

# **Cinterion<sup>®</sup> IoT SDK**

Getting Started

Version: 12

DocId: `iot_sdk_ug_v12`

User Guide:	<b>Cinterion® IoT SDK</b>
Version:	<b>12</b>
Date:	<b>2021-08-18</b>
DocId:	<b>iot_sdk_ug_v12</b>
Status:	<b>Confidential / Released</b>

**GENERAL NOTE**

THE USE OF THE PRODUCT INCLUDING THE SOFTWARE AND DOCUMENTATION (THE "PRODUCT") IS SUBJECT TO THE RELEASE NOTE PROVIDED TOGETHER WITH PRODUCT. IN ANY EVENT THE PROVISIONS OF THE RELEASE NOTE SHALL PREVAIL. THIS DOCUMENT CONTAINS INFORMATION ON THALES DIS AIS DEUTSCHLAND GMBH ("THALES") PRODUCTS. THE SPECIFICATIONS IN THIS DOCUMENT ARE SUBJECT TO CHANGE AT THALES'S DISCRETION. THALES GRANTS A NON-EXCLUSIVE RIGHT TO USE THE PRODUCT. THE RECIPIENT SHALL NOT TRANSFER, COPY, MODIFY, TRANSLATE, REVERSE ENGINEER, CREATE DERIVATIVE WORKS; DISASSEMBLE OR DECOMPILE THE PRODUCT OR OTHERWISE USE THE PRODUCT EXCEPT AS SPECIFICALLY AUTHORIZED. THE PRODUCT AND THIS DOCUMENT ARE PROVIDED ON AN "AS IS" BASIS ONLY AND MAY CONTAIN DEFICIENCIES OR INADEQUACIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THALES DISCLAIMS ALL WARRANTIES AND LIABILITIES. THE RECIPIENT UNDERTAKES FOR AN UNLIMITED PERIOD OF TIME TO OBSERVE SECRECY REGARDING ANY INFORMATION AND DATA PROVIDED TO HIM IN THE CONTEXT OF THE DELIVERY OF THE PRODUCT. THIS GENERAL NOTE SHALL BE GOVERNED AND CONSTRUED ACCORDING TO GERMAN LAW.

**Copyright**

Transmittal, reproduction, dissemination and/or editing of this document as well as utilization of its contents and communication thereof to others without express authorization are prohibited. Offenders will be held liable for payment of damages. All rights created by patent grant or registration of a utility model or design patent are reserved.

Copyright © 2021, THALES DIS AIS Deutschland GmbH

**Trademark Notice**

Thales, the Thales logo, are trademarks and service marks of Thales and are registered in certain countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other registered trademarks or trademarks mentioned in this document are property of their respective owners.

# Contents

<b>0</b>	<b>Document History .....</b>	<b>4</b>
<b>1</b>	<b>Introduction .....</b>	<b>6</b>
1.1	Supported Products .....	6
1.2	Related Documents .....	6
1.3	Prerequisites .....	6
1.4	Folder Structure .....	7
<b>2</b>	<b>Setup .....</b>	<b>8</b>
2.1	Software Installation on PC.....	8
2.1.1	Linux .....	8
2.1.2	Windows .....	8
2.2	Firmware Update on Module.....	9
2.3	Connecting PC and Module .....	11
2.4	USB Driver Installation (Windows only) .....	12
2.4.1	Windows 7 .....	12
2.4.2	Windows 10 .....	15
<b>3</b>	<b>Managing Embedded Applications .....</b>	<b>16</b>
3.1	Building Applications.....	16
3.2	Managing Signed Applications.....	17
3.2.1	Querying Secure Boot Status and Signature Parameters .....	17
3.2.2	Generating the Application Root of Trust.....	18
3.2.3	Signing an Application .....	18
3.2.4	Verifying an Application .....	19
3.2.5	Installing and Protecting the Application Root of Trust in the Module. ....	19
3.3	Downloading and Starting Applications .....	21
3.3.1	Limitations.....	21
3.4	Logging .....	22
3.5	Exception Handling .....	23
3.5.1	List Available Backtraces .....	23
3.5.2	Remove Backtraces.....	23
3.5.3	Process a Backtrace .....	24
3.6	Navigating Module File System .....	24
3.7	Monitoring Application RAM.....	25
<b>4</b>	<b>Developing Embedded Applications .....</b>	<b>26</b>
4.1	Available APIs.....	26
4.2	Exception Handler Development .....	28
4.2.1	Definition .....	29
4.2.2	Sample.....	31
4.2.3	Installation.....	33
4.3	Implementing Multiple Embedded Applications in Parallel .....	34

## 0 Document History

Preceding document: "Cinterion® IoT SDK" Version 11

New document: "Cinterion® IoT SDK" Version 12

Chapter	What is new
<a href="#">1.1</a>	Updated supported products.
<a href="#">3.1</a>	Added build.py as build option, and -p as product parameter.
<a href="#">3.3.1</a>	Removed some limitations.
<a href="#">4.3</a>	New section <a href="#">Implementing Multiple Embedded Applications in Parallel</a> .

Preceding document: "Cinterion® IoT SDK" Version 10

New document: "Cinterion® IoT SDK" Version 11

Chapter	What is new
<a href="#">1.1</a>	Removed hardware build information. Added TX62-W-B as supported product.
<a href="#">1.3</a>	Added further prerequisites.
<a href="#">2.1.1</a> , <a href="#">2.1.2</a>	Added numpy as installation step under Linux and Windows.
<a href="#">3.2</a>	Revised subsections to clarify certain points.
<a href="#">4.1</a>	Revised table listing available APIs.

Preceding document: "Cinterion® IoT SDK" Version 09

New document: "Cinterion® IoT SDK" Version 10

Chapter	What is new
<a href="#">2.2</a>	Revised firmware update notes.
<a href="#">3.2</a>	Updated signed application related information.
<a href="#">3.7</a>	Renamed section to <a href="#">Monitoring Application RAM</a> , and revised it.

Preceding document: "Cinterion® IoT SDK" Version 08

New document: "Cinterion® IoT SDK" Version 09

Chapter	What is new
Throughout document	Added PLS83-W as supported product, and adapted sections accordingly.
<a href="#">3.7</a>	New section Memory for RAM Instructions.

Preceding document: "Cinterion® IoT SDK" Version 07

New document: "Cinterion® IoT SDK" Version 08

Chapter	What is new
<a href="#">1.1</a>	Added TX62-W and TX82-W as supported products.

Preceding document: "Cinterion® IoT SDK" Version 06

New document: "Cinterion® IoT SDK" Version 07

Chapter	What is new
Throughout document	Added information on how to manage an application root of trust.

Chapter	What is new
<a href="#">3.5</a>	New section <a href="#">Exception Handling</a> .
<a href="#">4.2</a>	New section on <a href="#">Exception Handler Development</a> .

Preceding document: "Cinterion® IoT SDK" Version 05

New document: "Cinterion® IoT SDK" Version 06

Chapter	What is new
<a href="#">1.3</a>	Added OpenSSL as prerequisite.
<a href="#">3.2</a>	New section <a href="#">Managing Signed Applications</a> .

Preceding document: "Cinterion® IoT SDK" Version 04

New document: "Cinterion® IoT SDK" Version 05

Chapter	What is new
<a href="#">2.2</a>	Revised remarks about boot loader downloads.
<a href="#">2.3</a> , <a href="#">2.4</a> , <a href="#">3.3</a> , <a href="#">3.4</a>	Revised sections as there is no longer a dedicated USB logging port.
<a href="#">2.4</a>	Revised section to differentiate between Windows 7 and 10 USB driver installations. Removed description of USB power saving issue as it no longer exists.
<a href="#">4</a>	Added further API examples.

Preceding document: "Cinterion® IoT SDK" Version 03

New document: "Cinterion® IoT SDK" Version 04

Chapter	What is new
<a href="#">1.1</a>	Added further supported HW Build version.
<a href="#">2.2</a>	Added information that firmware download is now also possible via USB. Added details about required boot loader version for firmware updates.
<a href="#">2.4</a>	Revised USB device enumeration after USB driver installation under Windows.

Preceding document: "Cinterion® IoT SDK" Version 02

New document: "Cinterion® IoT SDK" Version 03

Chapter	What is new
<a href="#">3.3</a> , <a href="#">3.4</a>	Revised section to enhance logging descriptions.
<a href="#">4</a>	Revised chapter on API documentation.

Preceding document: "Cinterion® IoT SDK" Version 01

New document: "Cinterion® IoT SDK" Version 02

Chapter	What is new
<a href="#">2.3</a> , <a href="#">2.4</a>	Revised and enhanced description of USB port enumeration for Linux and Windows.
<a href="#">4</a>	Revised table listing API documentation (added footnote).

New document: "Cinterion® IoT SDK" Version 01

Chapter	What is new
---	Initial document setup.

# 1 Introduction

This document describes how to get started with the Cinterion® IoT SDK, i.e., the software development kit (SDK) for generating embedded applications that can be run on supported Cinterion® modules.

The purpose of this document is to guide you through the process of connecting the module, building the first application, and downloading it into the module.

## 1.1 Supported Products

This document applies to the following Thales modules:

Cinterion® module	Software revision	Application revision
EXS62-W	01.100	01.000.10
EXS82-W	01.100	01.000.10
EXS62-W	02.000	01.000.01
EXS82-W	02.000	01.000.01
TX62-W	01.000	01.000.03
TX82-W	01.000	01.000.03
PLS83-W/-X/-X2/-X3/-EP/-J/-LA	01.100 (01.106)	01.000.00
PLS63-W/-X/-X2/-X3/-EP/-J/-LA	01.100 (01.106)	01.000.00

## 1.2 Related Documents

- [1] AT Command Set for the supported product
- [2] Hardware Interface Description for the supported product
- [3] Cinterion® LGA DevKit User Guide
- [4] Application Note 62: Transport Layer Security for Client TCP/IP Services

## 1.3 Prerequisites

- Cinterion® module for use with the appropriate
- Cinterion® LGA DevKit (SM/L/T)
- Computer running either Windows 10 or Windows 7, or Linux as of for example Ubuntu 18.4
- Administrator rights on the computer
- OpenSSL (1.1.1 recommended) installed and accessible via PATH environment variable
- Python 2.7.17 (recommended) from this [location](#).
- If using HSM or other secure token for the private keys (e.g. pkcs11), the appropriate OpenSSL engine and needed drivers/libraries need to be installed
- Extracted 7z archive that is attached to the AN62 PDF file. For further details please refer to [4]: Section 3.2.1. Note that the Java desktop runtime environment should be 32bit only, to allow running the Java tools provided in the 7z archive.
- (U)SIM - as an option and only required if using GSM/UMTS/LTE connections

## 1.4 Folder Structure

The Cinterion® IoT SDK provided by Thales runs under Windows and Linux and includes appropriate firmware, USB drivers for Windows, as well as application examples.

You can use all the command line tools provided by the SDK directly.

The SDK has the following folder structure.

```
SDK/  
  common/      # Headers and libraries  
  doc/         # Documentation  
  drivers/     # USB driver package  
  examples/    # Sample applications  
  firmware/    # Module firmware  
  tools/       # LLVM compiler suite, firmware programming tool,  
               application download tool  
  README.txt
```

## 2 Setup

This chapter describes how to set up the Cinterion® IoT SDK. Please complete the steps in the following sections, before you can start managing embedded applications.

### 2.1 Software Installation on PC

In addition to the SDK please install the following additional software packages onto your PC.

#### 2.1.1 Linux

Install the Python 2.7.x package and pyserial 3.x.

```
# Install Python:
$ sudo apt-get install python

# Show version of python:
$ python -V

# Install pyserial:
$ sudo apt-get install python-serial

# Show version of pyserial:
$ sudo apt-cache policy python-serial

# Install numpy:
$ python -m pip install numpy

# Show version of numpy:
$ python -m pip show numpy
```

#### 2.1.2 Windows

Install Python 2.7.x and pyserial 3.x.

```
# Download and install Python 2.7:
https://www.python.org/downloads/release/python-2717/

# Add Python folder in PATH environment variable
$ setx PATH "%PATH%;C:\Python27"

# Show version of Python:
$ python -V

# Install pyserial:
$ python -m pip install pyserial

# Show version of pyserial:
$ python -m pip show pyserial

# Install numpy:
$ python -m pip install numpy

# Show version of numpy:
$ python -m pip show numpy
```



## 2.2 Firmware Update on Module

To update the module with a firmware that supports embedded processing please proceed as follows:

1. Mount the Cinterion® module onto the LGA DevKit. For more information on the LGA DevKit please refer to [3].
2. Connect the ASC0/USB or Native USB interface on the LGA DevKit to the PC using the USB cable.
3. Press the “ON” button on the LGA DevKit to start up the module.
4. Use the firmware.py tool to read the boot loader version installed on your module. If the version is older than the SBL1\_XXX.usf file located under SDK/firmware/<product>, please download the new boot loader onto the module.  
**Note:** Do not interrupt this boot loader update operation, e.g., by cutting the power.
5. Use the firmware.py tool to download the new firmware over ASC0 or USB. The new firmware is provided as \*.usf file under SDK/firmware/<product>/ (e.g., exs82.usf).

```
# Go to the tool directory from SDK (on linux, use '/' instead of '\')
$ cd SDK\tools

# Connect to the target specifying your port (on windows) or your device
#(on linux) and speed. If successful, these communication parameters
# will be saved locally (config.db) for next commands

# On windows
$ python connect.py -p COM1 -s 115200

# On linux (Note: Please add your user to the dialout group for accessing
# the tty devices without root privileges)
$ python connect.py -p /dev/ttyS0 -s 115200

# Read your current boot loader and firmware version
$ python firmware.py info
Modem version (rev): 00.006
Applicative version (arn): 01.400.00
Bootloader version (sbl): SBL1_107_1

# Download the new boot loader onto the module if required, i.e., if the
# *.usf file version is prior to the version number under SDK/firmware.
# In this case: 107 < 118.
$ python firmware.py update ..\firmware\exs_tx\SBL1_118.usf

# Download the new firmware onto the module
$ python firmware.py update ..\firmware\exs_tx\exs82.usf

# For more firmware commands and options please consult help
$ python firmware.py --help
```

### Notes:

With the new firmware, both the ASC0 interface, as well as the native USB interface, can be used to control the module. However, with this SDK and for embedded processing purposes, Thales recommends to employ the native USB interface to control the modem. It will allow the embedded application to use the ASC0 interface for other purpose (such as driving an external sensor or chipset), and enable more debugging tools.

So, please power down the module, and connect the USB cable from the PC to the native USB interface of the LGA DevKit before continuing.

**Notes:**

With EXSx2/TXx2, the USB port for the firmware update must be a modem port.

With PLS83-W, the USB port for the firmware update can be any enumerated ACM port (Hardware IDs: USB\VID\_1E2D&PID\_006x&MI\_00/02/04), except for the diagnostic port and the Ethernet interface.

**Notes:**

Firmware updates in a Linux environment are done using the tool glinswup located under "\SDK\tools\linux\". The tool was tested with Ubuntu14, Ubuntu16 and Ubuntu18 Linux. It is not recommended to use the tool with earlier Ubuntu versions.

## 2.3 Connecting PC and Module

Following the steps described in [Section 2.1](#) and [Section 2.2](#), please proceed as follows to connect the Linux or Windows PC to the LGA DevKit. Here we assume power is supplied to the LGA DevKit over the USB connector, which is the default setting.

1. Connect the native USB interface on the LGA DevKit to the PC using the USB cable.
2. Press the “ON” button on the LGA DevKit to start up the module.
3. With a Windows PC please install the USB drivers to be able to access the available USB interfaces. For a detailed description on how to do this as well as the USB enumeration please refer to [Section 2.4](#).
4. With a Linux PC please install the option driver for the vendor specific USB Diag port as follows:

```
# load the option driver
sudo modprobe option

# add new VID:PID to option driver
sudo sh -c "echo '1e2d 006B' > /sys/bus/usb-serial/drivers/option1/
new_id"
```

The USB interfaces will then enumerate as follows:

/dev/ttyACM0	Modem port for AT commands. Normal CDC-ACM driver used.
/dev/ttyACM1	2 <sup>nd</sup> USB port for AT commands. Normal CDC-ACM driver used.
/dev/ttyUSB0	Diagnostics port. Here, Option driver is required.

5. Open the connection via the native USB interface.

```
# Connect to the target specifying your port (on windows) or your device
# (on linux) and speed. If successful, these communication parameters
# will be saved locally (config.db) for next commands.

# On linux (Note: Please add your user to the dialout group for accessing
# the tty devices without root privileges)
$ python connect.py -p /dev/ttyACM0 -s 115200
Connection with the target on /dev/ttyACM0 at 115200
Done !

# On windows, specify the modem port as displayed in the figure shown
# in Section 2.4.
$ python connect.py -p COM15 -s 115200
Connection with the target on COM15 at 115200
Done !
```

Now, the basic setup of the SDK environment is complete, and embedded applications can be managed, as described in [Chapter 3](#).

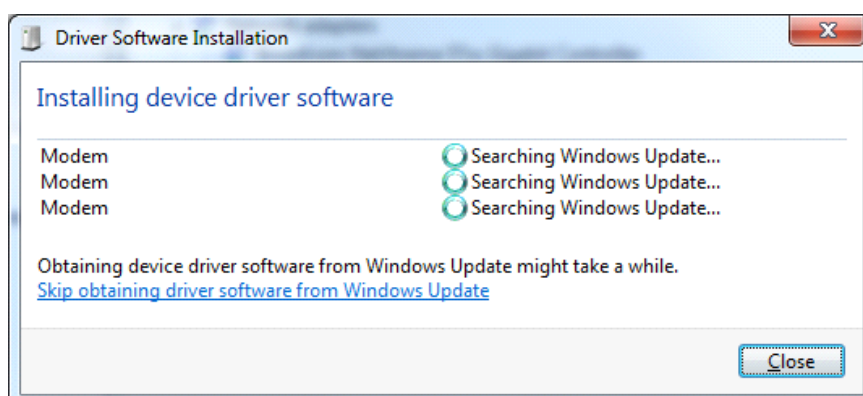
## 2.4 USB Driver Installation (Windows only)

Under Windows only, and on the first module startup over the native USB interface the USB driver needs to be installed as described in the sections below for Windows 7 ([Section 2.4.1](#)) and Windows 10 ([Section 2.4.2](#)) to be able to access the available USB interfaces.

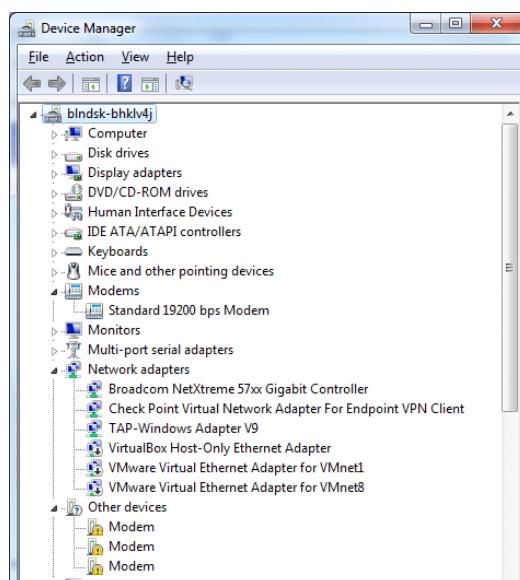
### 2.4.1 Windows 7

Under Windows 7, please install the USB driver provided by Thales as follows:

1. Close the initial driver installation window.

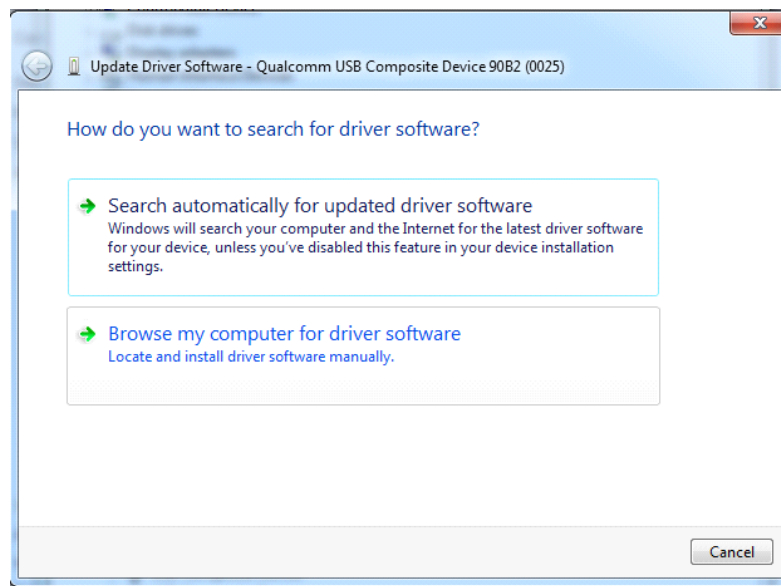


2. Open Device Manager, right click on the Modem devices under “Other Devices” one by one, and select “Update Driver Software” from the pop-up menu.

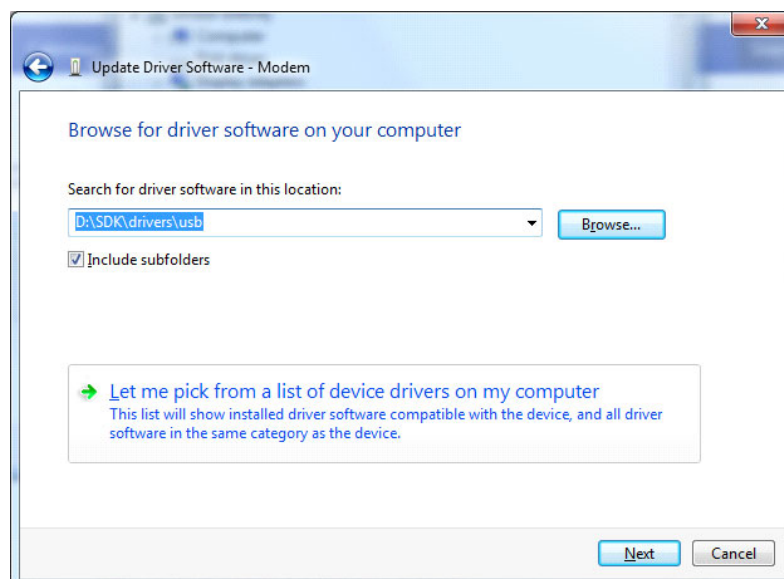


## 2.4 USB Driver Installation (Windows only)

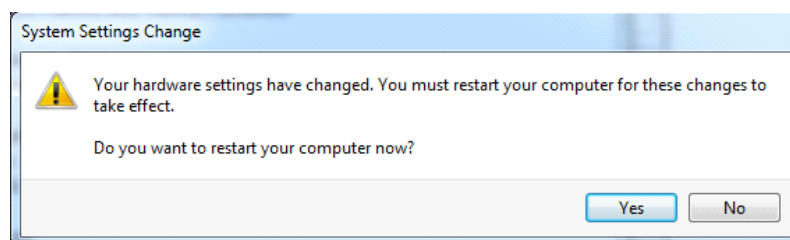
3. Select “Browse my computer for driver software” for each modem device.



4. Select the USB driver path from the SDK folder for each modem device.



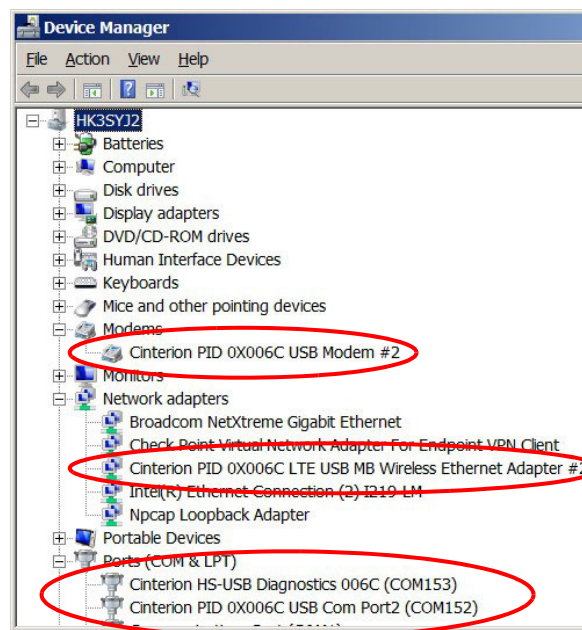
5. Restart your PC if asked to do so.



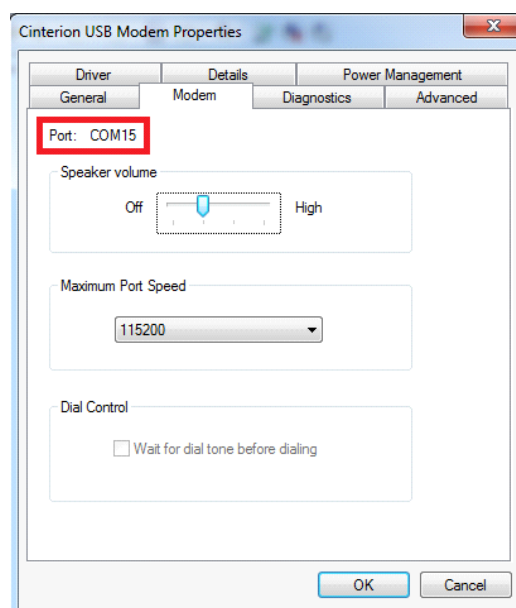
## 2.4 USB Driver Installation (Windows only)

6. Check - in the Device Manager - that the interfaces are available now. Logging output can be written to any of the AT command ports (i.e., the Modem port or USB port):

- Cinterion USB Modem: Modem port for AT commands
- Cinterion Ethernet Adapter: Ethernet adapter
- Cinterion Diagnostics: Diagnostics port
- Cinterion USB Com Port2: 2<sup>nd</sup> port for AT commands



7. Right click on the Modems port (Modems > Cinterion USB Modem) and select Properties. On the “Modem” tab check for the COM port number.

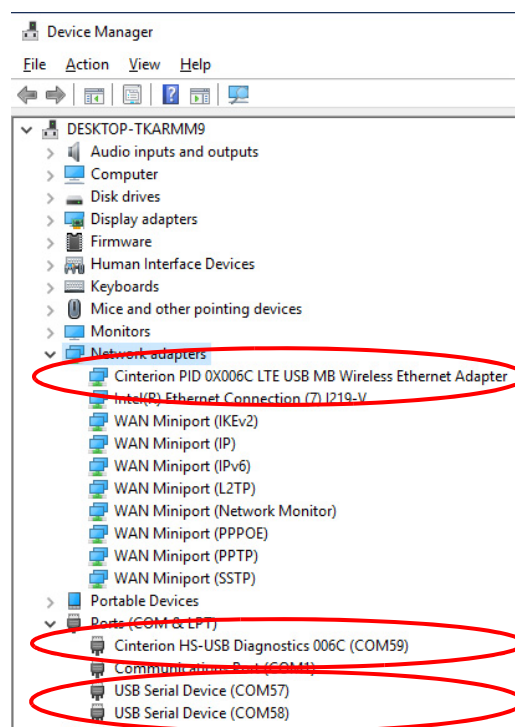


## 2.4.2 Windows 10

Under Windows 10, the standard USB driver included with the operating system is installed by default, and the following devices should then be available:

- Cinterion Ethernet Adapter: Ethernet adapter
- Cinterion Diagnostics: Diagnostics port
- 2 x USB Serial Device: Port for AT commands

Logging output can be written to any of the AT command ports.



Please note that as an option, it is also possible to install the USB driver provided by Thales in the same manner as described above in [Section 2.4.1](#) for Windows 7. In this case, one of the two default USB Serial Devices will also be enumerated as Modem port.

## 3 Managing Embedded Applications

This chapter describes basic management functionalities for your embedded applications.

### 3.1 Building Applications

As a first application sample there is a “Hello World” program included in the SDK. You can build the sample as follows; this will produce an unsigned application binary file.

```
# On any platform
$ cd SDK/examples/helloworld
$ python build.py -p [pls|exs_tx]

# On Linux:
$ cd SDK/examples/helloworld
$ ./build.sh -p [pls|exs_tx]

# On Windows:
$ cd SDK\examples\helloworld
$ build.bat -p [pls|exs_tx]

Application binary is SDK\examples\hello_world\build\helloworld.bin
```

*-p [pls|exs\_tx]* specifies the product (family) the sample application will run on. *<p/s>* stands for PLSx3 products, *<exs\_tx>* stands for EXSx as well as TXx products. If the *-p* parameter is omitted, *-p exs\_tx* is used as default setting.



## 3.2 Managing Signed Applications

### 3.2.1 Querying Secure Boot Status and Signature Parameters

The secure boot status of a module is shown in the information summary:

```
$ python app.py info
Name          Version  Status      Size  Location
0 application on target, 0 running
Autostart config: disabled
Secure boot: enabled
App signature config: ECC with sha384
```

By default secure boot is enabled, on the module, and therefore, all applications need to be signed in order to be executed on the module. The unsigned binary "helloworld" will not run yet.

The signature parameters currently configured for the module are also shown in the information summary. The default signature setting is ECC with sha384, which is recommended. These signing parameters can be configured to different ones. Please check `python app.py config --help` for more details.

Note: If the signature setting is RSA, please note, the signed process will be using ASN encoding.

On pre-release module samples the secure boot feature is most likely disabled, thus the [Managing Signed Applications](#) section can safely be ignored.

## 3.2.2 Generating the Application Root of Trust

In this section we will create the files needed to sign your "hello world" application. This operation can be done without the module being connected.

All examples use ECC keys since it is the default setting for signing. The following OpenSSL commands will create a new ECC key pair using curve SECP384R1, and a self-signed certificate for the application root of trust.

```
$ openssl ecparam -name secp384r1 -out secp384r1.param.pem
$ openssl ecparam -in secp384r1.param.pem -genkey -noout -out
app_rot.key
$ openssl ec -in app_rot.key -out app_rot.pub -pubout
$ openssl req -new -sha384 -x509 -out app_rot.der -outform DER -key
app_rot.key -days <days> -set_serial <serial> -subj <subject>
```

The parameters <days>, <serial>, and <subject> from the above example will have to be replaced as required for the certificate. An example for a full command would be:

```
$ openssl req -new -sha384 -x509 -out app_rot.der -outform DER -key
app_rot.key -days 7300 -set_serial 1 -subj /C=DE/ST=Berlin/L=Berlin/
O="THALES DIS AIS Deutschland GmbH"/OU="R&D"/CN="Demo App Root of Trust"
```

The public key can be calculated using the private key, or extracted from the certificate using the following OpenSSL command, assuming the certificate file is app\_rot.der, in DER format:

```
$ openssl x509 -in app_rot.der -inform DER -pubkey -noout > app_rot.pub
```

The above listed operations create the following application root of trust files:

- app\_rot.key contains the private key in PEM format (protect this file; do not lose or distribute it)
- app\_rot.pub contains the public key in PEM format (needed in [Section 3.2.4](#))
- app\_rot.der contains the public certificate in DER format (needed in [Section 3.2.5](#))

## 3.2.3 Signing an Application

Use the python script app.py to sign an application using your private key. This operation can be done without a module being connected.

The following command will sign your application binary file built in the "hello world" example (see [Section 3.1](#)) using the private key file app\_rot.key (see [Section 3.2.2](#)):

```
$ python app.py sign --key app_rot.key examples/helloworld/build/helloworld.bin
```

A pre-built example of a "hello world" application, signed with other keys, also available here: `../examples/helloworld/build/signed/helloworld.bin`.

### 3.2.4 Verifying an Application

Use the python script app.py to verify a signed application against a public key. This operation can be done without a module being connected.

The following command will verify your signed "hello world" application binary against your public key file app\_rot.pub (see [Section 3.2.2](#)):

```
$ python app.py verify --pubkey app_rot.pub --keyform pem  
..\examples\helloworld\build\signed\helloworld.bin  
Verification OK
```

### 3.2.5 Installing and Protecting the Application Root of Trust in the Module

In order to verify the signature of an application, the module needs the corresponding public information of the application root of trust. The root of trust can be installed as certificate file in DER format (created in [Section 3.2.2](#)).

By default this part of the secure boot infrastructure is left open for you to customize. This means that the protection of the Application Root of Trust is not active; it can be installed, deleted, and overwritten.

Of course we recommend that, in your final production, you complete the secure boot model of operation by turning on Application Root of Trust security.

Once secure mode is activated the Application Root of Trust certificate cannot be deleted or modified unless commands are signed with the corresponding private key. All details regarding management of secure mode and usage of the certificate tools are described in [\[4\]](#): "Certificate for Local Module Management".

For development and testing you can continue to use your module in unsecured mode.

To send the application root of trust DER file to the module you must create a binary file to send to the module. We will call this file "LoadAppRotCert.bin"; it is essentially a secure binary AT Command.

Ensure that a Java desktop runtime environment is installed, to allow you to run the Java tools from the 7z archive attached to the AN62 PDF file. For details see [Section 1.3](#). From it save and open the PDF embedded file "exs62-w\_exs82-w\_tls\_tools.7z". Expand amongst others the file called ".\exs62-w\_exs82-w\_tls\_tools\Tools\bin\win-x86\cmd\_IpCertMgr.jar", and use the following Windows command to build the file "LoadAppRotCert.bin":

```
java -jar .\exs62-w_exs82-w_tls_tools\Tools\bin\win-x86\cmd_IpCertMgr.jar  
-mode app_rot -cmd writecert -certfile app_rot.der -certIndex 0 -sigType NONE  
-file .\LoadAppRotCert.bin
```

For development and test purposes you can use your module in unsecured mode:

Ensure that your module is not locked. AT^SSECUC="SEC/MODE" should return 0 as below:

```
AT^SSECUC="SEC/MODE"  
^SSECUC: "SEC/MODE",0  
OK
```

You can use one simple AT Command to copy the file "LoadAppRotCert.bin" to the module:

```
AT^SBNW="app_rot",1  
CONNECT  
SECURE CMD READY: SEND COMMAND ...  
  
[Send BINARY file "LoadAppRotCert.bin"]  
  
SECURE CMD END OK  
OK
```

You can use one simple AT Command to read the stored App\_Rot DER file from the module:

```
AT^SBNR="app_rot"  
^SBNR: 0, size: "562", issuer: "..."  
OK
```

**Note:** For production you will want to switch your module into Secure Operation mode. The above, simplified commands become somewhat more complex.

However, this will prevent a 3rd party from modifying your AT^SBNR="app\_rot" DER certificate on the module, amongst other things.

```
AT^SSECUC="SEC/MODE"  
^SSECUC: "SEC/MODE",1 or 2  
OK
```

### 3.3 Downloading and Starting Applications

Use the python script app.py to manage your application on the module.

```
# List the application loaded on the target
$ python app.py info
Name                Version  Status      Size  Location

0 application on target, 0 running
Autostart config: disabled
Secure boot: enabled
App signature config: ECC with sha384

# Download your application on the target
$ python app.py download ..\examples\helloworld\build\signed\
helloworld.bin

# List again the application loaded on target
$ python app.py info
Name                Version  Status      Size  Location
helloworld          21027   stopped    10.4 KB  A:/helloworld.bin

1 application on target, 0 running
Autostart config: disabled
Secure boot: enabled
App signature config: ECC with sha384

# To view the output, open a second terminal to start the logging tool
# on the 2nd USB port (for Windows see Section 2.4, for Linux use
# /dev/ttyACM1)
$ python log.py read -d COM41

# Switch back to the first terminal and start the application
$ python app.py start helloworld

# Stop your application
$ python app.py stop helloworld

# For more commands and options about your applications (such as delete,
# rename, config), please consult help
$ python app.py --help
```

#### 3.3.1 Limitations

There are a few limitations with running embedded applications:

- You have to stop an application before you can start it again.
- Autostart delay is only possible in a range between 2s to 30s.

## 3.4 Logging

The SDK provides a toolkit allowing to extend the application code with debug messaging. This includes both embedded software with APIs as well as associated PC tools. The toolkit manages log filtering according to log message severity specified in the application code as well as timestamps as of module startup.

```
# To discover the logging capabilities, please build and download the
# logging example first (from SDK\examples\logging)
# This example puts out all log levels every 15 seconds

# Start the application
$ python app.py start logging

# Start the logging tool to read the logging output from the 2nd USB port
# (for Windows see Section 2.4, for Linux use /dev/ttyACM1)
# and filter logs by levels to keep only from 0-critical to 3-info
$ python log.py read -d COM41 -l 3
Cinterion Logging Tool ++++++
Start logging on USB Logging port (COM41) at 2020-03-18 16:03:26...
(use Control-C to exit)
05:23:23:365 CRIT:logging.c,127: [2] Critical message 0
05:23:23:366 ERR :logging.c,128: [2] Error message 1
05:23:23:366 HIGH:logging.c,129: [2] High message 2
05:23:23:367 INFO:logging.c,130: [2] Info message 3
# Use Control-C to exit

# Stop the application
$ python app.py stop logging

# Feel free to redirect the output using '> FILENAME'
$ python log.py read -d COM41 -l 3 > toto.txt
# For more commands and options about your logs, please consult help
$ python log.py --help
```

Please refer to the `Gina_Service_API` documentation, "Userware Logging", for more information about the embedded APIs.

**Note:** Every AT command port can be switched into logging mode. In this mode it is not possible to enter AT commands. The logging output to such a port is transparent, the logs are written without any protocol encapsulation. So, you can use a terminal application such as putty (Windows) or cutecom (Linux) to capture the logs. You just need to configure the module accordingly using "log.py config".

**Note:** As an alternative to the transparent output in logging mode you can put out logs as URCs (Unsolicited Result Codes) on the AT communication port. This interface however, is not recommended for a large amount of log messages. Another alternative would be to use the USB QXDM port - but this option requires an appropriate Qualcomm QXDM license. Please call "log.py config --help" for more information about setting these alternative ports.

## 3.5 Exception Handling

While developing or running an embedded application (APP) on the module, unexpected exceptions might occur. These exceptions should be analyzed and handled to resolve possible (runtime) errors within the APP.

To support APP developer with this kind of analysis and troubleshooting, each APP can store the last four exceptions on the module for later off-line processing on a PC connected to the module via USB. Please refer to [Section 4.2](#) for more details on how to develop an exception handler.

Now, the python script *backtrace.py* enables the APP developer to manage the exceptions:

- List the available backtraces (or also called stack dumps) per APP. The most recent trace has the internal number 1, the oldest one the number 4. See also [Section 3.5.1](#).
- Remove individual or all of the stored backtraces for an APP (to free some flash space and discard unused old traces). See also [Section 3.5.2](#).
- Process the backtrace, by providing the origin of the exception and the complete call stack with source line information. Such a report will normally contain valuable information, enabling the developer to rapidly explore the root cause of an exception, and to be able to remove it. Only in rare cases will it not be possible to analyze the call stack because of optimized code generation with absolute branches. See also [Section 3.5.3](#).

### 3.5.1 List Available Backtraces

To list the stored backtraces of a certain APP, the APP name must be given as a parameter:

```
$ python backtrace.py list exception3  
[1, 2, 4]
```

The expected output is a list of trace numbers - this list can be empty, if no trace exists. In this case please check the spelling of the APP's name.

### 3.5.2 Remove Backtraces

To remove stored backtraces of a certain APP, the APP's name as well as the number of the backtrace must be given as parameter ("all" removes all available backtraces of this APP):

```
$ python backtrace.py remove exception3 4  
Dump 4 of APP exception3 removed
```

The expected output is a message confirming the successful deletion of the backtrace.

If there is no backtrace, an error message is put out:

```
$ python backtrace.py remove exception_7 all  
Dump 1 of APP exception_7 removed  
Dump 2 of APP exception_7 not removed - not present  
Dump 3 of APP exception_7 removed  
Dump 4 of APP exception_7 removed
```

### 3.5.3 Process a Backtrace

The python script *backtrace.py* provides four parameters to process an APP's backtrace:

- APP name (must be the same as used to start the APP)
- Backtrace number (1 - 4)
- Source path of APP (map-file and elf-file must be present in subdirectory "build")
- Tool path of host-tools (directory "tools" in sdk/SDK)

```
$ python backtrace.py process exception3 1 ..\examples\exception3 .  
Process Dump 1 of APP exception3...  
Received 100 %  
sp is 0x43004c78, stackStart is 0x43003d30  
lr is 0x43000254  
pc is 0x430001e8  
*** Exception occurred at PC 0x430001e8 in function Func3 at line #139  
in exception3.c  
*** Called from 0x43000250 in function Func2 at line #154 in exception3.c  
*** Called from 0x43000278 in function Func1 at line #162 in exception3.c  
*** Called from 0x430002f8 in function _APP_main at line #184 in excep-  
tion3.c  
LR 0x0 - end of call stack detected
```

Please refer to [Section 4.2](#) for more details on how to develop an exception handler issuing the information contained in a backtrace.

## 3.6 Navigating Module File System

Use the python script *fs.py* to explore the module's file system. The script allows you to:

- Get statistics about the module file system drive 'A:/'
- Get statistics on each file (such as size)
- Get a list of files and directories
- Create/remove directories
- Upload/download files from/to PC
- Copy/rename/remove files
- Calculate file crc

```
# For further details about the fs tool, please consult help  
$ python fs.py --help
```



## 3.7 Monitoring Application RAM

The consumed RAM size for an IoT application comprises several parts, including execution regions like RO (Read Only), RW (Read Write), ZI (Zero Initialize), thread stack, and callback stack memory. Additionally, ThreadX requires another 160KB RAM for running application based activities.

The overall memory (RAM) size of a module can be found in [2]. To avoid out-of-memory issues, the following condition should be observed:

$$(ER\_RO + ER\_RW + ER\_ZI + \text{module start/stop thread stack size} + \text{module callback thread stack size} + 160KB) < (\text{overall RAM size})$$

The values of ER\_RO, ER\_RW and ER\_ZI can be determined from the generated .map file. The values for the module thread stack size and the module callback thread stack size can be found in the related .S file (preamble file).

If taking the helloworld application as an example - located under SDK\examples\helloworld - the following values can be determined:

ER\_RO, ER\_RW, ERZI size shown in "helloworld.map" file:

ER_RO	0x43000000	<b>0x305c</b>	# Offset: 0x58, LMA: 0x43000000
ER_RW	0x43003060	<b>0x24</b>	# Offset: 0x30b8, LMA: 0x43003060
ER_ZI	0x43003088	<b>0x464</b>	# Offset: 0x30dc, LMA: 0x43003088

Thread stack size shown in .S file:

.word	<b>8192</b>	@ Module Start/Stop Thread Stack Size
.word	<b>2046</b>	@ Module Callback Thread Stack Size

## 4 Developing Embedded Applications

### 4.1 Available APIs

To develop your application, quite a number of pre-integrated services are delivered as part of the firmware and are exposed to your application as APIs. These APIs are documented under SDK/doc. The following table lists the status for every documented API as well as associated examples.

API Document	Chapter	Supported		Example
		EXSx2/TXx2	PLSx3	
Gina_Service_API	Sending AT Commands	Yes	Yes	SDK\examples\atcommand
Gina_Service_API	Serial Device	Yes	Yes	SDK\examples\serial
Gina_Service_API	Safe Update	Yes	Yes	SDK\examples\update_fw SDK\examples\update_app
Gina_Service_API	Userware Logging	Yes	Yes	SDK\examples\logging
80-P8102-1	4 - DSS Net Control APIs	Yes	Yes	SDK\examples\ping
			No	SDK\examples\non_ip
80-P8102-1	5 - QAPI Networking Socket	Yes	Yes	SDK\examples\sockv4 SDK\examples\sockv6
80-P8102-1	6 - QAPI Network Security APIs	Yes	Yes	SDK\examples\tls
80-P8102-1	7 - QAPI Networking Services	Yes	Yes	SDK\examples\ping
80-P8102-1	7.16 - Constrained Application Protocol (CoAP)	Yes	No	SDK\examples\coap
80-P8102-1	8 - Domain Name System Client Service APIs	Yes	Yes	SDK\examples\dns
80-P8102-1	9 - MQTT API	Yes	Yes	SDK\examples\mqtt
80-P8102-1	10 - HTTP(S) APIs	Yes	Yes	SDK\examples\http
80-P8102-1	13 - LWM2M APIs	No	No	
80-P8102-1	14 - AT Forward Service Framework	No	No	
80-P8102-1	15.1 - Driver Access APIs for the APP Application Space	No	No	
80-P8102-1	15.2 - Command Line Interface	No	No	
80-P8102-1	16 - Sensor Manager APIs	No	No	
80-P8102-2	4.1 - GPIO Interrupt Controller APIs	Yes	Yes	SDK\examples\gpio SDK\examples\gpio_int
80-P8102-2	4.2 - PMM APIs	No	No	
80-P8102-2	5 - Diagnostic Services Module	No	No	
80-P8102-2	6 - Pulse Width Modulation	No	No	
80-P8102-2	7.2 - File System APIs	Yes <sup>1</sup>	Yes	
80-P8102-2	7.3 - FTL Data Types and APIs	No	No	
80-P8102-2	8.2 - USB APIs	No <sup>2</sup>	No	
80-P8102-2	9.1 - I2C Master APIs	Yes	Yes	SDK\examples\i2c

## 4.1 Available APIs

API Document	Chapter	Supported		Example
		EXSx2/TXx2	PLSx3	
80-P8102-2	9.2 - SPI Master APIs	Yes	No	
80-P8102-2	9.3 - UART APIs	No <sup>2</sup>	No <sup>2</sup>	SDK\examples\serial
80-P8102-2	10.1 - Timer APIs	Yes	No	SDK\examples\helloworld
80-P8102-2	10.2 - PMIC RTC APIs	Yes (roadmap)	No	
80-P8102-2	10.3 - PMIC Battery Status Information	Yes (roadmap)	No	
80-P8102-2	11.2 - ADC APIs	Yes	Yes	SDK\examples\adc
80-P8102-2	11.4 - TSENS APIs	No	No	
80-p8102-3	4 - GNSS Location Driver	Yes	Yes	SDK\examples\location
80-p8102-4	3 - System Power Save Management	Yes	No	SDK\examples\psm

<sup>1</sup>. QAPI\_FS\_Itr\_\* are not allowed to be used on EXSx2//TXx2 because of memory issues.

<sup>2</sup>. API has been replaced by Serial Device from Gina Service API.

## 4.2 Exception Handler Development

An embedded application (APP) is an application running in user-mode without system privileges and with restricted access to module-resources only. Peripherals and system services can only be accessed via calls to the operating system, memory can be accessed only as virtual memory under strict control of a Memory Management Unit (MMU).

If an APP breaks these rules and tries to access memory or resources that are protected, a processor exception is raised. Now, exception handling covers two aspects:

- Handling of somehow expected exceptions during application operation.
- Handling of unexpected exceptions during application development.

An APP can install an exception handler function that is triggered if an exception occurs. The exception handler manages exceptions that are to be expected under certain circumstances, and that can be handled by the application.

The handler firstly gathers information about the exception, then acts according to the situation at hand, and finally decides how to proceed:

- Ignore the exception and proceed with the next instruction after it. Ignoring an exception may mean that an illegal memory access is ignored and as a consequence the target of this access does not change its value but remains as before.
- Restart the APP without re-initializing the data. It must be emphasized however, that a restart of the APP as result of an exception does not set back all data to initial values as included in the APP image. Instead such re-initialization must be done by the exception handler explicitly or in a registered cleanup function. This is specifically true for allocated heap memory that is not automatically freed. To overcome a memory shortage situation by simply restarting the APP is therefore not possible - in such a case the allocated memory must be freed explicitly instead.
- Stop and unload the APP.

For the latter cases "Restart" as well as "Stop and unload", an APP cleanup function is called, if it has been registered by the APP via `gina_uwmod_set_cleanup`.

The exception handler has full access to all APP resources and data variables same as the APP's main thread has. It can therefore repair, modify and release internal data structures. It can also use certain APP variables to get an idea about what operation failed, and can try to rectify this. This is similar to "try {} - catch {}" scenarios in C++ exception handling. The handler can even change processor registers to influence further processing after returning from the exception handler - this however requires deep knowledge of the meaning of each of the registers for this specific exception and is not recommended.

A standard exception handling for unexpected exceptions would be to output information about an exception, and then hold and unload the APP.

### 4.2.1 Definition

An exception handler

- Gets a single parameter that is a pointer to a struct of type APPEXception and
- Returns an exception type APPEXceptionType that describes the further processing

```
typedef enum {
    APP_EXCEPTION_HOLD_AND_UNLOAD,
    APP_EXCEPTION_SKIP_AND_PROCEED,
    APP_EXCEPTION_HOLD_AND_RESTART
} APPEXceptionType;
APPEXceptionType _exception(volatile APPEXception *exp);
```

The APPEXception structure describes the exception in all its detail down to the processor architecture, e.g., an ARM7:

```
typedef enum {
    APP_EXCEPTION_BACKTRACE_REQUIRED,
    APP_EXCEPTION_BACKTRACE_NOT_REQUIRED
} APPEXceptionBackTrace;

typedef struct {
    APPEXceptionType type;
    APPEXceptionBackTrace backTrace;
} APPEXceptionAction;

typedef struct {
    uint32 data;
    uint32 instruction;
} FaultStatus;

typedef struct {
    uint32 regs[13];
    uint32 sp;
    uint32 lr;
    uint32 pc;
    uint32 cpsr;

    void *thread;
    uint32 cause;

    uint32 fault_address;
    FaultStatus fault_status;

    uint32 exceptionCounter;
    APPEXceptionAction action;
} APPEXception;
```

Member `regs[]` holds the contents of the ARM registers R0-R12 at the point in time when the exception occurs.

Member `sp` holds the contents of the stack-pointer, `lr` is the link-register and `pc` the program counter that points to the instruction causing the exception.

Pointer `thread` points to the OS thread that causes the exception, as it can be retrieved via OS-call `tx_thread_identify()`. This should always refer to the APP thread executing the function `_app_main()`- any other value is unexpected.

The member `cpsr` is the Current Processor Status Register of the ARM processor, which refers to the PSR at the time of the exception. It is expected that its lower 5 bits equal to 0x10 specifying the user-mode without any privileged access. Any other value of the lower 5 bits is unexpected.

Member `cause` describes the exception class with following different exception classes:

```
#define ERR_UNDEFINED_INSTRUCTION 2
#define ERR_ABORT_DOMAIN_FAULT 3
#define ERR_ABORT_PERMISSION_FAULT 4
#define ERR_ABORT_TRANSLATION_FAULT 5
#define ERR_ABORT_ALIGNMENT_FAULT 6
#define ERR_PRECISE_EXTERNAL_ABORT 6
#define ERR_ABORT_TLBMISSE_FAULT 7
#define ERR_ABORT_DEBUG_EXCEPTION 8
#define ERR_ABORT_ICACHE_FAULT 9
#define ERR_ABORT_PAGETABLE_FAULT 10
#define ERR_ABORT_TLBLOCK_FAULT 11
#define ERR_COPROCESSOR_DATA_ABORT 12
#define ERR_ABORT_PARITY_FAULT 13
#define ERR_ABORT_UNDEFINED 14
```

Members `fault_status` and `fault_address` hold the contents of the corresponding ARM registers, describing the current exception.

The `exceptionCounter` finally is incremented by the OS for each exception of this APP. It has no relation to the exception itself other than giving a unique number to differentiate between several exceptions that may occur one after the other. It also can be used in the exception handler to check whether a certain exception situation was successfully resolved or not, and to terminate the whole APP if exceptions are generated too often.

Member `action` holds two subcomponents:

- `type`
- `backtrace`

While the first is ignored (instead the return value of the exception handler is respected) the second controls whether a small backtrace image of some KB is stored internally to the flash filesystem of the module. The default value is `APP_EXCEPTION_BACKTRACE_REQUIRED`, but the exception handler can overwrite it with `APP_EXCEPTION_BACKTRACE_NOT_REQUIRED` and avoid the generation of the backtrace file.

The backtrace file supports an off-line analysis of an exception, which is useful during the development of the APP. The generation of this file however takes some time and also flash space that a sophisticated exception handler may want to save.

## 4.2.2 Sample

```
int errFatalFlag = 0;
char errFatalString[128];

#define ERR_FATAL(s, param1, param2, param3) ERR_FATAL_HANDLER(s,
__LINE__, param1, param2, param3)

void ERR_FATAL(char *string, int param1, int param2, int param3)
{
    char s[128];
    sprintf(s, string, param1, param2, param3);
    sprintf(errFatalString, "FATAL error in line #%d: %s", __LINE__, s);

    errFatalFlag = 1;
    *(unsigned long *)0xffffffff = 1;
}
```

The makro `ERR_FATAL` serves to add a panic-handling to the APP. It calls the function `ERR_FATAL_Handler` adding the line of the call and then intentionally raises an exception after storing an error string and setting the `errFatalFlag`. When the exception handler is called, it initially checks the flag and the `fault_address`, and in case both match to the panic-handling, it puts out the panic string and finally causes the system to hold and unload the APP.

It also checks the `fault_address` for the value `0x0` to specifically react to NULL-pointer accesses.

```
// APP exception-handler
// Called in context of exception thread in user mode with access to APP
resources
APPExceptionType _exception(volatile APPException *exp)
{
    int i;

    // init gina output - this must be done on a per-thread basis and this
    // exception handler is run in the context of an APP exception thread
    gina_init();

    // check for ERR_FATAL panic handling
    if (errFatalFlag && (exp->fault_address == 0xffffffff)) {
        gina_uwlog_printf("APP %s", errFatalString);
        return APP_EXCEPTION_HOLD_AND_UNLOAD;
    }

    if (exp->fault_address == 0x0) {
        gina_uwlog_printf("APP: NULL pointer accessed @ pc 0x%x, cause %d",
            exp->pc, exp->cause);
        return APP_EXCEPTION_HOLD_AND_UNLOAD;
    }
}
```

After checking for special panic-handling and NULL-pointer access a generic exception message is put out with details that should shed some light on the reasons for the unexpected exception.

```
Gina_uwlog_printf(" ");
gina_uwlog_printf("APP Exception occurred:");
gina_uwlog_printf("=====");
gina_uwlog_printf("Fault at PC 0x%x when accessing data at 0x%x",
exp->pc, exp->fault_address);

gina_uwlog_printf("Faulting thread 0x%x, cause 0x%x", exp->thread,
exp->cause);

for(i=0; i<=12; i++) {
    gina_uwlog_printf("r%02d: 0x%x", I, exp->regs[i]);
}

gina_uwlog_printf("sp is 0x%x", exp->sp);
gina_uwlog_printf("lr is 0x%x", exp->lr);

gina_uwlog_printf("exceptionCounter is 0x%x", exp->exceptionCounter);

return APP_EXCEPTION_HOLD_AND_UNLOAD;
}
```



### 4.2.3 Installation

The exception handler is called in the context of a separate exception thread that is generated alongside the main APP thread and a system related APP callback thread. This exception thread needs some stack space and an idea about the function it should call as exception handler.

Both is specified in the preamble-part of the APP, and is OS dependent and related to ThreadX.

The exception handler has to be specified in field "Reserved 2" at offset 0x48, its stack size in field "Reserved 3" at offset 0x4c:

```
p2align 2
.word    _exception - . + . @ Reserved 2 - exception handler
.p2align 2
.word    4096               @ Reserved 3 - stack size of exception
                          @ thread - must be added to Start/Stop
                          @ Thread Stack Size
```

The size of the stack of the exception thread must also be added to the preamble at offset 0x28 in the field "Module Start/Stop Thread Stack Size":

```
.p2align 2
.word    4096 + 4096        @ Module Start/Stop Thread Stack Size +
                          @ stack size of exception thread
```

If no exception handler is installed, a default handler is used that does nothing and simply returns APP\_EXCEPTION\_HOLD\_AND\_UNLOAD after calling the cleanup function and generating the backtrace file.

If the specified stack size is too small (at least 200 bytes are necessary even for the default handler) an internal default stack of size 200 bytes is used.

## 4.3 Implementing Multiple Embedded Applications in Parallel

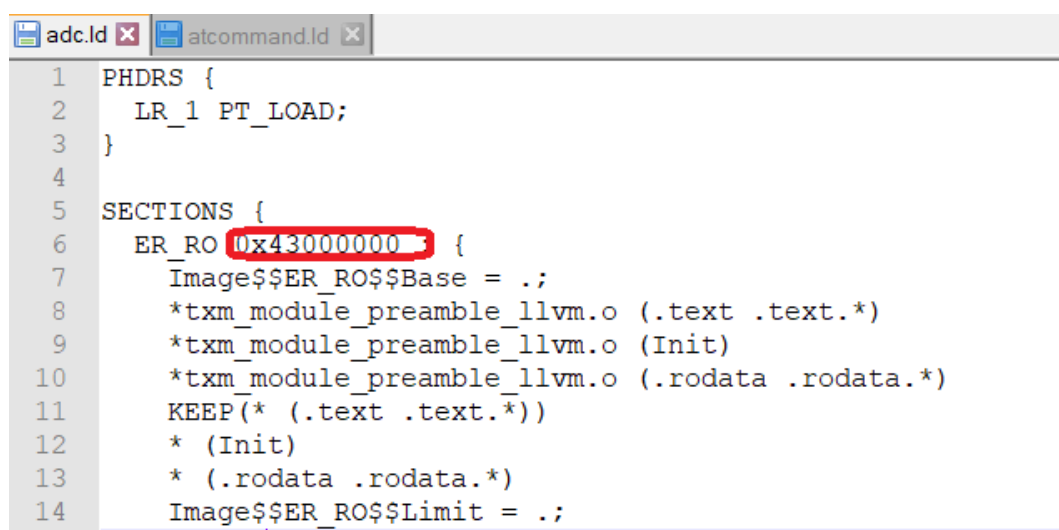
The supported products provide a ThreadX framework for embedded applications to load modules dynamically that are built separately from the resident component of the application. This section shows how to implement multiple embedded applications running in parallel.

### Virtual Address

Each application must be allocated for a unique virtual address range in \*.ld in order to run in parallel.

Examples:

ADC application, VA base address – 0x43000000

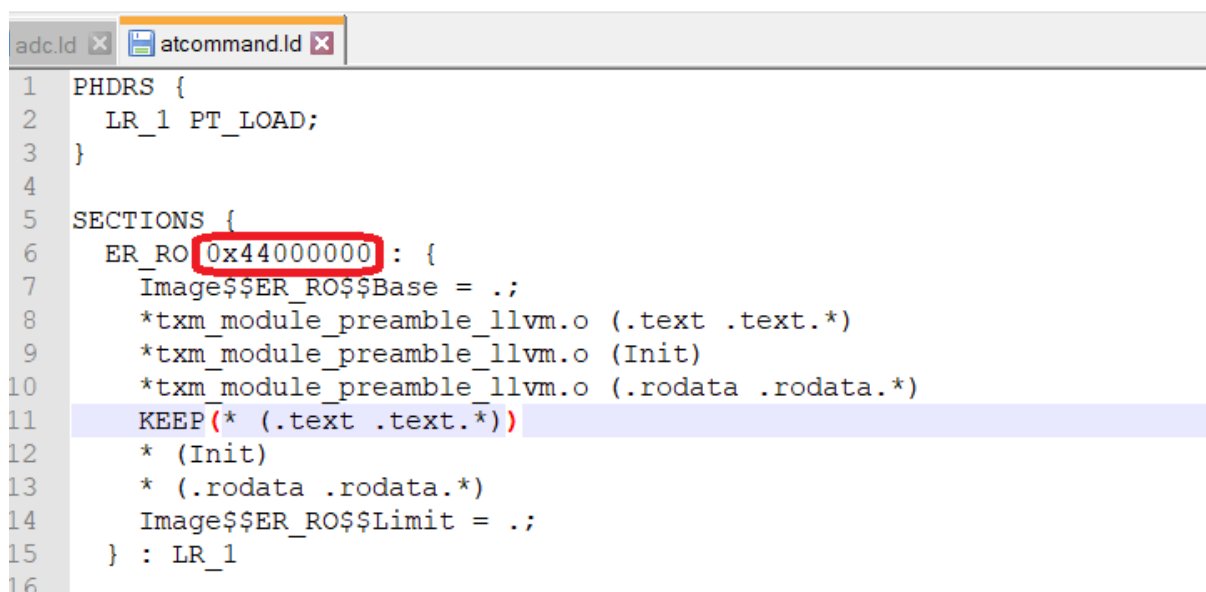


```

1  PHDRS {
2      LR_1 PT_LOAD;
3  }
4
5  SECTIONS {
6      ER_RO 0x43000000 : {
7          Image$$ER_RO$$Base = .;
8          *txm_module_preamble_llvm.o (.text .text.*)
9          *txm_module_preamble_llvm.o (Init)
10         *txm_module_preamble_llvm.o (.rodata .rodata.*)
11         KEEP(* (.text .text.*))
12         * (Init)
13         * (.rodata .rodata.*)
14         Image$$ER_RO$$Limit = .;

```

ATCommand application, VA base address – 0x44000000



```

1  PHDRS {
2      LR_1 PT_LOAD;
3  }
4
5  SECTIONS {
6      ER_RO 0x44000000 : {
7          Image$$ER_RO$$Base = .;
8          *txm_module_preamble_llvm.o (.text .text.*)
9          *txm_module_preamble_llvm.o (Init)
10         *txm_module_preamble_llvm.o (.rodata .rodata.*)
11         KEEP(* (.text .text.*))
12         * (Init)
13         * (.rodata .rodata.*)
14         Image$$ER_RO$$Limit = .;
15     } : LR_1
16

```

## 4.3 Implementing Multiple Embedded Applications in Parallel

GPIO application, VA base address – 0x45000000

```

1 PHDRS {
2     LR_1 PT_LOAD;
3 }
4
5 SECTIONS {
6     ER RO 0x45000000 : {
7         Image$$ER_RO$$Base = .;
8         *txm_module_preamble_llvm.o (.text .text.*)
9         *txm_module_preamble_llvm.o (Init)
10        *txm_module_preamble_llvm.o (.rodata .rodata.*)
11        KEEP(* (.text .text.*))
12        * (Init)
13        * (.rodata .rodata.*)
14        Image$$ER_RO$$Limit = .;
15    } : LR_1

```

**Module Application ID**

Each module application has its own unique module IDs.

The module application ID needs to be adapted in the txm\_module\_preamble\_llvm.s file.

Taking the three above examples this would look like:

Module-ADC:

```
.word      0x1000      @ Module ID (application defined)
```

Module-ATCommand:

```
.word      0x1001      @ Module ID (application defined)
```

Module-GPIO:

```
.word      0x1003      @ Module ID (application defined)
```

**Notes:**

Thales recommends to use virtual addresses in the range between 0x40000000 and 0x5FFFFFFF (as shown above in the examples). Please ensure that the virtual addresses for the individual embedded applications do not overlap.

A maximum of 5 embedded applications running in parallel at the same time are supported.



**THALES DIS AIS Deutschland GmbH**  
Werinherstrasse 81  
81541 Munich  
Germany

**THALES**