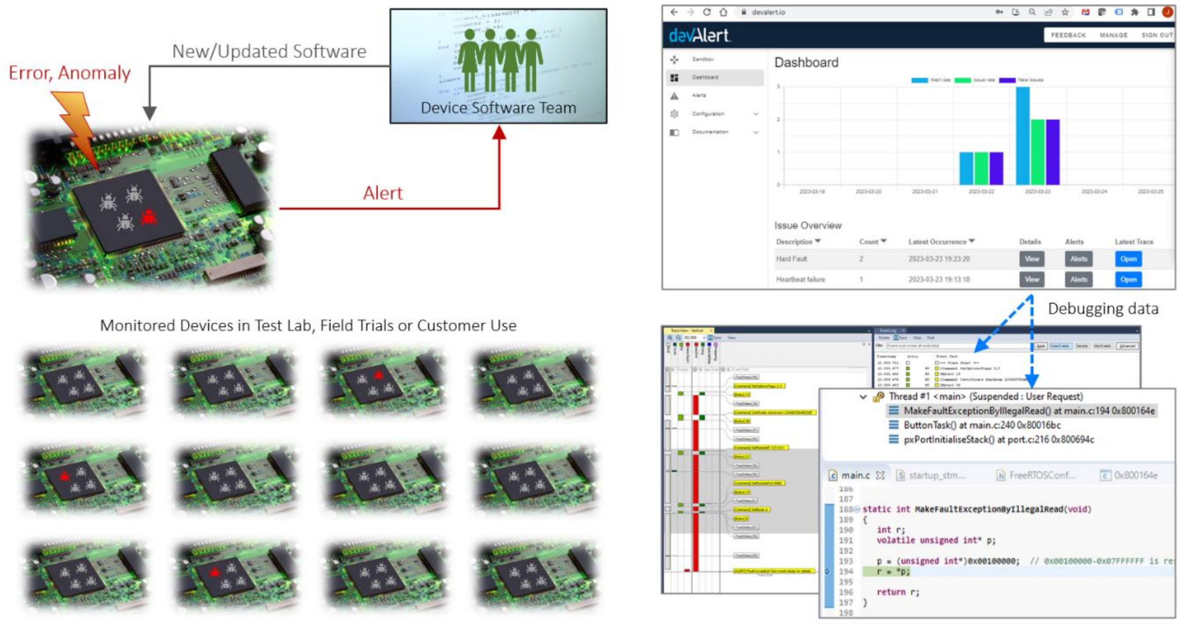
Getting Started with DevAlert on Zephyr

Percepio DevAlert is a cloud-based observability solution for edge devices and embedded software that lets you detect and analyze issues remotely during system testing, in field trials or in customer use. If your device software would encounter an unexpected situation or misbehave for other reasons, you will be notified right away and get all information needed to quickly solve the problem.

DevAlert uses a novel hybrid desktop/cloud architecture where all device data is easily available in the web browser while also letting you provide your own private data storage to ensure data control and privacy. And you can hook in your familiar desktop tools for remote debugging in a secure way.



Issue Overview

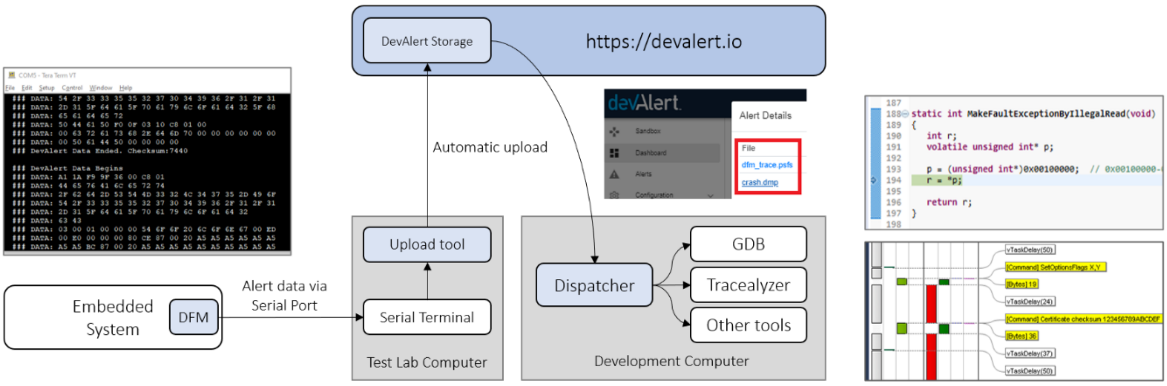
Description	Count	Latest Occurrence	Details	Alerts	Latest Trace
Hard Fault	2	2023-03-23 19:23:20	View	Alerts	Open
Heartbeat failure	1	2023-03-23 19:13:18	View	Alerts	Open

```
Thread #1 <main> (Suspended : User Request)
    MakeFaultExceptionByIllegalRead() at main.c:194 0x800164e
    ButtonTask() at main.c:240 0x80016bc
    pxPortInitialiseStack() at port.c:216 0x800694c
```

```
static int MakeFaultExceptionByIllegalRead(void)
{
   int r;
   volatile unsigned int* p;

   p = (unsigned int*)0x00100000;  // 0x00100000-0x07FFFFFF is re

   r = *p;

   return r;
}
```

DevAlert gives your team members a shared dashboard with remote access to diagnostic data, automatically collected in the devices on system errors and other anomalies. This may include core dumps, event traces and any other device data of interest.

This article explains how to set up Percepio DevAlert for monitoring local devices connected using serial port connections. This is useful as an initial setup for getting started with DevAlert and is also applicable for monitoring system testing on multiple devices in a test lab.



```
### DATA: 54 2F 33 33 35 35 32 37 30 34 39 36 2F 31 2F 31
### DATA: 2D 31 5F 64 61 5F 70 61 79 6C 6F 61 64 32 5F 68
### DATA: 65 61 64 65 72
### DATA: 50 44 61 50 F0 0F 03 10 C8 01 00
### DATA: 00 63 72 61 73 68 2E 64 6D 70 00 00 00 00 00 00
### DATA: 00 50 61 44 50 00 00 00 00
### DevAlert Data Ended. Checksum:7440

### DevAlert Data Begins
### DATA: A1 1A F9 9F 36 00 C8 01
### DATA: 44 65 76 41 6C 65 72 74
### DATA: 2F 62 64 2D 53 54 4D 33 32 4C 34 37 35 2D 49 6F
### DATA: 54 2F 33 33 35 35 32 37 30 34 39 36 2F 31 2F 31
### DATA: 2D 31 5F 64 61 5F 70 61 79 6C 6F 61 64 32
### DATA: 63 43
### DATA: 03 00 01 00 00 00 54 6F 6F 20 6C 6F 6E 67 00 ED
### DATA: 00 E0 00 00 00 80 CE 87 00 20 A5 A5 A5 A5 A5 A5
### DATA: A5 A5 BC 87 00 20 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
```

```
static int MakeFaultExceptionByIllegalRead(void)
{
   int r;
   volatile unsigned int* p;

   p = (unsigned int*)0x00100000;  // 0x00100000-0

   r = *p;

   return r;
}
```

https://percepio.com
Technical support: support@percepio.com

DevAlert relies on a target-side client library, DFM, that is added in your embedded application. This is responsible for encoding reports from your application into “alert” packets for DevAlert. Note that the DFM library is passive until its API is called from your code, e.g., in an error handler. What happens next depend on your configuration. The alert is either stored on the device and/or uploaded to cloud storage, depending on what “cloudport” and “storageport” modules that are used. The cloudport module defines how to output the data and the storageport defines how to read and write the data to a local storage, such as internal flash in the device. When using Zephyr you may also use the Retained Memory module to store alerts in a special RAM section that survives a (warm) restart. This is useful in cases where your cloudport or storageport modules are not operational. For example, if the storageport module uses external flash memory and SPI interrupts are required for flash memory writes, but the SPI interrupts are masked out during hardware fault exceptions. By storing the alerts to Retained Memory, the device can be restarted without losing the alert data. After the restart you may then call the DFM API to send any pending alerts from Retained Memory to your cloudport or storageport. Note that data in Retained Memory does not survive a power cycle.

In this article, we will set up a solution where the alert data is transmitted from the DFM client in your embedded system to your desktop computer using the serial port, e.g. via a UART or a local network connection. The DFM library includes a “cloudport” for serial port transfer where the data is printed as Hex strings, received by a serial terminal application on a host computer and then piped to a provided “upload tool” (devalerthttps) that uploads to your DevAlert evaluation account storage.

In a production device, you may instead upload the data directly from the device to your own cloud storage, for example using MQTT over Wi-Fi, or simply store the DevAlert data on the device for later retrieval via a temporary connection to host computer.

DevAlert is designed with data privacy in mind and allows for using customer-hosted storage solutions, for example an Amazon S3 bucket in your own AWS account. This way, the diagnostic payload data never leaves your private domain. However, in this example we are using the Percepio-hosted storage provided with the evaluation account to simplify the setup. This way, you don’t need to configure a cloud-side integration.

Processor support

The target-side client of DevAlert, the DFM library, can be used with any embedded processor and is optimized to fit in 32-bit microcontrollers.

The core parts of DFM are processor-agnostic and can be extended with more processor-specific modules for adding diagnostic data in the alerts, such as core dumps and event traces.

At the time of writing, Percepio provides two such diagnostic modules for Zephyr:

- Core dump support using the Zephyr Core Dump module.
- Tracealyzer support using Percepio TraceRecorder.

RTOS support

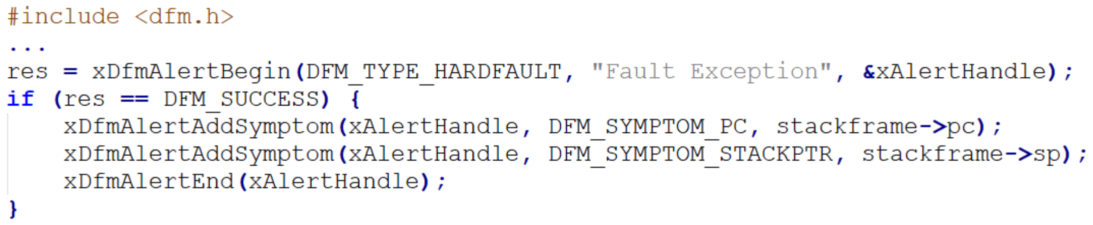
DevAlert’s client side part DFM includes support for Zephyr RTOS. DFM is available in the Zephyr repository as a submodule since Zephyr 3.5.0, so you configure it easily using the kconfig system.

The TraceRecorder library also has built-in support for Zephyr and can be easily added using the kconfig system. Both DFM and the TraceRecorder can be enabled under Modules -> Percepio.

If you want to have the Retained Memory support in DFM then make sure to enable that setting via Modules -> Percepio -> DevAlert -> Retained Memory support.

Using the DevAlert Target-side Client (DFM)

The target-side client of DevAlert, DFM, is provided as a C library under the Apache 2.0 license. DFM is intended to be called on errors and anomalies in the device and encodes the provided data into an "alert" packet for DevAlert. A code example is shown below.



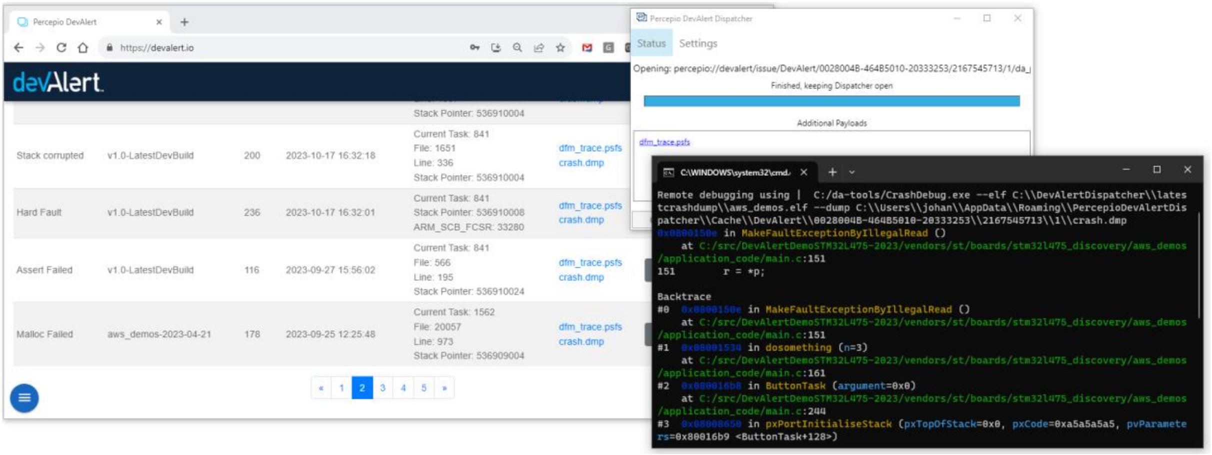
```
#include <dfm.h>
...
res = xDfmAlertBegin(DFM_TYPE_HARDFAULT, "Fault Exception", &xAlertHandle);
if (res == DFM_SUCCESS) {
    xDfmAlertAddSymptom(xAlertHandle, DFM_SYMPTOM_PC, stackframe->pc);
    xDfmAlertAddSymptom(xAlertHandle, DFM_SYMPTOM_STACKPTR, stackframe->sp);
    xDfmAlertEnd(xAlertHandle);
}
```

The first argument to xDfmAlertBegin() is the alert type, followed by an arbitrary message string and finally a DfmAlertHandle_t pointer where the resulting alert handle will be stored. This is then used in additional calls where more information is added. Finally you call xDfmAlertEnd where the data is stored and/or transmitted, depending on the DFM configuration.

The alert types are defined in dfmCodes.h. This header file is generated using the DevAlert console and should not be modified manually, since the definitions must reflect the cloud service database. You can define your own alert types in the DevAlert console under Configuration -> Alert Types and then update dfmCodes.h in Configuration -> Code Export.

The xDfmAlertAddSymptom calls creates a "fingerprint" that characterizes the reported issue, for example using important processor registers and other information. Each piece of fingerprint data is called a "symptom". The fingerprint is used to group the individual alerts into "issues" displayed in the DevAlert dashboard. This way, you don't need to inspect each individual alert but can consider all alerts within the same "issue" as repetitions of the same problem, assuming you have a sufficiently detailed fingerprint. We recommend including at least the program counter and the stack pointer.

The real power of DevAlert comes when including "payloads" such as core dumps, system traces and other device data. To view such diagnostic payloads after receiving the alert, you simply click the links in the DevAlert dashboard. This starts the Dispatcher tool on your local computer, that downloads the payload data and launches it in a suitable desktop tool. In the below example, we selected "crash.dmp" from a Hard Fault alert, and Dispatcher tool launched a GDB script to display the selected core dump.



				Stack Pointer: 536910004	
Stack corrupted	v1.0-LatestDevBuild	200	2023-10-17 16:32:18	Current Task: 841 File: 1651 Line: 336 Stack Pointer: 536910004	dfm_trace.psfs crash.dmp
Hard Fault	v1.0-LatestDevBuild	236	2023-10-17 16:32:01	Current Task: 841 Stack Pointer: 536910008 ARM_SCB_FCSR: 33280	dfm_trace.psfs crash.dmp
Assert Failed	v1.0-LatestDevBuild	116	2023-09-27 15:56:02	Current Task: 841 File: 566 Line: 195 Stack Pointer: 536910024	dfm_trace.psfs crash.dmp
Malloc Failed	aws_demos-2023-04-21	178	2023-09-25 12:25:48	Current Task: 1562 File: 20057 Line: 973 Stack Pointer: 536909004	dfm_trace.psfs crash.dmp

« 1 2 3 4 5 »

Opening: percepio://devalert/issue/DevAlert/0028004B-464B5010-20333253/2167545713/1/da_

Finished, keeping Dispatcher open

Additional Payloads

dfm_trace.psfs

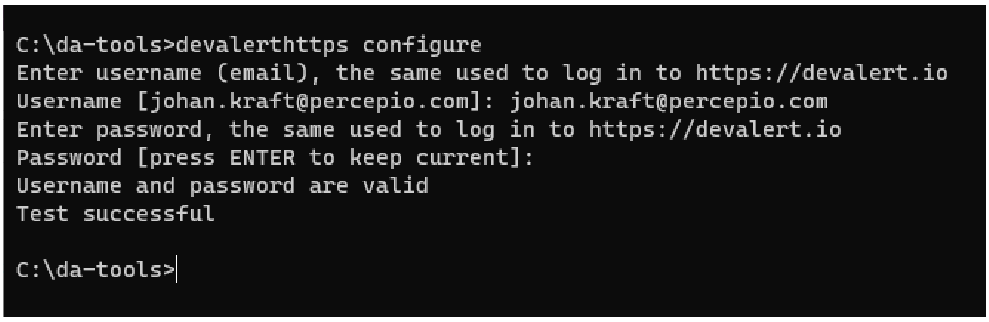
```
Remote debugging using |  C:/da-tools/CrashDebug.exe --elf C:\\DevAlertDispatcher\\lates
tcrashdump\\aws_demos.elf --dump C:\\Users\\johan\\AppData\\Roaming\\PercepioDevAlertDis
patcher\\Cache\\DevAlert\\0028004B-464B5010-20333253\\2167545713\\1\\crash.dmp
0x0800150e in MakeFaultExceptionByIllegalRead ()
    at C:/src/DevAlertDemoSTM32L475-2023/vendors/st/boards/stm32l475_discovery/aws_demos
/application_code/main.c:151
151         r = *p;

Backtrace
#0  0x0800150e in MakeFaultExceptionByIllegalRead ()
    at C:/src/DevAlertDemoSTM32L475-2023/vendors/st/boards/stm32l475_discovery/aws_demos
/application_code/main.c:151
#1  0x08001534 in dosomething (n=3)
    at C:/src/DevAlertDemoSTM32L475-2023/vendors/st/boards/stm32l475_discovery/aws_demos
/application_code/main.c:161
#2  0x080016b8 in ButtonTask (argument=0x0)
    at C:/src/DevAlertDemoSTM32L475-2023/vendors/st/boards/stm32l475_discovery/aws_demos
/application_code/main.c:244
#3  0x08008650 in pxPortInitialiseStack (pxTopOfStack=0x0, pxCode=0xa5a5a5a5, pvParamete
rs=0x80016b9 <ButtonTask+128>)
```

DevAlert Device Integration

To get started with DevAlert and DFM, follow the following steps. Some require a bit more instructions and have their own subsection after this guide. The first steps (1-11) will let you upload a test data file and see this in you DevAlert account. Then we will set up the device integration using the DFM library with data transfer using a serial port. Finally, we extend the solution with Core Dump and Tracealyzer traces for improved remote debugging support.

1. Sign up for an evaluation account at https://devalert.io/auth/signup.
2. Open the welcome email to get your temporary password.
3. Sign in at https://devalert.io/auth/login and update your password.
4. At the welcome screen, select “Activate Service”.
5. Install Python (if you don’t already have it) from https://www.python.org and make sure it is accessible from your terminal. You may need to add it to your PATH environment variable.
6. Download the DevAlert tools from https://percepio.com/downloads/devalert-tools.zip and extract the contents to a new directory, e.g. “devalert-tools”.
7. Open a terminal and enter your “devalert-tools” directory.

   Run “devalerthttps configure” to configure the upload. The username is the email used when registering your DevAlert evaluation account and the password is DevAlert account password.

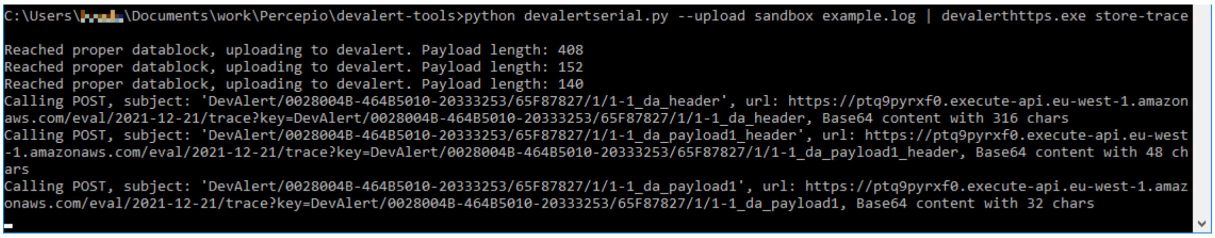
   

   ```
   C:\da-tools>devalerthttps configure
   Enter username (email), the same used to log in to https://devalert.io
   Username [johan.kraft@percepio.com]: johan.kraft@percepio.com
   Enter password, the same used to log in to https://devalert.io
   Password [press ENTER to keep current]:
   Username and password are valid
   Test successful

   C:\da-tools>
   ```

   Note that the username/password authentication in devalerthttps is mainly intended for evaluation accounts to simplify the setup. Production accounts uses mutual TLS.

8. Check that devalerthttps reports “Test successful”. This usually takes about 10 seconds.
9. To test the upload tool, run the following command in your “devalert-tool” directory. This will upload an example alert from the provided file example.log.

   ```
   python devalertserial.py --upload sandbox example.log | devalerthttps.exe store-trace
   ```

   ```
   C:\Users\[redacted]\Documents\work\Percepio\devalert-tools>python devalertserial.py --upload sandbox example.log | devalerthttps.exe store-trace

   Reached proper datablock, uploading to devalert. Payload length: 408
   Reached proper datablock, uploading to devalert. Payload length: 152
   Reached proper datablock, uploading to devalert. Payload length: 140
   Calling POST, subject: 'DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_header', url: https://ptq9pyrxf0.execute-api.eu-west-1.amazon
   aws.com/eval/2021-12-21/trace?key=DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_header, Base64 content with 316 chars
   Calling POST, subject: 'DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_payload1_header', url: https://ptq9pyrxf0.execute-api.eu-west
   -1.amazonaws.com/eval/2021-12-21/trace?key=DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_payload1_header, Base64 content with 48 ch
   ars
   Calling POST, subject: 'DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_payload1', url: https://ptq9pyrxf0.execute-api.eu-west-1.amaz
   onaws.com/eval/2021-12-21/trace?key=DevAlert/0028004B-464B5010-20333253/65F87827/1/1-1_da_payload1, Base64 content with 32 chars
   _
   ```

   If using Linux, use the devalerthttps tool instead of devalerthttps.exe.

   This should result in three uploaded data chunks. It may take a few seconds for each upload. Then close the upload tool by pressing Ctrl-C.

10. Check the dashboard at https://devalert.io. The test alert should appear after a few seconds and should include a payload called “demo_payload.txt”.
11. Setup the Dispatcher tool to view the provided test data, as described in the section **DevAlert Dispatcher - Basic Setup** below.

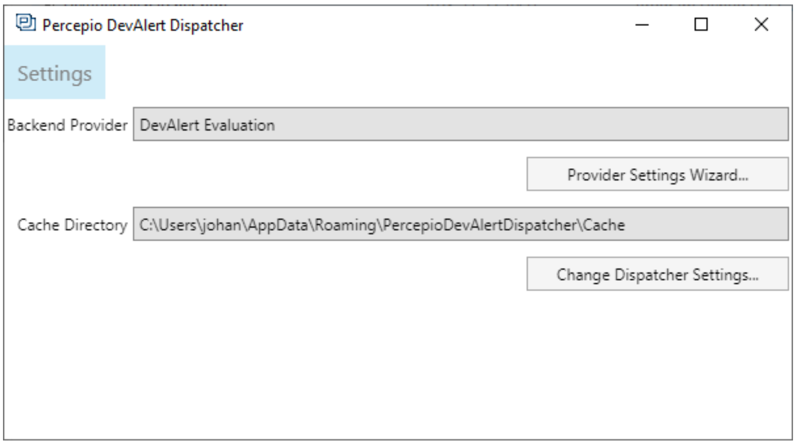
Step 11. DevAlert Dispatcher – Basic Setup

To access a payload from the DevAlert dashboard, we need to download and configure DevAlert Dispatcher. This is a special downloader tool that runs on your local computer when you click on the payload links in the DevAlert dashboard. This is the key to the unique data control and privacy in DevAlert, as the payloads can be stored separately, outside the DevAlert service, and don’t need to pass through Percepio servers. For evaluation accounts the storage is hosted by Percepio for simplicity, but it is still separated from the core DevAlert service.

Dispatcher also loads the downloaded data into the right desktop tool, according to you configuration, meaning all payloads are accessible with a single click.

11.1. Sign in to your DevAlert account and download the latest version of the Dispatcher tool from https://devalert.io/dispatcher.

11.2. Extract all files to your “devalert-tools” directory, locate the the DevAlertDispatcher executable and start it manually. The Dispatcher tool normally starts in a few seconds although it can take a bit longer the first time. You should see the main screen like depicted below.


Percepio DevAlert Dispatcher
Settings

Backend Provider: DevAlert Evaluation

Provider Settings Wizard...

Cache Directory: C:\Users\johan\AppData\Roaming\PercepioDevAlertDispatcher\Cache

Change Dispatcher Settings...

11.3. Select “Provider Setting Wizard…” and press “Next”.

11.4. Enter your username (email) and password. Select “Test” to verify the credentials. Next.

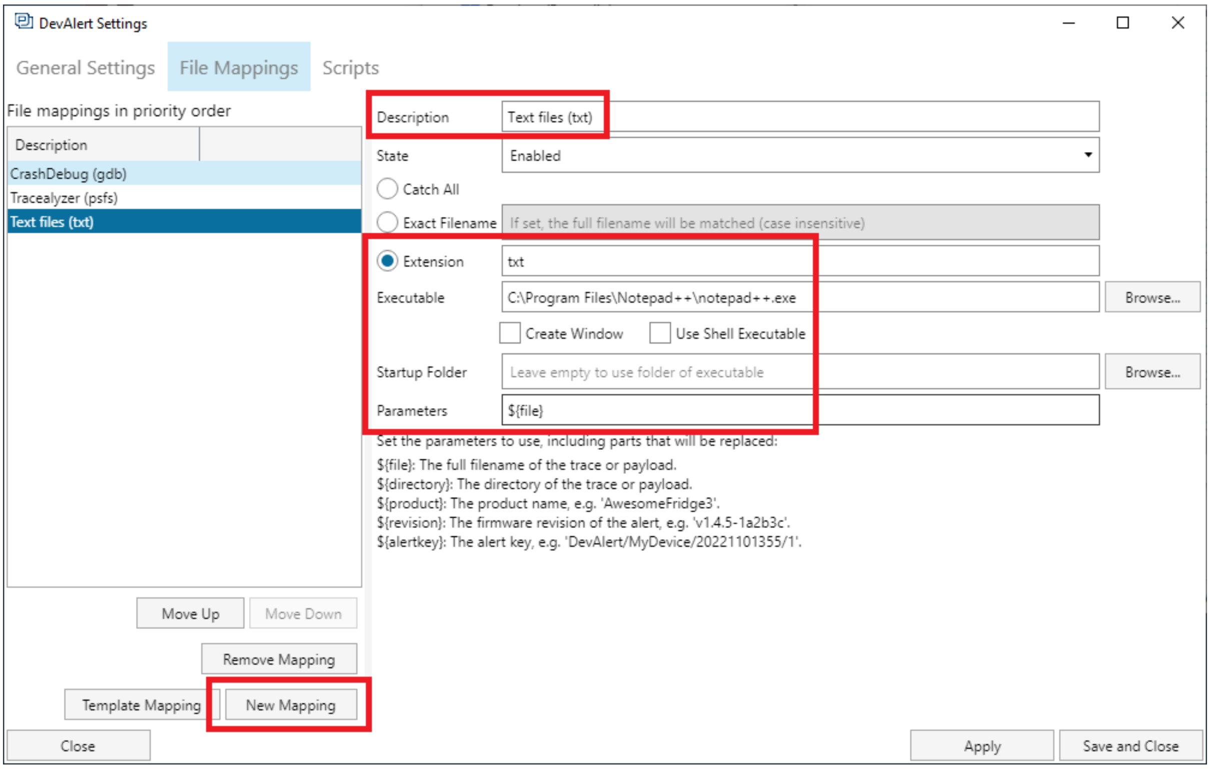
11.5. As “Backend provider” select “DevAlert Evaluation”. Next.

11.6. Dispatcher will ask you’re your username (email) and password a second time. This is for the backend configuration. Select “Test” to verify the credentials. Next, and finish the wizard.

11.7. On the main Dispatcher screen, select “Change Dispatcher Settings…”.

11.8. Verify that “Download Link Status” reads “Handling Percepio links from DevAlert.” This is the status of the web browser integration, where Dispatcher is registered as a protocol handler. If the status is different, click “Enable Download Links” to register the tool manually.

11.9. Select File Mappings and select “New Mapping”. We need to create a mapping for text files (.txt) to view the example payload.



File mappings in priority order

Description
CrashDebug (gdb)
Tracealyzer (psfs)
Text files (txt)

Description: Text files (txt)

State: Enabled

○ Catch All

○ Exact Filename: If set, the full filename will be matched (case insensitive)

● Extension: txt

Executable: C:\Program Files\Notepad++\notepad++.exe

□ Create Window □ Use Shell Executable

Startup Folder: Leave empty to use folder of executable

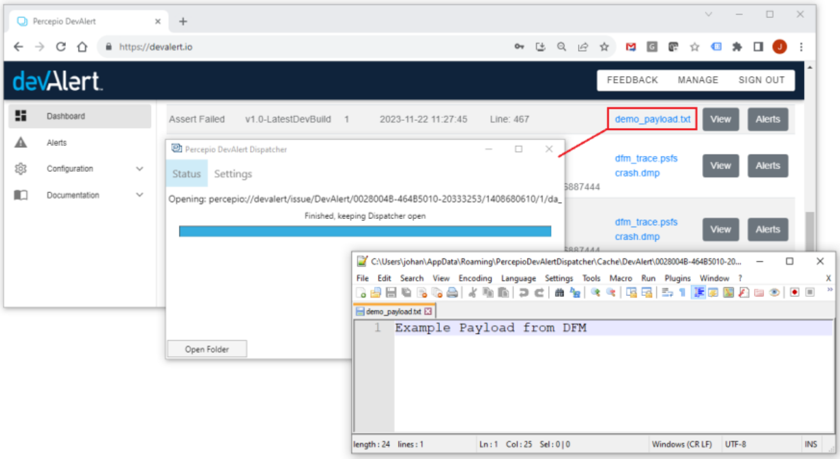
Parameters: ${file}

Set the parameters to use, including parts that will be replaced:
${file}: The full filename of the trace or payload.
${directory}: The directory of the trace or payload.
${product}: The product name, e.g. 'AwesomeFridge3'.
${revision}: The firmware revision of the alert, e.g. 'v1.4.5-1a2b3c'.
${alertkey}: The alert key, e.g. 'DevAlert/MyDevice/20221101355/1'.

11.10. Select “Extension” and enter “txt”. Add the path to your preferred text editor and enter ${file} under parameters. This is the path to payload file once downloaded.

11.11. Save and Close. Close Dispatcher.

11.12. Open your DevAlert dashboard at https://devalert.io. Click on the “demo_payload.txt” link in the example alert. This downloads the selected payload and launches the data in the right tool, according to your Dispatcher configuration.



Assert Failed v1.0-LatestDevBuild 1 2023-11-22 11:27:45 Line: 467

Opening: percepio://devalert/issue/DevAlert/0028004B-464B5010-20333253/1408680610/1/da_

Finished, keeping Dispatcher open

C:\Users\johan\AppData\Roaming\PercepioDevAlertDispatcher\Cache\DevAlert\0028004B-464B5010-20...

```
Example Payload from DFM
```

length : 24 lines : 1 Ln : 1 Col : 25 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

Congratulations, you should now have the devalerthttps upload and Dispatcher tool working!

The next step is to set up your own alerts on your own device. This requires adding the DevAlert client (DFM) in your embedded system. Follow the steps below.

12. Find or create a suitable demo project for your target processor in your development environment. The demo application can be “anything”, but it is recommended to use a simple example project. This is because the demo project we are setting up in this guide will be configured to use Percepio-hosted evaluation storage and parts of the device memory contents will be uploaded there. For real projects with proprietary code this data should be uploaded to your own storage, and only meta-data will be sent to Percepio’s cloud service.
13. Enable serial port communication in Zephyr’s kconfig. Make sure you can print text to a serial port or similar and receive it on the host computer in a terminal program. The terminal program needs to support logging the device output to a file. For Windows users TeraTerm is recommended and used as example in this guide. A suitable configuration of TeraTerm is described in **TeraTerm Setup** below.
14. Integrate the DFM client as described in **DFM Library Integration** below.
15. To get Core Dumps, follow the steps described in **Core Dumps** below.
16. To capture Tracealyzer traces in your alerts, integrate the TraceRecorder library in your project as described in **Adding Tracealyzer Support** below.

You should now have a basic DevAlert setup working using the Serial cloudport. From here you can be extend and modify the solution to suit your needs.

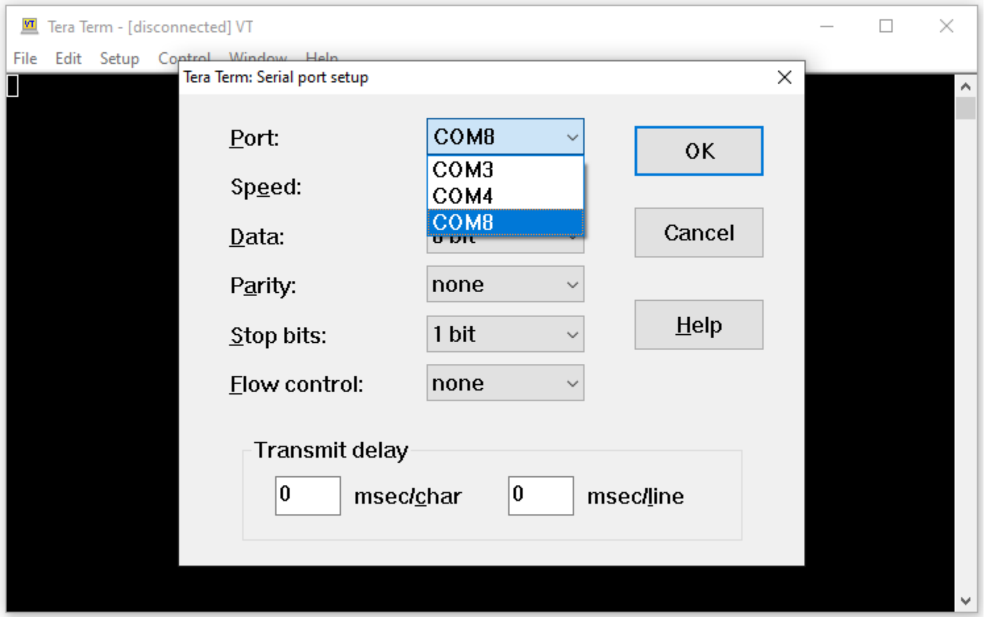
If you have questions or feedback, please contact support@percepio.com.

Step 13. TeraTerm Setup

We have found TeraTerm a very convenient serial terminal program for receiving and logging serial port data on Windows, but there are many alternatives that often can be configured in similar ways. If you prefer a different serial terminal application, configure that in a similar way such that the output is logged to a file in the “devalert-tools” directory.

If you are running Linux (or WSL) you can use: (stty raw; cat > {FILENAME}) < /dev/{SERIAL_DEVICE}

To configure Teraterm to receive DFM data over a serial connection (COM port), select Setup -> Serial Port. Select the right COM port (usually the last if your device was recently plugged in) and Speed. The other settings are usually fine as is.


Tera Term - [disconnected] VT
File Edit Setup Control Window Help
Tera Term: Serial port setup
COM3
COM4
COM8
OK
Cancel
Help

Port: COM8

Speed:

Data: 8 bit

Parity: none

Stop bits: 1 bit

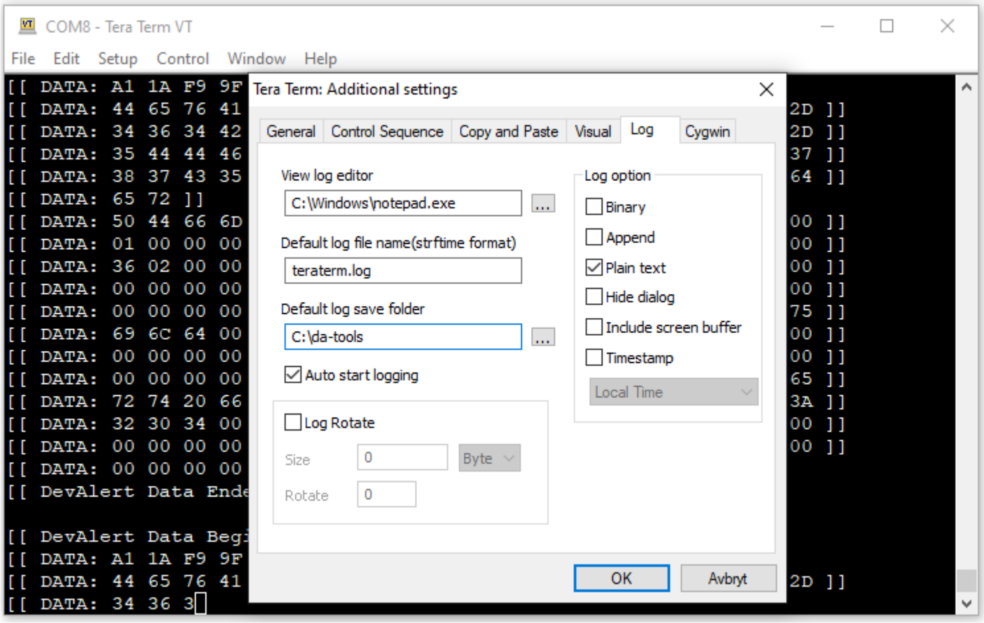
Flow control: none

Transmit delay

0 msec/char 0 msec/line

Next, run your embedded application and make sure the output is presented correctly.

Select Setup -> Additional settings -> Log and configure the settings like below:



```
[[ DATA: A1 1A F9 9F
[[ DATA: 44 65 76 41          2D ]]
[[ DATA: 34 36 34 42          2D ]]
[[ DATA: 35 44 44 46          37 ]]
[[ DATA: 38 37 43 35          64 ]]
[[ DATA: 65 72 ]]
[[ DATA: 50 44 66 6D          00 ]]
[[ DATA: 01 00 00 00          00 ]]
[[ DATA: 36 02 00 00          00 ]]
[[ DATA: 00 00 00 00          00 ]]
[[ DATA: 00 00 00 00          75 ]]
[[ DATA: 69 6C 64 00          00 ]]
[[ DATA: 00 00 00 00          00 ]]
[[ DATA: 00 00 00 00          65 ]]
[[ DATA: 72 74 20 66          3A ]]
[[ DATA: 32 30 34 00          00 ]]
[[ DATA: 00 00 00 00          00 ]]
[[ DATA: 00 00 00 00
[[ DevAlert Data Ende

[[ DevAlert Data Begi
[[ DATA: A1 1A F9 9F
[[ DATA: 44 65 76 41          2D ]]
[[ DATA: 34 36 3
```

View log editor: C:\Windows\notepad.exe

Default log file name(strftime format): teraterm.log

Default log save folder: C:\da-tools

☑ Auto start logging

☐ Log Rotate

Size 0 Byte

Rotate 0

Log option

☐ Binary

☐ Append

☑ Plain text

☐ Hide dialog

☐ Include screen buffer

☐ Timestamp

Local Time

Default log save folder: your Devalert tools folder

Auto start logging: Checked

Log Rotate: Unchecked

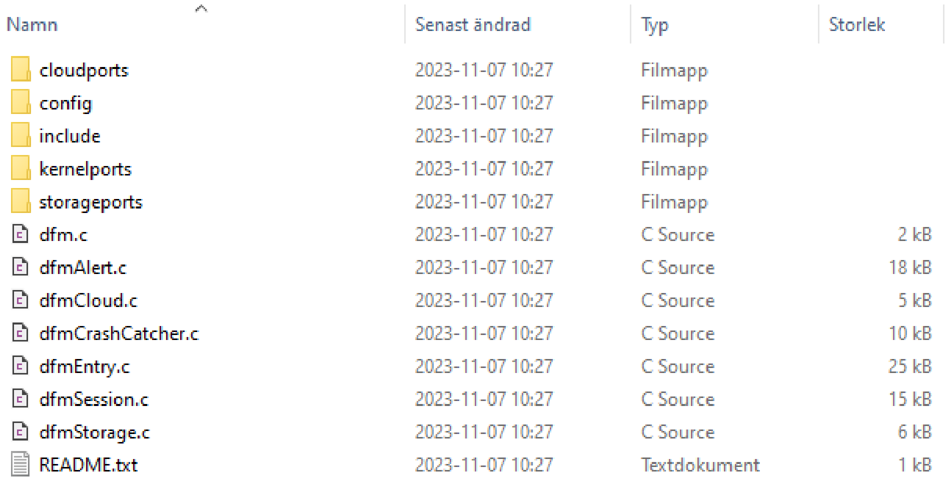
Append: Unchecked

Finally, select Setup -> Save setup and overwrite the default settings file (TERATERM.INI).

Close Teraterm and continue with Step 14 below.

Step 14. DFM Library Integration

14.1. To enable DFM in your project, run menuconfig and enable it in Modules -> Percepio. The following files will be included in your build.



Namn	Senast ändrad	Typ	Storlek
cloudports	2023-11-07 10:27	Filmapp	
config	2023-11-07 10:27	Filmapp	
include	2023-11-07 10:27	Filmapp	
kernelports	2023-11-07 10:27	Filmapp	
storageports	2023-11-07 10:27	Filmapp	
dfm.c	2023-11-07 10:27	C Source	2 kB
dfmAlert.c	2023-11-07 10:27	C Source	18 kB
dfmCloud.c	2023-11-07 10:27	C Source	5 kB
dfmCrashCatcher.c	2023-11-07 10:27	C Source	10 kB
dfmEntry.c	2023-11-07 10:27	C Source	25 kB
dfmSession.c	2023-11-07 10:27	C Source	15 kB
dfmStorage.c	2023-11-07 10:27	C Source	6 kB
README.txt	2023-11-07 10:27	Textdokument	1 kB

14.2. Enable Cyclic Redundance Check (CRC) support. This is used by the “Serial” Cloudport which we use for this demo. It may also be required for Retained Memory to verify data integrity if that is enabled

14.3. Select the “Serial” cloudport module from modules -> Percepio -> DevAlert -> CloudPort Config -> Cloudport. This specifies how DFM will output the data. It is easy to create your own cloud port supporting any connection type you have available. Contact support@percepio.com for additional assistance if necessary.

14.4. DFM also needs a storageport module, that specifies how to store data on the device. In this demo we don’t need a storageport so selecting None at modules -> Percepio -> DevAlert -> StoragePort Config -> Storageport. This will use a dummy storageport that doesn’t store any data. For other DFM setups you may implement your own storageport or use one of the other provided storageport modules, e.g., Flash or File system.

14.5. If you require Retained Memory for temporary storage of Alerts in situations when the cloudport and storageport is not available (such as hardfault context), follow these steps:

14.5.1. Next we need to set up a Retained Memory section called retention0 in the device tree. It is recommended to create a file named app.overlay for this. Follow the Zephyr guides on how to do this for Retained Memory, e.g.

https://docs.zephyrproject.org/latest/services/retention/index.html.

Note: We have experienced issues with using Retained Memory on Zephyr 3.5 together with a Nordic device when the SRAM was fully mapped (such as 64 KB being split into 60 KB + 4 KB) and where the retained memory region includes the uppermost part of the SRAM range. The Zephyr 3.5 checksum calculation then reported the Retained Memory data as invalid. This could be solved by not placing the Retained Memory at the very end of the SRAM memory range, e.g. by leaving a small amount of memory unmapped.

14.5.2. Make sure "Generic Zephyr RAM retained memory driver" is enabled and disable the Mutex support for Retained Memory. This is found in two places:

- Device drivers -> Retained memory drivers
- Subsystems and OS Services -> Retention support

14.5.3. Enable Modules -> Percepio -> DevAlert -> Retained Memory support

14.6. Add a call to xDfmInitialize() in the startup. Place the call in the main() function. In the current version of DFM this function needs two callback functions, described below.

14.6.1. xGetUniqueSessionID - Should provide a "session identifier". This should ideally be a unique string that is never repeated for a particular device, for example a reboot counter that is stored in flash memory or a wall clock timestamp if available. For an evaluation setup, you may consider using a random number. But this is NOT recommended for production devices as this may cause alerts to be incorrectly classified as duplicates and rejected.

14.6.2. xGetDeviceName - The second callback should provide the device name. For evaluation purposes you may use a constant string here, like "Device123". For production devices this should use a unique serial number for the particular device. Note that spaces are not allowed in the device names.

Both callbacks should write the ID to the provided buffer (cBuffer) and the length of the string to *pulBytesWritten. See the code example below.

```
DfmResult_t myGetSessionID(char cBuffer[], uint32_t ulSize, uint32_t* pulBytesWritten);
DfmResult_t myGetDeviceName(char cBuffer[], uint32_t ulSize, uint32_t* pulBytesWritten);

...

if (xDfmInitialize(myGetSessionID, myGetDeviceName) == DFM_FAIL)
{
    printf("Failed to initialize DFM\r\n");
}

...

DfmResult_t myGetSessionID(char cBuffer[], uint32_t ulSize, uint32_t* pulBytesWritten)
{
    uint32_t nBytes = snprintf(cBuffer, ulSize, "%08X", <SessionID> );
    if (nBytes > 0) {
        *pulBytesWritten = nBytes;
        return DFM_SUCCESS;
    }
    return DFM_FAIL;
}

DfmResult_t myGetDeviceName(char cBuffer[], uint32_t ulSize, uint32_t* pulBytesWritten)
{
    uint32_t nBytes = snprintf(cBuffer, ulSize, "MyDevice");
    if (nBytes > 0){
        *pulBytesWritten = nBytes;
        return DFM_SUCCESS;
    }
    return DFM_FAIL;
}
```

14.7. After the call to xDfmInitialize(), add a test alert like in the following code example.

```
#include <dfm.h>

DfmAlertHandle_t xAlertHandle = 0;
char* payloadString = "My own payload from DFM";

...

if (xDfmAlertBegin(DFM_TYPE_ASSERT_FAILED, "Demo alert", &xAlertHandle) == DFM_SUCCESS) {
    xDfmAlertAddSymptom(xAlertHandle, DFM_SYMPTOM_LINE, __LINE__);
    xDfmAlertAddPayload(xAlertHandle,
                          payloadString, strlen(payloadString), "my_payload.txt");
    xDfmAlertEnd(xAlertHandle);
}
```

14.8. Build and run your project and make sure you see DFM data in the serial console.

14.9. Configure your serial terminal program to save the data to a log file (e.g. “mydevice.log”) and upload the data like in the previous example:

```
python devalertserial.py –upload sandbox <FILENAME> | devalerthttps.exe store-trace
```

If you are running Linux or WSL you can use:

```
(./devalertserial.py --upload sandbox <FILENAME>) | (./devalerthttps store-trace)
```

You should now see your own alert in the DevAlert dashboard.

Congratulations, you now have the full chain working from device to dashboard!

In the next steps we will add two kinds of alert payloads, core dumps and traces for Tracealyzer. These steps are optional but recommended to see the full potential of DevAlert.

Step 15. Core Dumps in Zephyr (optional)

Core dumps include registers, stack and other memory contents. This way you can inspect the system state in detail at a fault location. Core dumps can be provided both on fault exceptions (e.g. on invalid memory access) and when creating custom alerts in your code.

For most devices this will require Retained Memory to be configured. Refer to this guide regarding Retained Memory to configure it properly.

To collect Core Dumps from Zephyr on DFM alerts, follow these steps:

(But first, remember to remove the test alert from step 14.6.)

15.1. Enable the following settings in kconfig:

- Subsystems and OS Services -> Debugging Options -> Core Dump
- Subsystems and OS Services -> Debugging Options -> Core Dump -> Coredump backend -> Backend subsystem for coredump defined out of tree
- Subsystems and OS Services -> Debugging Options -> Core Dump -> Memory Dump -> Minimal
- Modules -> percepio -> DevAlert (DFM) -> Enable Core Dump support
  - Depending on your device you can opt to temporarily save the Core Dump in Retained Memory or store it long term using the storageport. Refer to this guide regarding Retained Memory to configure it properly.

15.2. To send any stored alert data after a restart, add a call to xDfmAlertSendAll() right after xDfmInitialize(…). This will use the cloudport (in our case a Serial port) to transfer any alerts saved in Retained Memory or stored using a proper storageport.
Note: If you do not wish to attempt to send the Alerts on each startup you can instead call xDfmAlertStoreRetainedMemory() to transfer any Alerts in Retained Memory to the permanent storage in the storageport. When it is time to send the alerts you can call xDfmAlertSendAll() to do so.

15.3. For testing purposes we will now generate a hard fault on the device, triggering DFM to generate an alert and CoreDump in Retained Memory, then reboot the device:

```
(void)xDfmInitialize(...);      // Initializes DevAlert (DFM).
(void)xDfmAlertSendAll();      // Send all alerts in storage
k_msleep(5000);                // For testing...
*(volatile int*)(0xEFFF0000);  // Set a breakpoint here to avoid a restart loop
```

When you step through this code, the device will hardfault at the last line. After reboot it will output the alert over the serial connection. Assuming you have the upload tools running, the alert will then be uploaded it to the DevAlert cloud service.

Note: If your debugger halts the execution directly in the hard fault handler, you need to resume the execution for the alert data output to occur. This usually happens because of a setting in your debugger tool. In Eclipse-based IDEs, this is usually found in the Debug Configuration -> Startup -> "Halt on Execution".

15.4. You should now be able to see the alert in the DevAlert Dashboard. Click on Details for the alert and download the CoreDump.dmp file.

DevAlert Dispatcher should launch and start downloading the file. It will complain that no valid mapping has been found for .dmp files. This is expected as we haven't configured a script or program that will handle this type of file.

Create a custom script for viewing the core dump. As a first step in your script, copy the dump file to a specific directory that is easy to find in later steps, e.g:

```
#!/bin/bash
mkdir -p ~/devalert-latest
cp $1 ~/devalert-latest/latest.dmp
```

Create a new File Mapping in Dispatcher (see Section 11.9) for .dmp files that calls your script. It will need to know the location of the .dmp file, so we pass the ${file} variable from Dispatcher as parameter to the script. This is $1 in the bash script.

You may extend the script with the following steps to view the core dump, or run them manually on the downloaded core dump file.

- Start the GDB server:

  ./zephyrproject/zephyr/scripts/coredump/coredump_gdbserver.py <elf file> <dmp file>

- Start the GDB client (using your sdk version):
  zephyr-sdk-0.16.4/arm-zephyr-eabi/bin/arm-zephyr-eabi-gdb <elf file>

- Connect to the GDB server:
  `target remote localhost:1234`

- To see the backtrace from the core dump, type "bt" in the GBD client console.

Congratulations, you should now have working core dump support!

If you are using an IDE debugger supporting GDB, you may use your IDE debugger as GDB client instead of using the command-line client. This is configured in the debug configuration of your IDE, depending on what IDE you are using. Create a new debug configuration (or similar) and instruct it to run the following GDB command: `target remote localhost:1234.`

Step 16. Adding Tracealyzer Support (optional)

Percepio Tracealyzer is a visualization tool for event traces that simplifies debugging, especially for more complex issues in multithreaded RTOS applications. This can be used within DevAlert by integrating the TraceRecorder library and providing the trace data as an alert payload. This way, you can collect traces on system errors and anomalies showing the timeline of events just before the alert was created.

Tracealyzer supports a number of common real-time operating systems using different tracing libraries, e.g. LTTng for Linux and Percepio TraceRecorder for MCU-class RTOSes. The DevAlert client library, DFM, has so far only been tested with the Percepio TraceRecorder. This includes support for FreeRTOS, SafeRTOS, Zephyr, ThreadX, PX5 and bare metal systems. The latter can be extended with custom instrumentation for any RTOS or similar C/C++ system using the Tracealyzer SDK.

16.1. To get started with Tracealyzer, sign up for a free evaluation license at https://percepio.com/tracealyzer/download-tracealyzer/

16.2. Follow the integration guides at https://percepio.com/tracealyzer/gettingstarted/, in line with the specific instructions below. Additional information is found in Tracealyzer User Manual (see Help menu), in particular the section "Creating and Loading Traces". This is the place to go if you want to configure a bare metal integration.

16.3. Enable the following kconfig options:

* Subsystems and OS Services->Tracing Support->Tracing Format->Percepio Tracealyzer support

* Modules->Percepio->TraceRecorder->Stream Port->Ring Buffer

* Modules->Percepio->DevAlert (DFM)->Enable Core Dump support->Save Trace

Modify your main.c code to look like this:

```
xTraceEnable(TRC_START);    /* Initializes and starts tracing */

xDfmInitialize(…);          /* Initializes DevAlert (DFM). User provides parameters. */

xDfmAlertSendAll();         /* Send all alerts in storage or Retained Memory*/

k_msleep(5000);             /* For testing */

*(volatile int*)(0xEFFF0000); /* Set breakpoint here to avoid repeated restarts */
```

The "Save Trace" option mentioned above will enable tracing and automatically add traces in CoreDump alerts. To include traces in custom alerts, call xDfmAlertAddTrace() in between xDfmAlertBegin and xDfmAlertEnd, like this:

```
if (xDfmAlertBegin(DFM_TYPE_ASSERT_FAILED, "Demo alert", &xAlertHandle) == DFM_SUCCESS) {
    xDfmAlertAddSymptom(xAlertHandle, DFM_SYMPTOM_LINE, __LINE__);
    xDfmAlertAddPayload(xAlertHandle,
                        payloadString, strlen(payloadString), "my_payload.txt");
    xDfmAlertAddTrace(xAlertHandle);
    xDfmAlertEnd(xAlertHandle);
    xTraceEnable(TRC_START);
}
```

Note that xDfmAlertAddTrace(xAlertHandle) will disable tracing to ensure that new events are not added while the trace buffer is being read by DFM, as this may cause incorrect data. As shown above, you may add xTraceEnable(TRC_START) after xDfmAlertEnd() to resume the tracing.

You may also consider adding a User Event (e.g. xTracePrintF) to mark the alert in the trace. You may add multiple such events or add additional parameters to log any information of relevance. Make sure to add these calls before calling xDfmAlertAddTrace(xAlertHandle).

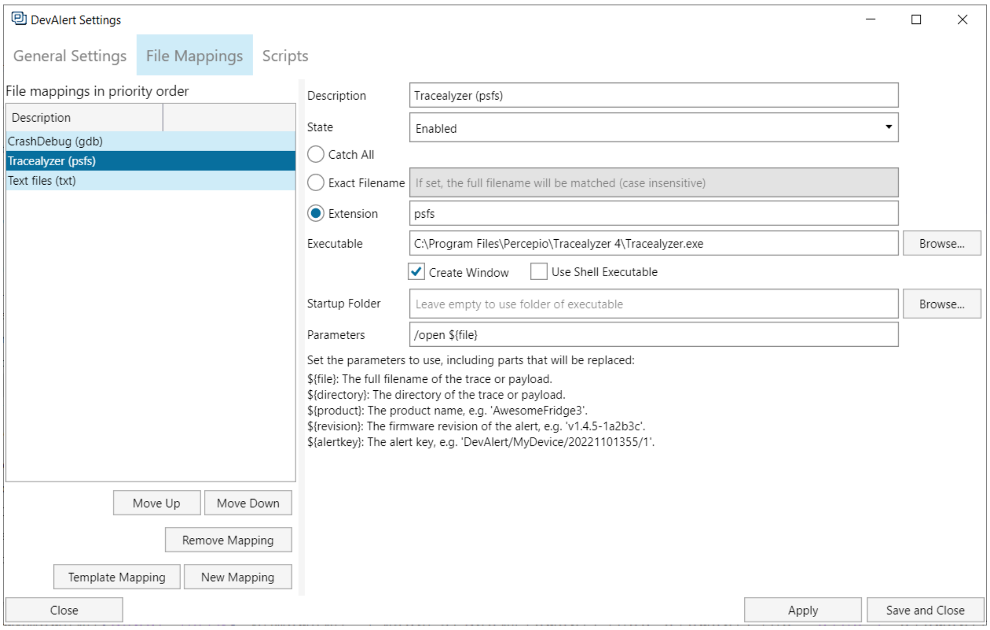
Make sure to configure the trace size to ensure the entire alert including coredump and trace fits within your Retained Memory. If this is too small, no alert will be provided. In that case, try increasing the Retained Memory size in your app.overlay. If you can't increase the Retained Memory size more, you may consider reducing the trace buffer size:

* Modules->Percepio->TraceRecorder->Ring Buffer Config->Buffer size

Another option to reduce the alert size is to exclude the trace data, using this setting:

* Modules->Percepio->DevAlert (DFM)->Enable Core Dump support->Save Trace

16.4. In Dispatcher, add a File Mapping for Tracealyzer as demonstrated below.



File mappings in priority order

Description
CrashDebug (gdb)
Tracealyzer (psfs)
Text files (txt)

Description: Tracealyzer (psfs)

State: Enabled

○ Catch All

○ Exact Filename: If set, the full filename will be matched (case insensitive)

● Extension: psfs

Executable: C:\Program Files\Percepio\Tracealyzer 4\Tracealyzer.exe

☑ Create Window ☐ Use Shell Executable

Startup Folder: Leave empty to use folder of executable

Parameters: /open ${file}

Set the parameters to use, including parts that will be replaced:
${file}: The full filename of the trace or payload.
${directory}: The directory of the trace or payload.
${product}: The product name, e.g. 'AwesomeFridge3'.
${revision}: The firmware revision of the alert, e.g. 'v1.4.5-1a2b3c'.
${alertkey}: The alert key, e.g. 'DevAlert/MyDevice/20221101355/1'.

16.5. When you run the application and upload an alert, the dashboard will show a "dfm_trace.psfs" payload. Click it and Dispatcher should start Tracealyzer and load the trace.

Congratulations, you should now have Tracealyzer support enabled in DevAlert. To learn more about Tracealyzer, see e.g. https://percepio.com/tracealyzer/ and https://percepio.com/category/handson/.

If Tracealyzer doesn't start in 10 seconds or so, check if you have another open Tracealyzer instance and close that first.

You have now completed the DevAlert getting started guide and have a DevAlert setup for using serial port transfer to your evaluation account. To use DevAlert in other ways, for example uploading the alerts directly from the device to cloud, you need a production account and other cloudport and storageport modules. This may require developing custom modules to get your desired setup. This is not covered by this guide but there are example modules included in the DFM repository. Note the modules under kernelports/<RTOS>. For example kernelports/FreeRTOS/AWS_MQTT, which is intended for MQTT upload to AWS IoT Core. This allows for storing the data in your own AWS account.

If you want assistance with setting up DevAlert for your needs, or have general questions, feel free to contact support@percepio.com or sales@percepio.com.

NOTES

To enable the "Flash" storageport, make sure to enable FLASH, FLASH_MAP and FCB.

To enable the "File system" storageport, you need to enable FLASH, FLASH_MAP and FILE_SYSTEM.