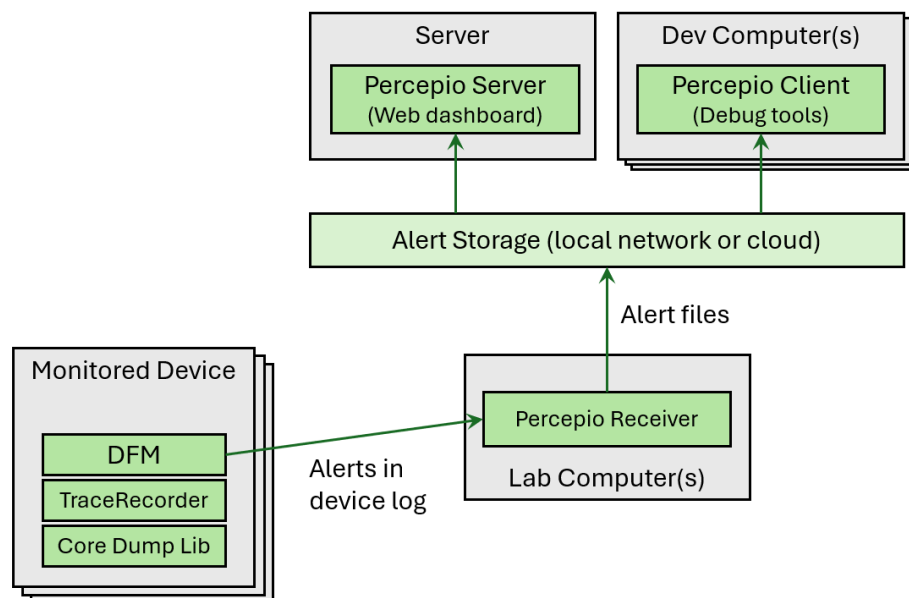
Version 2025.2, for Windows hosts

Percepio Detect™ provides Continuous Observability for embedded software during development, integration testing (CI/CT) and field validation. This is designed for “always on” monitoring, capturing runtime issues automatically at the first signs of trouble. This offers continuous on-device monitoring without needing continuous data transmission. Data is only emitted when a runtime issue is detected in the device software, minimizing the data volume.

- Avoid the pains of issue reproduction with detailed debugging data from the original issue, turning nightmare bugs into quick fixes.
- Detect reliability risks like “near misses” and multi-threading issues.
- Get easy access to incident debugging data for the whole team.
- Debug on production boards without debug ports. A basic UART is sufficient.
- Monitor devices in the field, for example during field validation.
- All device data and IP remain in your private network. No external cloud service.

## Solution Overview

Percepio Detect is designed as a multi-user solution for use in private development networks. The solution consists of four parts, as shown below. In the demo setup, both the Server and Client run on your local computer. The demo doesn’t require setting up the device-side DFM library or the Receiver since example alert files are included and loaded automatically.



- **Percepio DFM:** The main target-side component of Percepio Detect. This outputs "alerts" on faults and anomalies, including debugging data.

- **Detect Receiver:** Reads the device output, extracts the DFM data, converts it to the expected format and saves it as alert files for the Detect Server.
- **Detect Server:** Reads alert files from Receiver and presents a summary in the web browser dashboard. Provides easy access to debugging data from the alerts.
- **Detect Client:** An integrated set of developer tools for debugging alerts, including Tracealyzer and a core dump viewer. Runs on each user's local computer.

## Terminology

- **Alert:** A machine-readable "problem report" created by the DFM library on a device.
- **Alert Type:** The main type or reason for the alert, for example "Hard Fault".
- **Symptom:** An integer value providing additional characteristics for an alert, like the stack pointer or program counter.
- **Payload:** Debugging data provided by an alert, e.g. traces and core dumps.
- **Issue:** The group of alerts with same Symptoms and Alert Type (same "fingerprint").

## Preparation Steps

- Install [Docker Desktop](#).
- Make sure Python (v3.x) is installed.

## Running the Demo

After you have completed the preparation steps above, follow these steps to run the bundled demo. Note that the device integration is not required here, as the demo uses pre-recorded example alerts found in the test-data folder.

1. Start Docker Desktop. Make sure it states "Engine Running" in the bottom left corner.



2. Open the server start script, **perceprio-server.bat**, in a text editor and locate the assignment of the LICENSE variable. Update this with your Perceprio Detect license key and save the file.

```
set LICENSE="ABCD-ABCD-ABCD-ABCD"
```

3. Open a terminal in the **perceprio-server** folder and run:

```
.\perceprio-server.bat start
```

4. Navigate to the perceprio-client-windows folder and start the Client by running:

```
cd ..\perceprio-client-windows
.\perceprio-client.bat
```

Or, since this doesn't take any arguments, you can simply double-click on **perceprio-client.bat** in File Explorer.

perceprio-client-windows				
Namn	Senast ändrad	Typ	Storlek	
content	2024-12-07 11:31	Filmapp		
coredumpviewer	2024-12-07 11:31	Filmapp		
dispatcher	2024-12-07 11:31	Filmapp		
tracealyzer	2024-12-09 13:32	Filmapp		
core_dump_viewer.bat	2024-12-06 13:43	Windows-komma...	1 kB	
dispatcher-settings-windows.json	2024-12-06 13:45	JSON File	2 kB	
perceprio-client.bat	2024-12-06 12:37	Windows-komma...	1 kB	
perceprio-client.py	2024-12-06 12:43	Python File	5 kB	

Make sure the Client starts up without errors.

```
C:\Windows\system32\cmd.exe
Perceprio Detect Client, version 2025.2
Directory check: OK
Ready.
```

5. Open <http://127.0.0.1:8080> and check that you see the Server dashboard. It may take a few seconds for the Server to start up and load the demo data. The dashboard is updated every 5 seconds, but you may refresh the web browser manually for faster updates.

On the main page, the "Issue Overview" shows a summary of all alerts in the Server database.

Each row represents one "Issue", i.e. the group of alerts having identical Alert Type and Symptoms. This simplifies overview and analysis, in case of many reported alerts.

Issue Overview						
Description	Revision	Count	Latest Occurrence	Device Id	Symptoms	Payloads
Stack corruption detected	DemoSTM32L4-20250402	1	4/8/2025, 11:34:28 AM	StopwatchDemoSTM32L4-20250401	Current Task: 0 File: 1651 Line: 343 Stack Pointer: 536968864	<a href="#">dfm_trace.psfs</a> <a href="#">cc_coredump.dmp</a>
Stopwatch alert	DemoSTM32L4-20250402	2	4/8/2025, 11:34:28 AM	StopwatchDemoSTM32L4-20250401	Stopwatch ID: 1	<a href="#">dfm_trace.psfs</a>
Hard Fault	DemoSTM32L4-20250402	3	4/8/2025, 11:34:28 AM	StopwatchDemoSTM32L4-20250401	Current Task: 0 CFSR Register: 33280 Stack Pointer: 536968896	<a href="#">dfm_trace.psfs</a> <a href="#">cc_coredump.dmp</a>
Assert Failed	DemoSTM32L4-20250402	1	4/8/2025, 11:34:27 AM	StopwatchDemoSTM32L4-20250401	Current Task: 0 File: 1197 Line: 99 Stack Pointer: 536968896	<a href="#">dfm_trace.psfs</a> <a href="#">cc_coredump.dmp</a>

The most important columns are:

- **Revision:** shows the version of the device software that produced the alert.
- **Device Id:** An identifier of the device.
- **Symptoms:** The "fingerprint" of the issue.
- **Payloads:** Links to debugging data, pointing to the most recent alert of the issue.
- **Details:** Shows additional information for the most recent alert.

6. Let's begin with crash debugging. In the server dashboard, locate the "Hard Fault" row.

This might be on page 2. Try clicking the column labels such as Description or Latest Occurrence to change the sorting.

Once you found “Hard Fault,” click the “cc\_coredump.dmp” link in the Payloads column. This will display the selected core dump in the core dump viewer tool of the Client.

```
=== Call Stack =====
#0  0x080079d0 in transform_sensor_value (raw_value=162, scale_factor=1, offset=0) at C:\src\github-repos\Demo
    adjusted_divisor = 0
#1  0x08007a64 in demo_crash_alert () at C:\src\github-repos\Demos\UsageExamples\10_dfm_crash_alert.c:107
    result = 19
#2  0x08004f38 in vTaskDemoDriver (pvParameters=0x0) at C:\src\github-repos\Demos\STM32CubeDemos-B-L475E-IOT0
    demoToRun = 3
#3  0x080032ac in pxPortInitialiseStack (pxTopOfStack=<optimized out>, pxCode=<optimized out>, pvParameters=<
    ARM_CM4F\port.c:209
No locals.
Backtrace stopped: previous frame identical to this frame (corrupt stack?)

=== Source Code =====
#0  0x080079d0 in transform_sensor_value (raw_value=162, scale_factor=1, offset=0) at C:\src\github-repos\Demo
59     return (raw_value + offset) / adjusted_divisor;
54     if(scale_factor == 0)
55         return 0;
56
57     int adjusted_divisor = adjust_divisor(scale_factor);
58
59     return (raw_value + offset) / adjusted_divisor;
```

This core dump was triggered by a hard fault exception in the device and automatically reported as an Alert, with the core dump attached as a Payload.

The “Call Stack” section shows the function that failed (the top one), the function arguments and the prior function calls in the current thread. The “Source Code” section shows the source code at the fault location. The core dump viewer also shows other sections, like registers and disassembly.

The core dump solution is based on GDB and CrashCatcher, with Perceptio improvements to enable more compact core dumps. The examples in the demo are often below 500 bytes.

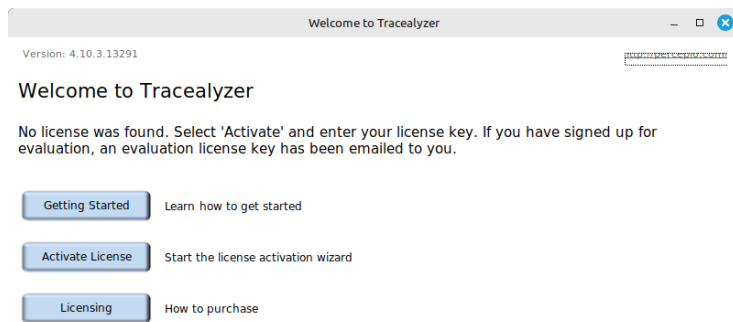
Core dumps are generated automatically on processor fault exceptions by the fault handler included in the DFM library, but can also be triggered by calling the DFM\_TRAP() macro from your application-level fault handling code. See also **Demo Alert Types** for more information.

**7.** Now let’s have a look at a Tracealyzer trace from the alerts. In the Dashboard, locate the “Stopwatch alert” row and click the “dfm\_trace.psfs” link.

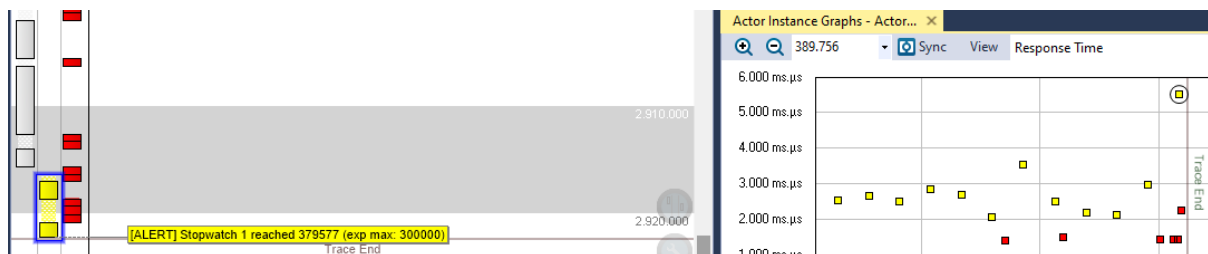
Stopwatch alert	2	4/8/2025, 3:01:52 PM	<a href="#">dfm_trace.psfs</a>	<button>View</button>	<button>Alerts</button>
-----------------	---	----------------------	--------------------------------	-----------------------	-------------------------

This will open the trace in the Tracealyzer tool, included in the Detect Client.

If the “Welcome to Tracealyzer” screen opens, you need to enter your license key. Select “Activate License” and **“Activate key online”**. Enter your Tracealyzer evaluation license key, which is provided in the same email as the Detect evaluation license.

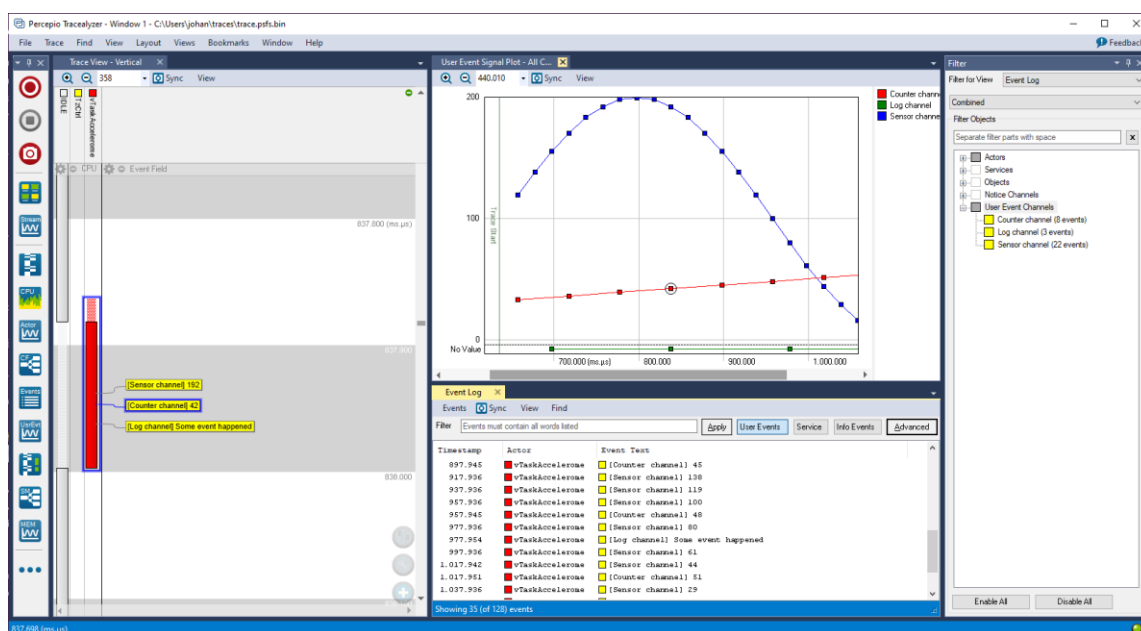


You should now see the trace file visualized in Tracealyzer. The trace data comes from the TraceRecorder library, that supports both RTOS kernel tracing and several kinds of application logging. If the trace doesn't open, check the Troubleshooting section below.



The tracing is based on kernel instrumentation and logging calls. The events are written to a circular RAM buffer. On alerts, the trace buffer is included as a payload and shows the most recent events before the alert. Tracealyzer traces can be included with all alert types.

The TraceRecorder library supports not only kernel tracing, but also offers extensive logging support, for example in the `trcPrint.h` API. Logging such events, *User Events*, has similar syntax as `printf` calls but are often orders of magnitude faster (microseconds). They appear as yellow labels, like below, useful for providing additional details for debugging.



## Troubleshooting

If Tracealyzer fails to open the trace with the error message “Failed to find handler for opening this trace. This could be...”, it can be due to Windows security restrictions on applications in downloaded Zip file. In that case, you can try one of the following methods:

- Method A: Right-click the downloaded zip file, select “Properties” and see if there is an “Unblock” option at the bottom of the General page. If present, check the checkbox to unblock the contents. Then extract the zip file again.
- Method B: If you already extracted the zip file, another solution is to run the powershell command “Get-ChildItem -Path . -Recurse | Unblock-File” in the Detect directory and then restart Tracealyzer.

## Device Demo Projects

The [Percepio Demos Github repository](#) provides documented examples for both Tracealyzer and Detect. Demo projects are provided for STM32CubeIDE, MPLAB X IDE and IAR Embedded Workbench. While only a few boards are supported so far, the documentation and code examples are quite valuable on their own, also without running the demo application.

The demo application runs a set of specific feature demos, designed to be simple and easy to overview. Each example is a single short .c file, corresponding to different alert types. Links to each specific example are provided in the following section.

## Demo Alert Types

### Fault Exceptions (Hard Fault Alert)

Crashes are often detected by the processor itself, for example if an invalid memory address is used or random data is executed as code. This triggers a fault exception handler that can be used to log diagnostic information, but this is not always used to its full potential. Percepio Detect can capture and report all types of Arm Cortex-M fault exceptions and provide extensive diagnostic information such as core dumps and TraceRecorder traces. This is enabled by installing the DFM fault handler (DFM\_Fault\_Handler) as handler for fault exceptions.

```
==== Call Stack =====
#0 0x08007ac8 in functionY (arg1=-1) at C:\src\github-repos\Demos\UsageExamples\l1_dfm_custom_alert.c:36
No locals.
#1 0x08007aea in functionX (a=-1, b=-3) at C:\src\github-repos\Demos\UsageExamples\l1_dfm_custom_alert.c:48
No locals.
#2 0x08007b4c in demo_custom_alert () at C:\src\github-repos\Demos\UsageExamples\l1_dfm_custom_alert.c:83
    ret = 4
#3 0x08004f3e in vTaskDemoDriver (pvParameters=0x0) at C:\src\github-repos\STM32CubeDemos-B-L475E-IOT01A\Projects\B-L475E-IOT01A\Demo\main.c:100
    demoToRun = 4
#4 0x080032ac in pxPortInitialiseStack (pxTopOfStack=<optimized out>, pxCode=<optimized out>, pvParameters=<optimized out>) at C:\src\gl
ARM_CM4F\port.c:209
No locals.
Backtrace stopped: previous frame identical to this frame (corrupt stack?)

==== Source Code =====
#0 0x08007ac8 in functionY (arg1=-1) at C:\src\github-repos\Demos\UsageExamples\l1_dfm_custom_alert.c:36
36     return -1;
37     {
38         /* Output an "alert" from DFM to Percepio Detect (core dump and trace).
39          * Third argument is if to restart (=1) or not (=0). */
40         DFM_TRAP(DFM_TYPE_ASSERT_FAILED, "Assert failed, arg1 negative", 0);
41     }
42     return -1;
43 }
```

See also the [example on Github](#).

Copyright (c) Percepio AB, 2025

<https://percepio.com>

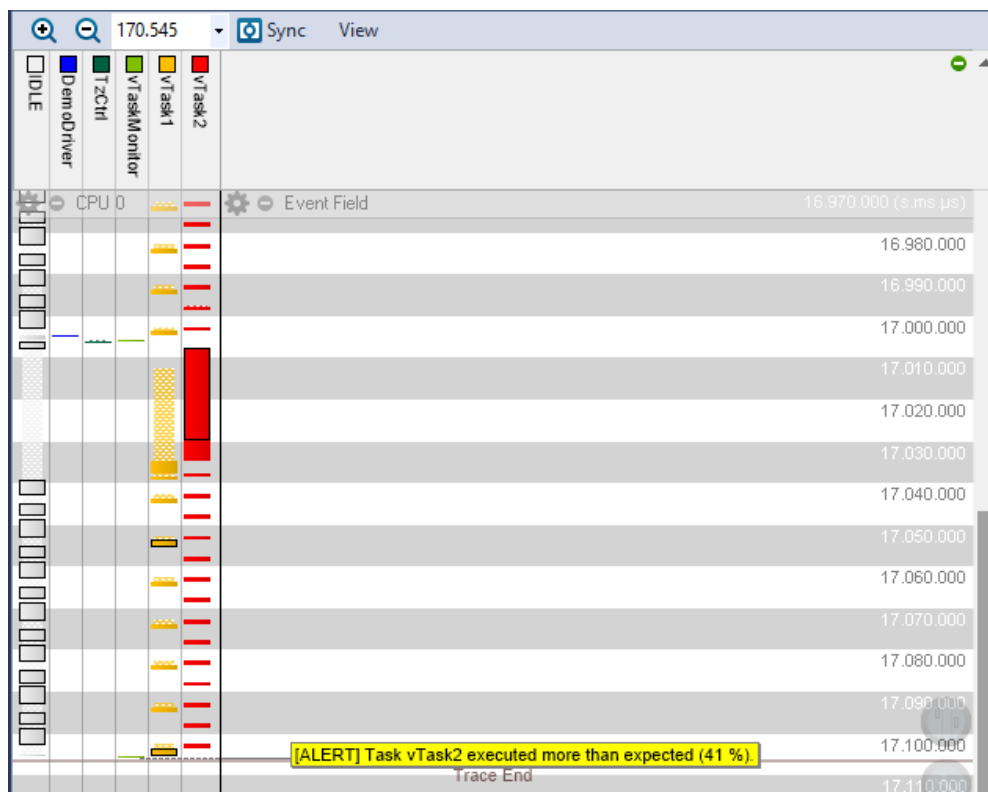
## TaskMonitor Alerts

The TaskMonitor feature in DFM lets you monitor the CPU time usage per thread, over unlimited time, even in the field, since DFM doesn't output data continuously but only at anomalies.

This is valuable for profiling and for verifying CPU usage budgets, and also allows for capturing elusive bugs by their side-effect on CPU time usage, for example if the device intermittently becomes unresponsive or has occasional performance issues. Issues like thread starvation, deadlocks or priority inversions can often be captured in this way, even before faults occur.

You configure the monitoring by calling `xDfmTaskMonitorRegister()` for each thread you like to monitor together with the upper and lower warning levels. You can select a single thread only, a subset of the threads or all threads if desired.

An alert with a TraceRecorder snapshot trace is produced if a monitored thread runs more than expected, or less than expected, over a certain time period. The time period is defined by periodic calls to the `xTraceTaskMonitorPoll()` function.



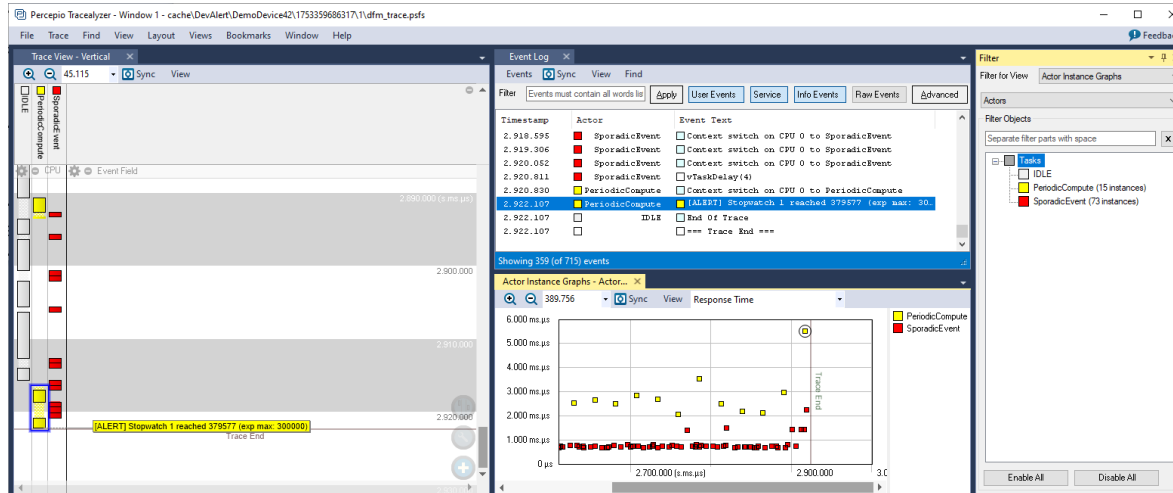
The TaskMonitor provides high and low watermarks for each thread that can be printed to the serial console using `xDfmTaskMonitorPrint()`. This is useful for basic profiling and for tuning the warning levels.

To avoid redundant alerts, alerts are only emitted if the CPU usage for a thread is above both the upper warning level and the previous high watermark, or below both the lower warning level and the previous low watermark. For example, if the warning levels allow for a CPU load between 10% and 20% and you occasionally get spikes at 80%, only the first spike will trigger an alert, unless later spikes are even higher than the previous ones.

See also the [example on Github](#).

## Stopwatch Alerts

Stopwatch alerts are triggered when the time between two points in the code is longer than expected. This can be used to monitor response time requirements and to detect issues by their side-effect on timing. This is particularly useful on RTOS issues, like thread starvation.



Stopwatch alerts are implemented by inserting DFM function calls at the starting point and end point of the relevant sequence. The highest expected runtime is specified when initializing the stopwatch. If exceeded, an alert is emitted in "End" function, but only if a new "high watermark" has been found. This avoids redundant stopwatch alerts.

Multiple stopwatches can be used in parallel and may span across different threads and interrupt handlers. To set the alert threshold, start with a high value (to avoid many alerts), run your tests and then inspect the High Watermark using the Stopwatch API.

See also the [example on Github](#).

## Stack Corruption Alerts

The DFM library includes support for compiler-added stack integrity checking. If using GCC or Clang, this requires using one of the [-fstack-protector](#) build flags. This adds a lightweight stack integrity check when returning from functions containing local buffers. If the stack is found to be corrupted, DFM will emit a "Stack corruption detected" alert at the exit of the function. Note that this compiler feature increases the code size a bit due to the added checks.

This feature is also supported for IAR Embedded Workbench by enabling the stack protection feature in the project options.

See also the [example on Github](#).



## Custom Alerts

Detect allows for defining your own alert types and reporting them programmatically from your code, for example in Assert statements or other error-handling code. The easiest way is to use the DFM\_TRAP macro, an easy one-liner for generating alerts containing a core dump with the stack trace, and optionally also a Tracealyzer trace.

### Example:

```
// Can be used in asserts like this:
// #define ASSERT(x, msg, ret) if(!(x)) { DFM_TRAP(DFM_TYPE_ASSERT_FAILED, msg, 0); return ret;}

int functionY(int arg1)
{
    if(arg1 < 0) // Or use an ASSERT macro like above, but this is clearer.
    {
        /*****
         * Output an "alert" from DFM to Percepio Detect (core dump and trace).
         * Third argument is if to restart (=1) or not (=0).
         *****/
        DFM_TRAP(DFM_TYPE_ASSERT_FAILED, "Assert failed, arg1 negative", 0);

        return -1;
    }
}
```

This generates a core dump on the same format as for fault exceptions. The example output is shown on the next page.

```
C:\Windows\system32\cmd.exe
0x08007ad6 <+58>:  movs    r0, r4
0x08007ad8 <+60>:  pop     {r4, pc}
End of assembler dump.

=== Call Stack ===
#0  0x08007ac8 in functionY (arg1=-1) at C:\src\github-repos\Demos\UsageExamples\11_dfm_custom_alert.c:36
No locals.
#1  0x08007aea in functionX (a=-1, b=-3) at C:\src\github-repos\Demos\UsageExamples\11_dfm_custom_alert.c:48
No locals.
#2  0x08007b4c in demo_custom_alert () at C:\src\github-repos\Demos\UsageExamples\11_dfm_custom_alert.c:83
    ret = 4
#3  0x08004f3e in vTaskDemoDriver (pvParameters=0x0) at C:\src\github-repos\Demos\STM32CubeDemos-B-L475E-IOT01A\Projects\B-L4
    demoToRun = 4
#4  0x080032ac in pxPortInitialiseStack (pxTopOfStack=<optimized out>, pxCode=<optimized out>, pvParameters=<optimized out>)
ARM_CM4F\port.c:209
No locals.
Backtrace stopped: previous frame identical to this frame (corrupt stack?)

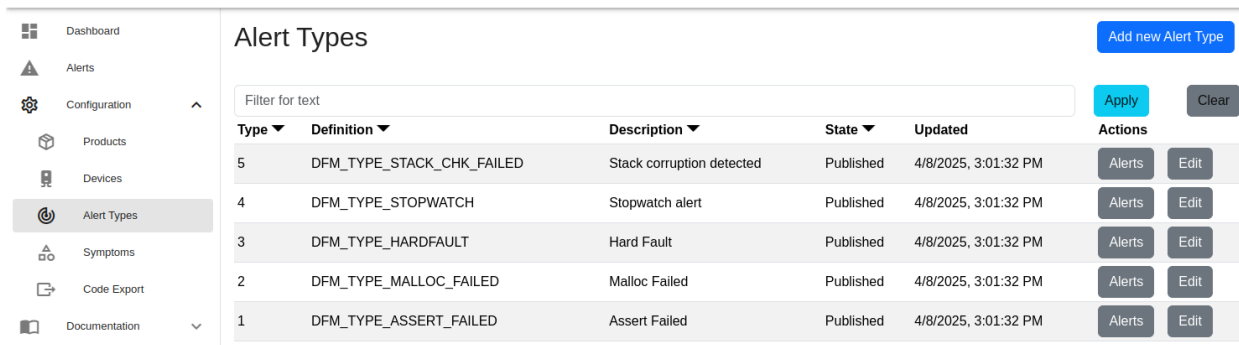
=== Source Code ===
#0  0x08007ac8 in functionY (arg1=-1) at C:\src\github-repos\Demos\UsageExamples\11_dfm_custom_alert.c:36
36     return -1;
37     {
38         /* Output an "alert" from DFM to Percepio Detect (core dump and trace).
39          * Third argument is if to restart (=1) or not (=0). */
40         DFM_TRAP(DFM_TYPE_ASSERT_FAILED, "Assert failed, arg1 negative", 0);
41
42         return -1;
43     }
44
45     xTracePrintf(demo_log_chn, "In functionY(%d)\n", arg1);
46
Press any key to continue . . .
```

See also the [example on Github](#).

## Adding New Alerts Types and Symptoms

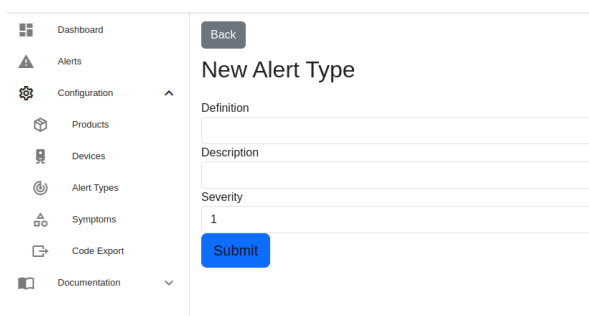
You may add additional alert types and symptoms using the Server dashboard, under "Configuration". Their names are stored in the Server database and is represented by an integer ID in on the device side for efficiency reasons, specified in the DFM library.

Open the **Alert Types** page. Note the "Status" column, where "Published" means that the numeric ID is locked and can't be deleted and reused with a different meaning. This avoids mismatching definitions in between firmware revisions.



Type	Definition	Description	State	Updated	Actions
5	DFM_TYPE_STACK_CHK_FAILED	Stack corruption detected	Published	4/8/2025, 3:01:32 PM	Alerts Edit
4	DFM_TYPE_STOPWATCH	Stopwatch alert	Published	4/8/2025, 3:01:32 PM	Alerts Edit
3	DFM_TYPE_HARDFAULT	Hard Fault	Published	4/8/2025, 3:01:32 PM	Alerts Edit
2	DFM_TYPE_MALLOC_FAILED	Malloc Failed	Published	4/8/2025, 3:01:32 PM	Alerts Edit
1	DFM_TYPE_ASSERT_FAILED	Assert Failed	Published	4/8/2025, 3:01:32 PM	Alerts Edit

To add a new entry, select "Add new Alert Type".



Back

New Alert Type

Definition

Description

Severity

1

Submit

- Definition: The name of the numeric ID in DFM. Must start with "DFM\_TYPE\_".
- Description: The display name shown in the dashboard (Issue Overview).
- Severity: (not yet implemented).

After adding new entries, go to the **Code Export** page to generate an updated **dfmCodes.h**.

This workflow ensures matching definitions on device and server. You can then create alerts using your new Alert Type, for example using DFM\_TRAP:

```
DFM_TRAP (DFM_TYPE_ALERT_XYZ, "Error XYZ", 0);
```

## Learning More

More information on how to set up and customize Percepio Detect is found in `readme.txt` in root of the Percepio Detect package. While it is not needed for just running the demo, make sure to read it before setting up your own Detect solution.

If you have questions or feedback, please [contact Percepio](#).

Copyright (c) Percepio AB, 2025

<https://percepio.com>