General Information Guide

Cross-Domain Development Kit XDK110
Platform for Application Development

Bosch Connected Devices and Solutions

BOSCH
Invented for life

XDK110: General Information Guide	
Document revision	2.0
Document release date	17.08.17
Workbench version	3.0.0
Document number	BCDS-XDK110-GUIDE-GENERAL-INFORMATION
Technical reference code(s)	
Notes	Data in this document is subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance. **Subject to change without notice**

XDK General Information Guide

PLATFORM FOR APPLICATION DEVELOPMENT

Key Features

- All-in-one sensor development kit:
    - No need for component selection and hardware assembly
    - No need for deployment of a real-time operating system
- Drivers for all system components included
- Includes 8 different Micro Electrical-Mechanical Sensors (MEMS)
- WLAN and Bluetooth Low Energy communication technology
- LW-M2M Protocol Stack
- XDK Workbench included as Integrated Development Environment for programming XDK
- Access to the XDK developer community for online technical support, idea discussion, exchange and more
- Small form factor
- High-level Sensor API for the standard user and powerful low-level API for the power user
- Functional extendibility via the included 26 pin extension port
- 2 programmable push buttons
- 1 system LED and 3 programmable LEDs
- Debugger port
- CE, FCC and IC certified

General Description

The XDK110 is a wireless sensor device to enable rapid prototyping of sensor based products and applications for the internet of things (IoT). It thereby allows users a step in-between the first hardware prototype and series production, or simply as the first compact prototype. The device was built in a way that Bosch can easily adapt the product for mass production and the users' unique sensor requirements. The XDK110 creates the opportunity for users to try out more advanced programming on the device itself to realize a sensor which processes data and reports events instead of simply transmitting raw data.

The product XDK110 consists of the XDK110 sensor device, comes with a SW development environment called the XDK Workbench as well as access to the XDK developer community. XDK110 allows the user the ability to experiment with different ideas, with different sensor measuring principles, coupled together with different communication technologies. XDK110 thereby allows the user to better understand their end solution requirements, but also their potential series product before having to invest substantial sums of engineering development activity.

XDK110 is built using a modular hardware and software platform, meaning that mass production of variations of XDK is easy, scalable and cost effective for the user when moving forward with their idea.

Table of Contents

1. Getting Started

1.1 Intended Use

The XDK110 Cross-Domain Development Kit is a prototyping platform for Internet of Things (IoT) use cases. It works with Windows 7 or higher.

1.2 Hardware Overview

1.2.1 Main Components

- MCU: 32-bit microcontroller ARM Cortex M3 EFM32GG390F1024 (Silicon Labs)
- Communication interface
    - Bluetooth Low Energy
    - Low power IEEE 802.11b/g/n WLAN
- Inertial sensors (9DOF)
    - Accelerometer: BMA280 (Bosch Sensortec)
    - Gyroscope: BMG160 (Bosch Sensortec)
    - Magnetometer: BMM150 (Bosch Sensortec)
    - Inertial measurement unit BMI160 (Bosch Sensortec)
- Environmental sensors
    - Combined humidity / temperature / air pressure sensor: BME280 (Bosch Sensortec)
    - Ambient light: MAX44009 (Maxim Integrated)
    - Microphone for noise detection: AKU340 (Akustica)
- Internal Li-Ion rechargeable battery, 560 mAh capacity
- Integrated antennas
- User interface:
    - 3 programmable status LEDs
    - 2 programmable push-buttons
    - Micro SD card1
    - J-Link debug interface2
    - Interface for extension board

1.2.2 Included Deliveries

- Device in its housing, with built-in lithium ion rechargeable battery
- “XDK Gateway” extension board for easy access to additional MCU functionality, incl. connector cable
- Micro USB 2.0 connector cable
- Mounting plate and screws

[1] Micro SD card not included

[2] L-Link debug adapter needed, not included; see JTAG Debug Interface

1.2.3 Technical Specifications

Table 1: XDK110 Specifications

Name	Value
Temperature Range	-20 °C – 60 °C (operating) 0 °C – 45 °C (charging)
Humidity	10 – 90 % r.H., non-condensing
IP Rating	IP 30 (IEC 60529)
Flammability classification	HB (IEC 60695-11-10/-20; CSA C 22.2)
Voltage	5 V DC
Charging Current	500 mA Maximum
Communication (cable)	USB
Wireless LAN	IEEE 802.11 b/g/n
Bluetooth 4.0 low energy	IEEE 802.15.1

Please note that full sensor performance might be limited to a more narrow temperature range, and sensor performance might be decreased when operating outside this corner cases. Refer to the sensor datasheets for details.

1.2.4 XDK Device

Picture 1: Bosch XDK Device

1.2.5 User Interface: LEDs & Buttons

The XDK has four LEDs to signal the operation mode it is currently in, three of which can be configured by the user when in application mode. The fourth (green) LED is used to display the state of the charger chip and cannot be accessed by software.

The following status can be displayed by the green LED:

Table 2: Meaning of the green LED

Green LED	Hardware Status
On	Battery is charging or charging suspended by thermal loop
Off	Charging done/Recharging after termination/IC disabled or No valid Input Power/Battery absent
Blinking	Safety timers expired

The following status can be displayed by the red, orange and yellow LED:

Table 3: Meaning of the programmable LEDs

Red LED	Orange LED	Yellow LED	Bootloader Mode	Application Mode
On	Off	Off	Active; no USB connection or driver not loaded	No function
Off	Off	Off	Inactive	
Blinking	On/Off	Off	Active; program invalid, cannot boot	
On	On	Off	Active, USB connection detected	XDK stopped, assertion occurred
On	Blinking	Off	Active; Data is transmitted	No function
On	On	On	Inactive	XDK stopped, stack overflow occurred

The XDK110 uses the LED to show the following states:
After the XDK110 has been switched on, a running light demo will start.

1.3 Getting Started

1.3.1 Software Download and installation

The software for the XDK is available online. You can download the software from *http://www.xdk.bosch-connectivity.com.* Go to the download section to get the latest software package and demos.
Download the software package "XDK Workbench" from the website http://www.xdk.bosch-connectivity.com and start the installer. The software package contains all necessary components. Program examples, demos and a toolbox are included.

1.3.2 Connecting XDK110

Connect the USB cable (included in delivery) to the USB connector of your PC and the Mini USB connector of the XDK.

Note: Functional Limitations:

The device can be impaired or damaged if the power source is inadequate.
- Do not use cables >3 m.
- Use only certified USB power adapters/connectors.

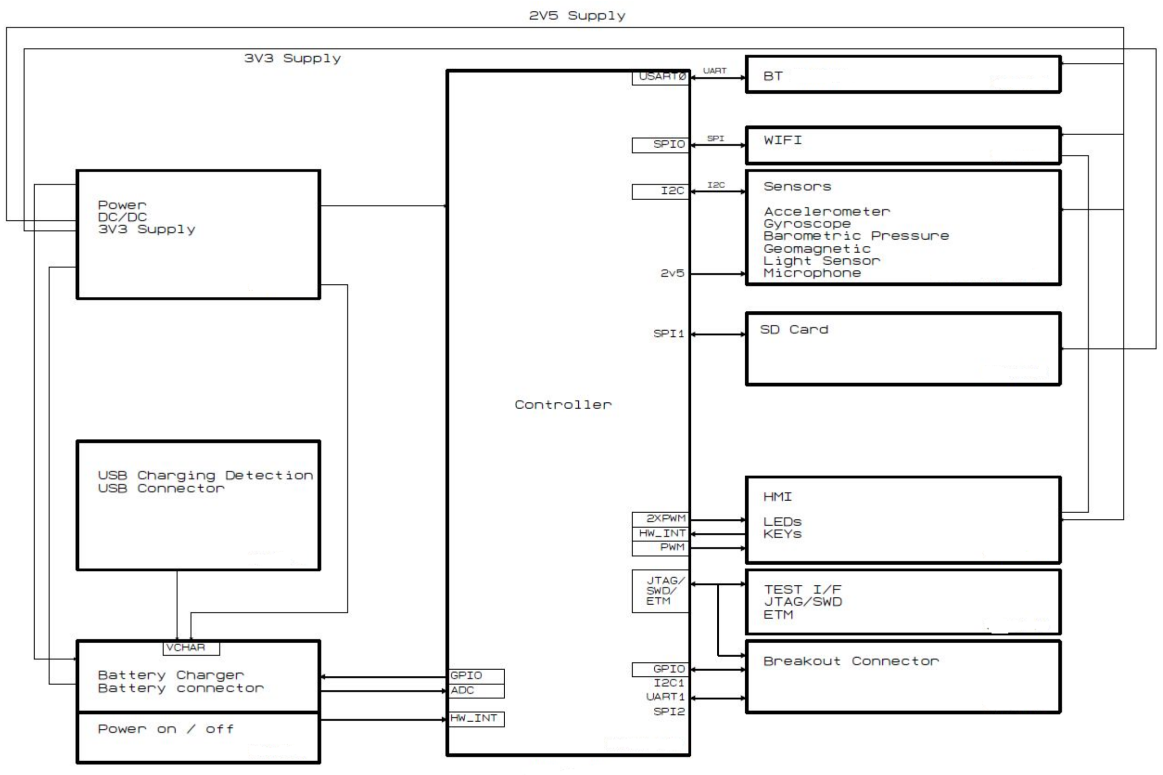
---

2. Functional Description

2.1 Block Diagram

The following Picture 2 shows a simplified block diagram of the XDK110:

Picture 2: Block diagram of XDK110



2.2 Power Management

For the easy application in portable applications, the XDK is equipped with a 560mAh Li-Ion rechargeable battery. While connected to a power source via the Micro USB connector, the battery will automatically be charged. Charging is indicated by the green LED. The device can be charged independent of the position of the power switch. The maximum charging current depends on the charger used.

Table 4: Charger Type and Current

Charger Type	Maximum Current
Standard downstream port (SDP): Typical form found in desktop and laptop computers	100 mA
Charging downstream port (CDP): Higher current USB port for PCs, laptops, and other hardware (indicated by a lightning symbol)	500 mA
Dedicated charging port (DCP): Power sources like wall warts and auto adapters; charging can occur with no digital communication at all.	500 mA or 100 mA depending on charger

Additional details on these port types are described in the USB *Battery Charging Specification, Rev 1.1, 4/15/*

2.3 Sensors

XDK110 is equipped with a multitude of sensors on board, which makes it an ideal tool for application development for the Internet of Things.

This documentation refers to the datasheets of the sensors for a deeper understanding of the components. It is, however, vital to understand that sensor performance will be limited by the application hardware. As an example, the temperature range of most sensors will be -40 to +85 °C in the datasheet, while the XDK is only specified for a usage between -20 and +60 °C due to physical and safety limits of the battery. Also, sensor accuracy is generally higher in the sensor datasheet than it can be expected from the target device, since the assembly process and mechanical stress of the board influence the performance of any sensor. The absolute maximum ratings of the key parameters are stated for each sensor. For more details, please refer to the respective sensor datasheet.

2.3.1 BMM150: Three-Axis Geomagnetic Sensor

The BMM150 is a standalone geomagnetic sensor for consumer market applications. It allows measurements of the magnetic field in three perpendicular axes. Based on Bosch's proprietary FlipCore technology, performance and features of BMM150 are carefully tuned and perfectly match the demanding requirements of all three-axis mobile applications such as electronic compass, navigation or augmented reality.

Table 5: BMM150 Key Features

Specification	Value
Resolution	0.3 µT
Zero-B Offset	±50 µT
Non-Linearity	<1 % FS
Magnetic Range (typical)	±1300 µT (X;Y-Axis); ±2500 µT (Z-Axis)
Average Current Consumption	170 µA (Low Power Preset); 500 µA (Normal Mode)
Interrupts	New Data; Magnetic Threshold High/Low

For more information please refer to the datasheet:
http://ae-bst.resource.bosch.com/media/products/dokumente/bmm150/BST-BMM150-DS001-01.pdf

2.3.2 BMA280: Three-Axis Acceleration Sensor

The BMA280 is an advanced, ultra-small, triaxial, low-g acceleration sensor with digital interfaces, aiming for low-power consumer electronics applications. Featuring 14-bit digital resolution, the BMA280 allows very low-noise measurement of accelerations in three perpendicular axes and thus senses tilt, motion, shock and vibration in cellular phones, handhelds, computer peripherals, man-machine interfaces, virtual reality features and game controllers.

Table 6: BMA280 Key Features

Specification	Value
Digital Resolution	14 bit
Resolution (in ±2 g range)	0.244 mg
Measurement Ranges (programmable)	±2 g, ±4 g, ±8 g, ±16 g
Sensitivity (calibrated)	±2 g: 4096 LSB/g ±4 g: 2048 LSB/g ±8 g: 1024 LSB/g ±16 g: 512 LSB/g
Zero-g Offset (typical, over life-time)	±50 mg
Noise Density (typical)	120 µg/√Hz
Bandwidths (programmable)	500 Hz ... 8 Hz
Current Consumption (full operation)	130 µA (@2 kHz data rate)

For more information please refer to the datasheet:
http://www.bosch-sensortec.com/en/homepage/products_3/3_axis_sensors/acceleration_sensors/bma280/bma280

2.3.3 BMG160: Three-Axis Angular Rate Sensor

The BMG160 is an ultra-small, digital three-axis angular rate sensor with a measurement range up to 2000 °/s and a digital resolution of 16 bit for consumer electronics applications. The BMG160 allows low-noise measurement of angular rates in three perpendicular axes and is designed for use in cellular phones, handhelds, computer peripherals, man-machine interfaces, virtual reality features, remote and game controllers.

Table 7: BMG160 Key Features

Specification	Value
Digital Resolution	16 bit
Measurement Ranges (programmable)	± 125 °/s, ± 250 °/s, ± 500 °/s, ± 1000 °/s, ± 2000 °/s
Sensitivity (calibrated)	± 125 °/s: 262.4 LSB/ °/s ± 250 °/s: 131.2 LSB/ °/s ± 500 °/s: 65.5 LSB/ °/s ± 1000 °/s: 32.8 LSB/ °/s ± 2000 °/s: 16.4 LSB/ °/s
Zero-Rate Offset (typical)	± 1 °/s
Zero-Rate Offset over Temperature	0.015 °/s/K
Noise Density (typical)	0.014 °/s/√Hz
Low-Pass Filter Bandwidths (programmable)	230, 116, 64, 47, 32, 23, 12 Hz
Date Rates (programmable)	2000, 1000, 400, 200, 100 Hz
Current Consumption (full operation)	5.0 mA
Current Consumption (fast power-up)	2.5 mA

For more information please refer to the datasheet:
http://ae-bst.resource.bosch.com/media/products/dokumente/bmg160/BST-BMG160-DS000-09.pdf

2.3.4 BMI160: Inertial Measurement Unit

The BMI160 is a small, low-power, low-noise 16-bit inertial measurement unit designed for use in mobile applications like augmented reality or indoor navigation which require highly accurate, real-time sensor data. In full operation mode, with both the accelerometer and the gyroscope enabled, the current consumption is typically 950 µA, enabling always-on applications in battery driven devices.

Table 8: BMI160 Key Features

Specification	Value
Digital Resolution	Accelerometer (A): 16 bit Gyroscope (G): 16 bit
Measurement Ranges (programmable)	(A): ± 2 g, ± 4 g, ± 8 g, ± 16 g (G): ± 125 °/s, ± 250 °/s, ± 500 °/s, ± 1000 °/s, ± 2000 °/s
Sensitivity (calibrated)	(A): ±2g: 16384 LSB/g ±4g: 8192 LSB/g ±8g: 4096 LSB/g ±16g: 2048 LSB/g (G): ±125 °/s: 262.4 LSB/ °/s ±250°/s: 131.2 LSB/ °/s ±500 °/s: 65.6 LSB/ °/s ±1000 °/s: 32.8 LSB/ °/s ±2000 °/s: 16.4 LSB/ °/s
Zero-Point Offset	(A): ±40mg (G): ± 10 °/s
Noise Density (typical)	(A): 180 µg/√Hz (G): 0.008 °/s/√Hz
Bandwidths (programmable)	1600 Hz … 25/32 Hz

For more information please refer to the datasheet:
http://www.bosch-sensortec.com/en/homepage/products_3/6_axis_sensors_2/inertial_measurement_unit_1/bmi160/bmi160_1

Note: Why two gyroscopes?

As you will have noticed by now, the XDK has two MEMS gyroscopes on board: A stand-alone BMG160 and another one combined with an accelerometer in the inertial measurement unit BMI160.
The sensor in the BMI160 is an "open-loop" gyroscope, which results in very low noise and low power consumption (~1 mA). The BMG160 works in "closed loop", which results in an excellent zero-Ω offset change over temperature (TCO), which can be two orders of magnitude below the open-loop version. As a result, the power consumption is higher (~5 mA).

We encourage you to try out both sensors, and decide which is best for your application.

2.3.5 BME280: Environmental Sensor

The BME280 is an integrated environmental sensor developed specifically for mobile applications where size and low power consumption are key design constraints. The unit combines individual high-linearity, high-accuracy sensors for pressure, humidity and temperature, designed for low current consumption (3.6 µA @ 1 Hz), and long-term stability. The humidity sensor features an extremely fast response time which supports performance requirements for emerging applications such as context awareness, and high accuracy over a wide temperature range. The pressure sensor is an absolute barometric pressure sensor which features exceptionally high accuracy and resolution at very low noise. The integrated temperature sensor has been optimized for very low noise and high resolution. It is primarily used for temperature compensation of the pressure and humidity sensors, and can also be used for estimating ambient temperature.

Table 9: Sensor Measurement Ranges and Accuracies

Parameter	Condition	Min		Max	Unit
Operating Temperature Range	Operational	-20	25	+60	°C
	Full Accuracy	0		+60	°C
Operating Pressure Range		300		1100	hPa
Absolute Accuracy Pressure			±10		hPa
Absolute Accuracy Temperature			±2		K
Absolute Accuracy Humidity			±10		%RH
Average Current Consumption (1Hz Data Refresh Rate)	H, T		1.8		µA
	P, T		2.8		µA
	H, P, T		3.6		µA

Note: Measurement errors due to self-heating

As any electronic component, the sensor is subject to self-heating, which can be from the sensor itself, as well as from other components used at the time of measurement. Unfortunately, this self-heating strongly depends on the usage, and as the XDK can be freely programmed, the self-heating depends on so many factors that we cannot provide a universal compensation algorithm. The sensor can be influenced to some extent by the sampling rate and by how fast a reading is taken after switching it on (see sensor datasheet for details).
For a user-defined program, we suggest monitoring the outside temperature with a reference thermometer and once a stable offset value is observed, use this to compensate the measurements. As an indication, a self-heating of 3-4 K under a normal load profile using sensors and Bluetooth can be expected.
The accuracy of the humidity measurement is strongly related the temperature, as relative humidity is calculated using the temperature reading. Hence, a higher temperature leads to a lower relative humidity. Once the temperature compensation is done, the relative humidity can be calculated using the temperature difference.

For more information please refer to the datasheet:
http://www.bosch-sensortec.com/en/homepage/products_3/environmental_sensors_1/bme280/bme280_1

2.3.6 AKU340: Acoustic Noise Sensor

The XDK is equipped with an AKU340 analog microphone. Due to limitations of the MCU, the detection of sound patterns (e.g. voice) is not possible. However, the AKU340 can be used as a sensor to detect ambient noise.

Unfortunately, the current version of the XDK workbench does not contain an implementation of the noise sensor into the API. However, it is possible to access the device via the following pins:

- PD9: AKU340_VDD
- PD4: AKU340_OUT (can be sampled using the analog/digital converter)

For more information please refer to the datasheet:
http://www.akustica.com/Files/Admin/PDFs/Datasheets/DS26%2D1%2E03%20AKU340%20Datasheet%2Epdf

2.3.7 MAX44009: Ambient Light Sensor

The MAX44009 ambient light sensor is ideal for a number of portable applications such as smartphones, notebooks, and industrial sensors. At less than 1µA operating current, it is the lowest power ambient light sensor in the industry and features an ultra-wide 22-bit dynamic range from 0.045 lux to 188,000 lux.

For more information please refer to the datasheet:
http://datasheets.maximintegrated.com/en/ds/MAX44009.pdf

2.4 Radios

2.4.1 WLAN

XDK is equipped with a state-of-the-art low-power WLAN transceiver.

Key parameters:

Protocol: 802.11b/g/n – 2.4GHz
TX Power:
- 17 dBm at 1 DSSS
- 17.25 dBm at 11 CCK
- 13.5 dBm at 54 OFDM

RX Sensitivity:
- –94.7 dBm at 1 DSSS
- –87 dBm at 11 CCK
- –73 dBm at 54 OFDM

Security: WPA2 (Personal and Enterprise Security)

2.4.2 Low-Energy Bluetooth

XDK uses a Bluetooth low energy controller compliant to Bluetooth specification v4.0.

Key parameters:

Protocol: Bluetooth v4.0 – 2.4GHz
TX Power: -18dBm to +3dBm (programmable)
RX Sensitivity: -80 dBm
Security: AES-128

2.5 SD Card

XDK110 is equipped with a Micro SD card reader that allows the user to store e.g. sensor readout data or log other events. Due to the processor architecture, SD cards over 32 GB (SD V3 standard) are not supported. Additionally as filesystem are only FAT and FAT32 supported. Please make sure that the card used supports data transmission via SPI.

2.6 Microcontroller

The XDK110 uses a Cortex M3 MCU of the Silicon Labs' EFM32™ 32-bit microcontroller family. It's extremely energy efficient and is especially suited for use in low-power and energy sensitive applications. This allows you to design your applications for efficiency and long battery lifetime from the very beginning. Details can be found in the MCU datasheet and MCU Reference Manual.

Note: Limitations of energy modes

To make the USB connection fully usable for development purposes, the Energy Modes EM3 and EM4 of the Gecko MCU are not available on XDK110.

---

3. XDK Software Overview – The XDK Workbench

The software for the XDK is available online. You can download the software from http://xdk.bosch-connectivity.com. Go to the download section to get the latest software package and demos. The software package contains all necessary components. Program examples, demos, drivers and a toolbox are included. Download the software package “XDK Workbench” and start the installer. The installer will lead you through the installation procedure. After a successful installation, Windows will install the drivers for the XDK 110 automatically when the device is detected.
After installation the Help can be reached under Help\Help Contents\XDK API-Documentation.
The XDK software is based on FreeRTOS and completely operated via USB. That means flashing, charging and sending data to the PC using “Printf”. A logging framework allows sneaking into existing modules.
In general always a complete image will be transferred onto target. The Workbench compiles a complete application; afterwards the image will be flashed onto the XDK using USB.
Except for the bootloader XDK provides no “always there” software.

3.1 Power Consumption and Energy Management

The XDK reactively supplies power to different components. That is, only when the application activates a certain peripheral the power is switched on.
The XDK API makes sure that when you call the initialize function of a software module, all relevant hardware components and modules are powered up. In turn this means that as long as you do not initialize a component (e.g. a sensor), this component does not consume power. However, some peripherals are always powered up when XDK boots. The following components are considered vital to XDK’s operation and hence are always initialized:

- GPIO Pins (including potential pull-up resistor)
- I2C Module
- USB Module

Additional peripherals (such as the SPI module) are only initialized when they are required (e.g. when initializing WLan or the SD card). The default initialization of the XDK SDK can be replaced, as outlined in 3.6.2.3 ff.

3.1.1 Battery Status

To determine the battery voltage of XDK, the battery is connected via a management circuit to pin PD7 of the MCU. This pin can be configured with the ADC software module to allow sampling the battery voltage. Note that the values measured are not the actual battery voltage, since this is too high for the MCU in to be connected directly. A voltage divider is used to bring down the voltage to ~50% of the current voltage supplied to the system via the power management circuit.
The battery charger of XDK gives information about the current charging mode via the green LED of XDK. However, this pin is not connected to the MCU. Determining the current charging mode in the software running on XDK is therefore not possible. The green LED only shows the charging status and cannot be controlled by software.

Table 10: Battery Status

Green LED	Charging Mode
Continuous	Charging/Charging suspended by thermal loop
Flashing	Safety timers expired
Off	Charging done/Recharging after termination/ IC disabled or no valid input power/Battery absent

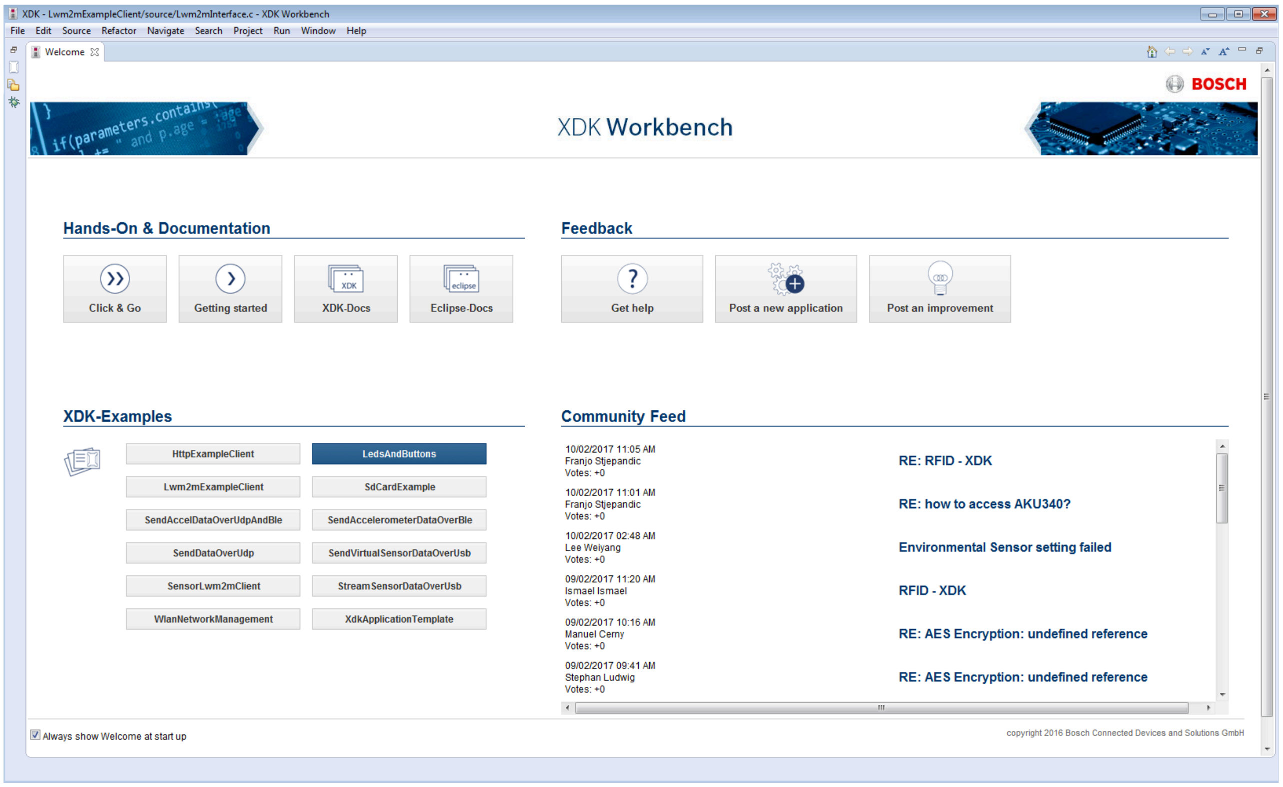
3.1.2 Sleep Management

XDK SDK currently does not support putting the MCU of XDK to sleep. While there is a software module called RTC, this module is neither used on XDK SDK nor has it been tested on XDK. Therefore, using it is at your own risk and may potentially be very frustrating.

3.2 XDK Workbench - Welcome Screen

Start the XDK Workbench by clicking on the XDK icon on your desktop. The default path for the application is C:\XDK-Workbench\XDK-Workbench.exe.
During startup a splash screen appears. The welcome screen will be shown afterwards.

Picture 3: XDK Workbench Welcome Screen



10/02/2017 11:05 AM
Franjo Stjepandic
Votes: +0
RE: RFID - XDK

10/02/2017 11:01 AM
Franjo Stjepandic
Votes: +0
RE: how to access AKU340?

10/02/2017 02:48 AM
Lee Weiyang
Votes: +0
Environmental Sensor setting failed

09/02/2017 11:20 AM
Ismael Ismael
Votes: +0
RFID - XDK

09/02/2017 10:16 AM
Manuel Cerny
Votes: +0
RE: AES Encryption: undefined reference

09/02/2017 09:41 AM
Stephan Ludwig
Votes: +0
RE: AES Encryption: undefined reference

The welcome screen is sectioned in four parts:
Hands-On & Documentation (upper left), Feedback (upper right), XDK-Examples (lower left) and the Community feed (lower right; currently not available).

3.2.1 Hands On & Documentation

This section contains four buttons.

Click&Go:	Access to the empty workspace
Getting Started:	The shortcut to the XDK Forum
XDK Docs:	This button will open the XDK help
Eclipse Docs:	This button will open the Eclipse help

3.2.2 Feedback

Clicking the respective button will take you the matching XDK community forum section to get help, to post a new application or to post an improvement.
Inside the community you will find hints, tips and tricks and you can get in touch with other developers.

3.2.3 XDK Examples - Basics

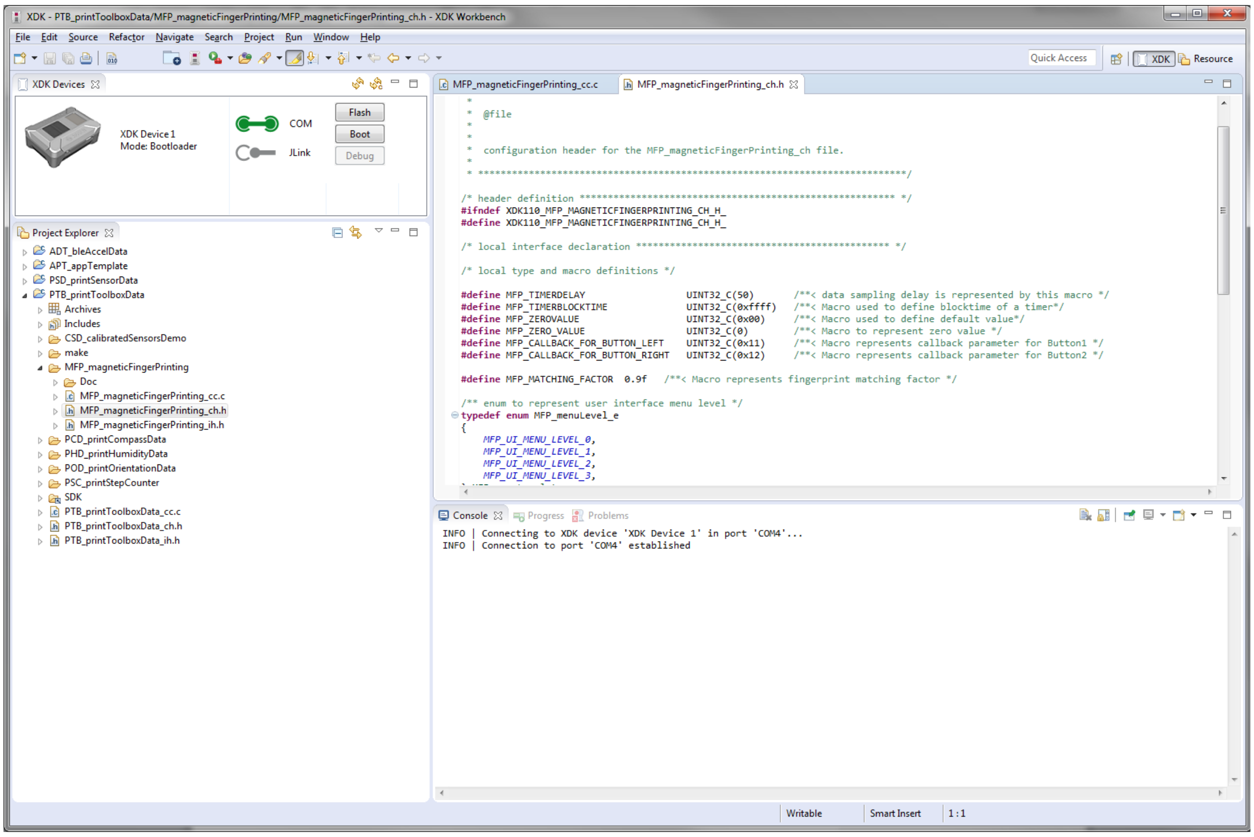
The XDK comes with a variety of application examples that can be imported via the welcome screen of the XDK Workbench. Choose one example and click on it. The chosen example will be imported to the workspace. The examples show how to access a particular interface. (eg. Wi-Fi, SD card, …). More information can be found in the Workbench Examples guide at xdk.io/guides.

3.2.4 Community Feed

Here you will find the latest community updates.

3.3 XDK Workbench – Workspace (XDK View)

Picture 4: XDK Workspace



XDK - PTB_printToolboxData/MFP_magneticFingerPrinting/MFP_magneticFingerPrinting_ch.h - XDK Workbench
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
XDK
Resource
XDK Devices
XDK Device 1
Mode: Bootloader
COM
JLink
Flash
Boot
Debug
Project Explorer
ADT_bleAccelData
APT_appTemplate
PSD_printSensorData
PTB_printToolboxData
Archives
Includes
CSD_calibratedSensorsDemo
make
MFP_magneticFingerPrinting
Doc
MFP_magneticFingerPrinting_cc.c
MFP_magneticFingerPrinting_ch.h
MFP_magneticFingerPrinting_ih.h
PCD_printCompassData
PHD_printHumidityData
POD_printOrientationData
PSC_printStepCounter
SDK
PTB_printToolboxData_cc.c
PTB_printToolboxData_ch.h
PTB_printToolboxData_ih.h
MFP_magneticFingerPrinting_cc.c
MFP_magneticFingerPrinting_ch.h
Console
Progress
Problems
Writable
Smart Insert
1 : 1

```c
 *
 *  @file
 *
 *
 *  configuration header for the MFP_magneticFingerPrinting_ch file.
 *
 * ****************************************************************************/

/* header definition ******************************************************** */
#ifndef XDK110_MFP_MAGNETICFINGERPRINTING_CH_H_
#define XDK110_MFP_MAGNETICFINGERPRINTING_CH_H_

/* local interface declaration ********************************************* */

/* local type and macro definitions */

#define MFP_TIMERDELAY                  UINT32_C(50)       /**< data sampling delay is represented by this macro */
#define MFP_TIMERBLOCKTIME              UINT32_C(0xffff)   /**< Macro used to define blocktime of a timer*/
#define MFP_ZEROVALUE                   UINT32_C(0x00)     /**< Macro used to define default value*/
#define MFP_ZERO_VALUE                  UINT32_C(0)        /**< Macro to represent zero value */
#define MFP_CALLBACK_FOR_BUTTON_LEFT    UINT32_C(0x11)     /**< Macro represents callback parameter for Button1 */
#define MFP_CALLBACK_FOR_BUTTON_RIGHT   UINT32_C(0x12)     /**< Macro represents callback parameter for Button2 */

#define MFP_MATCHING_FACTOR  0.9f   /**< Macro represents fingerprint matching factor */

/** enum to represent user interface menu level */
typedef enum MFP_menuLevel_e
{
    MFP_UI_MENU_LEVEL_0,
    MFP_UI_MENU_LEVEL_1,
    MFP_UI_MENU_LEVEL_2,
    MFP_UI_MENU_LEVEL_3,
```

```
INFO | Connecting to XDK device 'XDK Device 1' in port 'COM4'...
INFO | Connection to port 'COM4' established
```

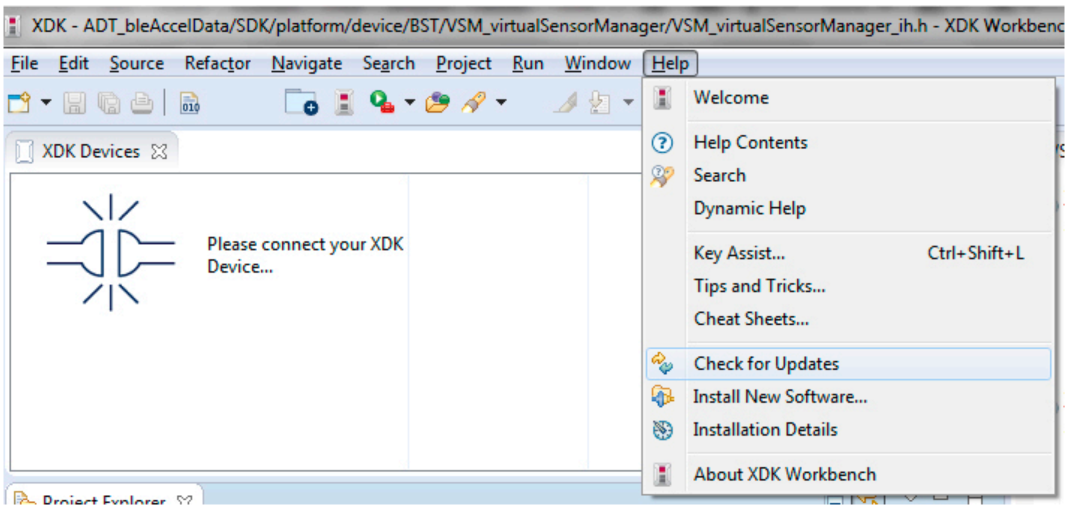
Picture 4 shows the default view of the XDK Workbench. A detailed description about how to get familiar with the XDK Workbench is provided in the Workbench First Steps Guide at xdk.io/guides. This includes a major overview of the features of the XDK Workbench and how to get started with an own application on the XDK.

3.3.1 Updating XDK Workbench

XDK Workbench has a built-in update functionality that ensures that you are always using the latest version. XDK Workbench automatically checks for updates on a regular base.
To trigger the update process manually, perform the following steps:

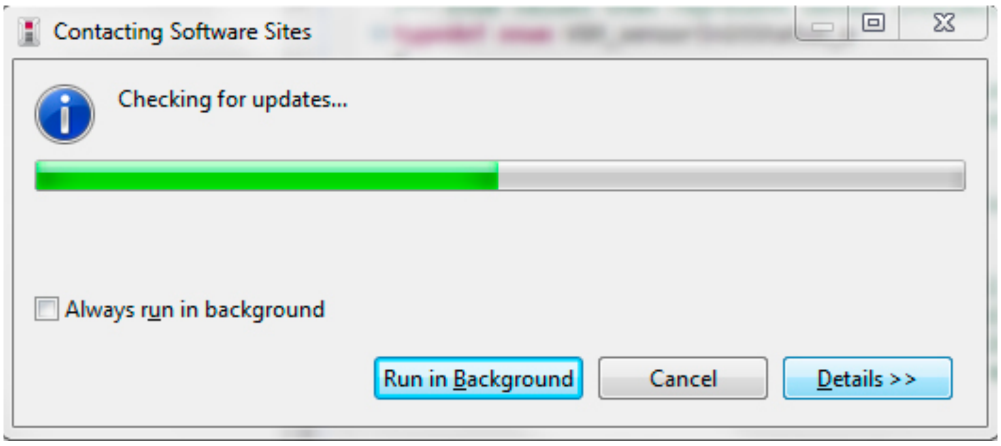
1. In XDK-Workbench, click on “Help” -> “Check for Updates”

Picture 5: Check for Updates


XDK - ADT_bleAccelData/SDK/platform/device/BST/VSM_virtualSensorManager/VSM_virtualSensorManager_ih.h - XDK Workbenc
File Edit Source Refactor Navigate Search Project Run Window Help
XDK Devices
Please connect your XDK Device...
Welcome
Help Contents
Search
Dynamic Help
Key Assist... Ctrl+Shift+L
Tips and Tricks...
Cheat Sheets...
Check for Updates
Install New Software...
Installation Details
About XDK Workbench
Project Explorer

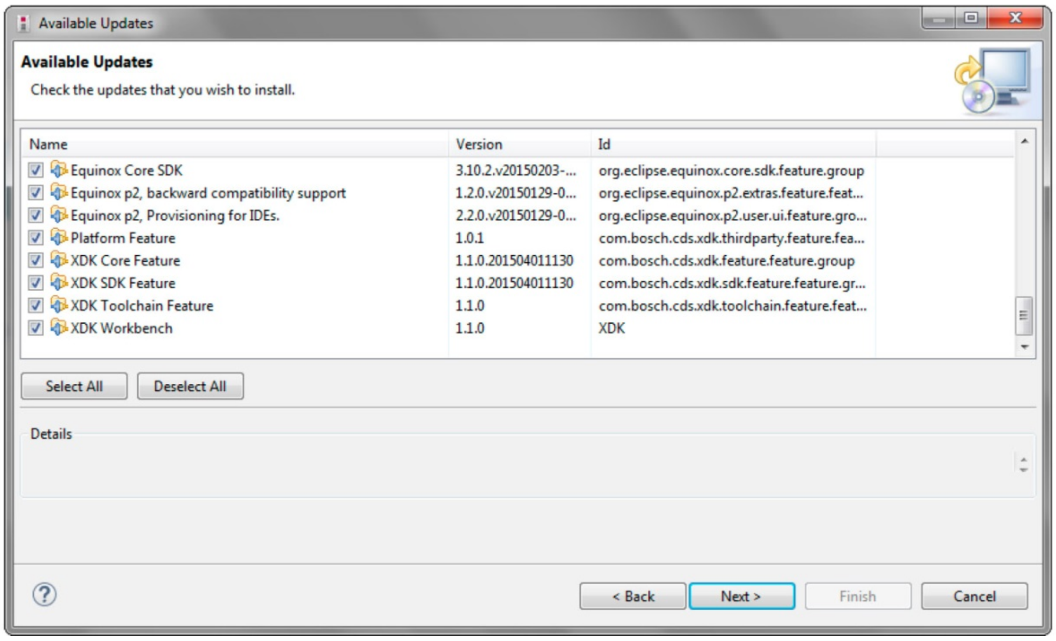
2. XDK Workbench will contact the BCDS XDK Software Update Site and look for new versions of the XDK Workbench.

Picture 6: Check for Updates in Progress



3. Select all offered updates and click on “Next”

Picture 7: Available Updates Window



Available Updates

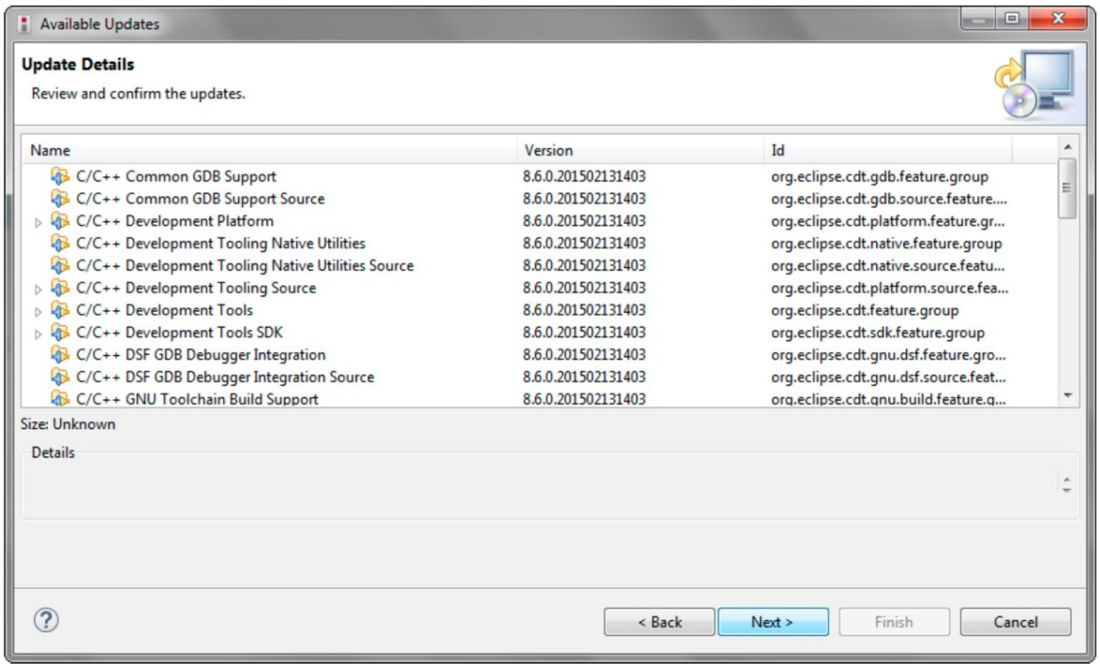
Available Updates

Check the updates that you wish to install.

Name	Version	Id
☑ Equinox Core SDK	3.10.2.v20150203-...	org.eclipse.equinox.core.sdk.feature.group
☑ Equinox p2, backward compatibility support	1.2.0.v20150129-0...	org.eclipse.equinox.p2.extras.feature.feat...
☑ Equinox p2, Provisioning for IDEs.	2.2.0.v20150129-0...	org.eclipse.equinox.p2.user.ui.feature.gro...
☑ Platform Feature	1.0.1	com.bosch.cds.xdk.thirdparty.feature.fea...
☑ XDK Core Feature	1.1.0.201504011130	com.bosch.cds.xdk.feature.feature.group
☑ XDK SDK Feature	1.1.0.201504011130	com.bosch.cds.xdk.sdk.feature.feature.gr...
☑ XDK Toolchain Feature	1.1.0	com.bosch.cds.xdk.toolchain.feature.feat...
☑ XDK Workbench	1.1.0	XDK

4. XDK Workbench will present you a list of components that will be updated. Click “Next” to continue.

Picture 8: Update Details



Available Updates

Update Details

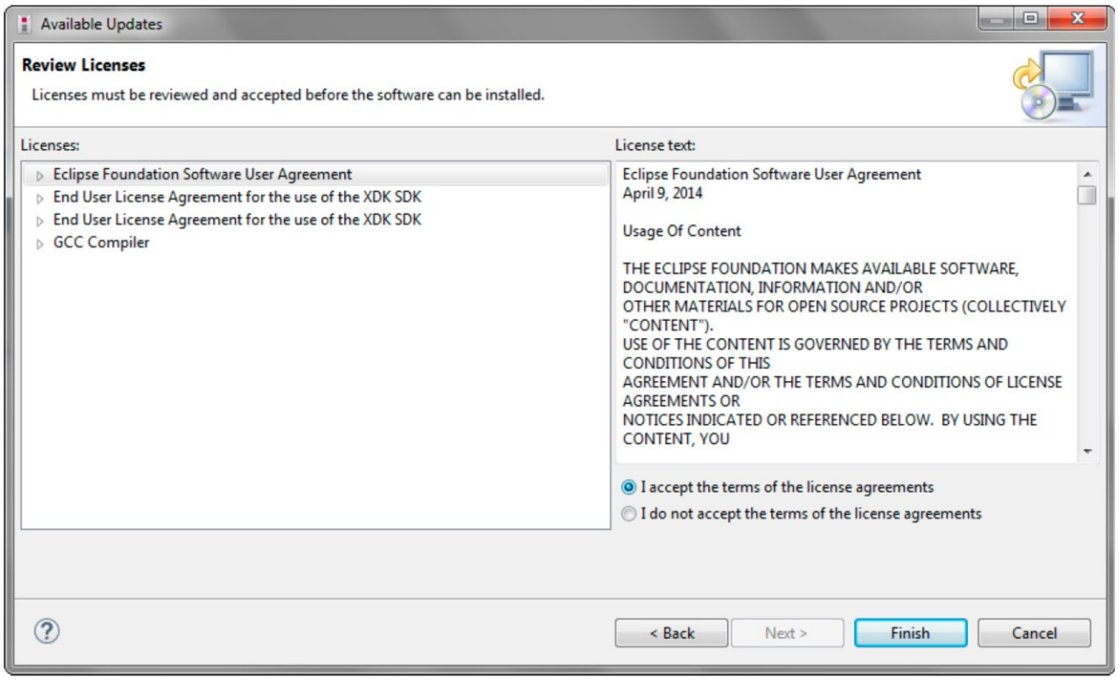
Review and confirm the updates.

Name	Version	Id
C/C++ Common GDB Support	8.6.0.201502131403	org.eclipse.cdt.gdb.feature.group
C/C++ Common GDB Support Source	8.6.0.201502131403	org.eclipse.cdt.gdb.source.feature....
C/C++ Development Platform	8.6.0.201502131403	org.eclipse.cdt.platform.feature.gr...
C/C++ Development Tooling Native Utilities	8.6.0.201502131403	org.eclipse.cdt.native.feature.group
C/C++ Development Tooling Native Utilities Source	8.6.0.201502131403	org.eclipse.cdt.native.source.featu...
C/C++ Development Tooling Source	8.6.0.201502131403	org.eclipse.cdt.platform.source.fea...
C/C++ Development Tools	8.6.0.201502131403	org.eclipse.cdt.feature.group
C/C++ Development Tools SDK	8.6.0.201502131403	org.eclipse.cdt.sdk.feature.group
C/C++ DSF GDB Debugger Integration	8.6.0.201502131403	org.eclipse.cdt.gnu.dsf.feature.gro...
C/C++ DSF GDB Debugger Integration Source	8.6.0.201502131403	org.eclipse.cdt.gnu.dsf.source.feat...
C/C++ GNU Toolchain Build Support	8.6.0.201502131403	org.eclipse.cdt.gnu.build.feature.g...

Size: Unknown

5. In case an update will contain new licenses, please check them thoroughly. Decide if you want to accept the licenses (all at once) and click on “Finish”.

Picture 9: Licence Review Window


Available Updates

Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

Licenses:
- Eclipse Foundation Software User Agreement
- End User License Agreement for the use of the XDK SDK
- End User License Agreement for the use of the XDK SDK
- GCC Compiler

License text:

Eclipse Foundation Software User Agreement
April 9, 2014

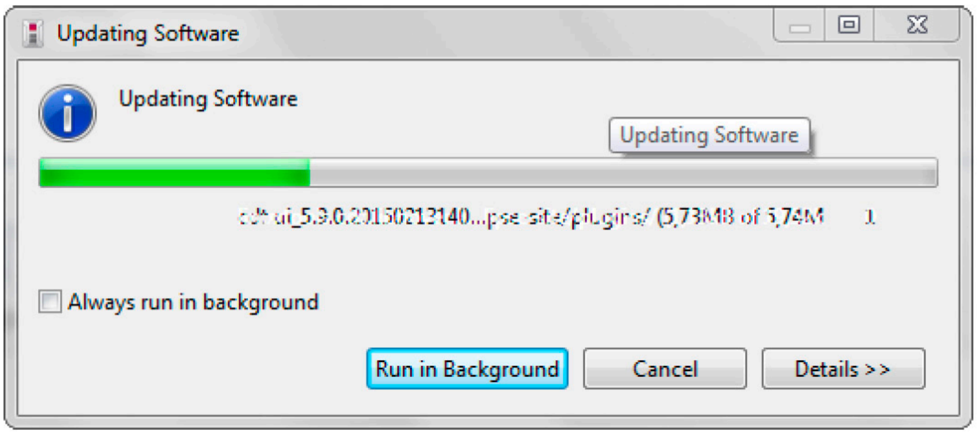
Usage Of Content

THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU

[✓ I accept the terms of the license agreements]
I do not accept the terms of the license agreements

6. XDK Workbench will now download the updated components. This may take a couple of minutes, depending on your internet connection bandwidth.

Picture 10: Update in Progress


Updating Software

Updating Software

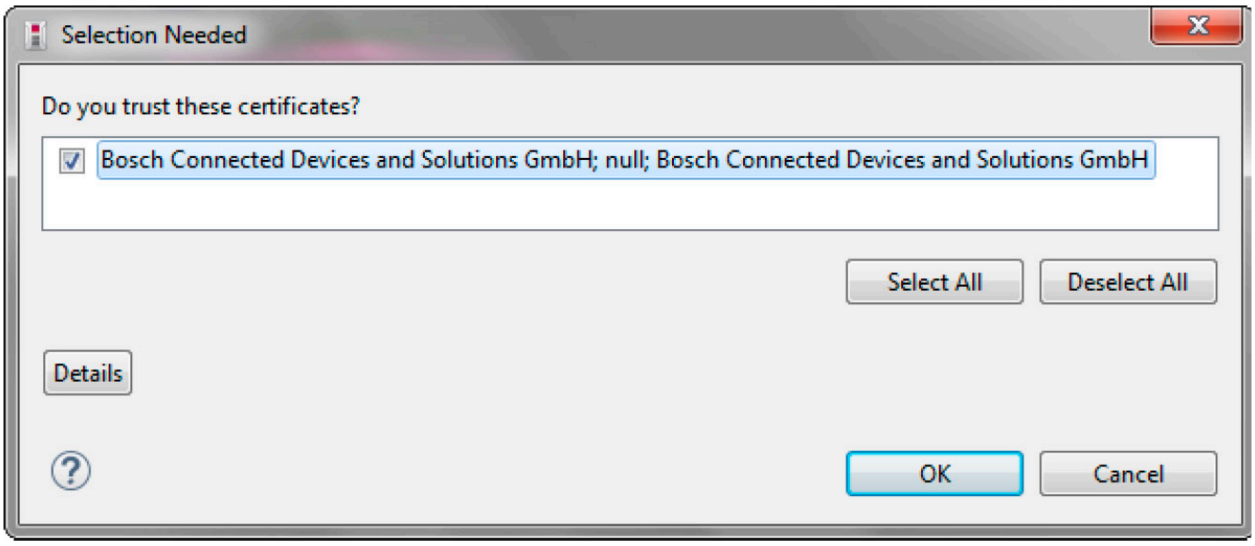
Updating Software

```
[illegible] ui_5.3.0.20150213140...pse-site/plugins/ (5,73MB of 5,74M[illegible]
```

Always run in background

7. Before installing the updates, XDK Workbench requires you to verify the code signature certificate of BCDS. Please check it and press “OK”.

Picture 11: Certificate Verification


Selection Needed

Do you trust these certificates?

[✓] Bosch Connected Devices and Solutions GmbH; null; Bosch Connected Devices and Solutions GmbH

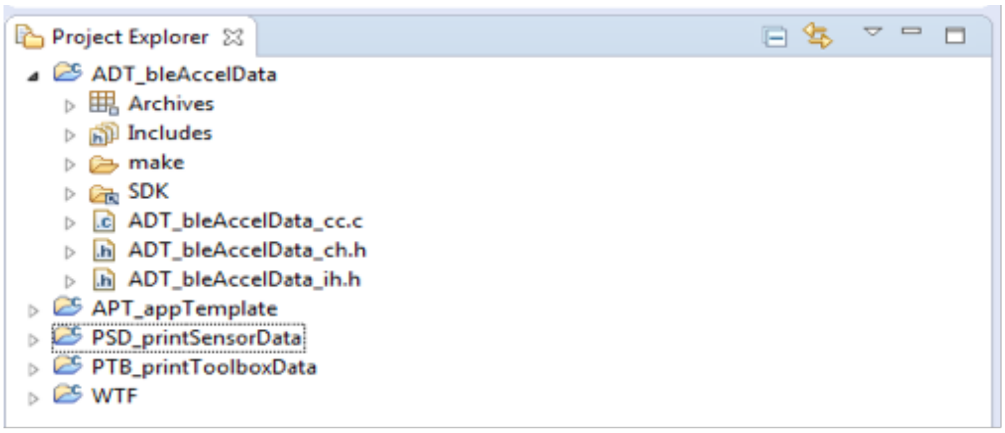
8. After installing the updates, XDK Workbench must be restarted for the updates to take effect.

Debug

For debugging a J-Link adapter is required (see chapter 4.3 for details).
Choose a previously compiled project. The XDK Workbench configures the debug configuration automatically. The binary will be downloaded to the XDK and the debugger will be started. The XDK Workbench proposes to switch to debug perspective.

3.3.2 Project Explorer

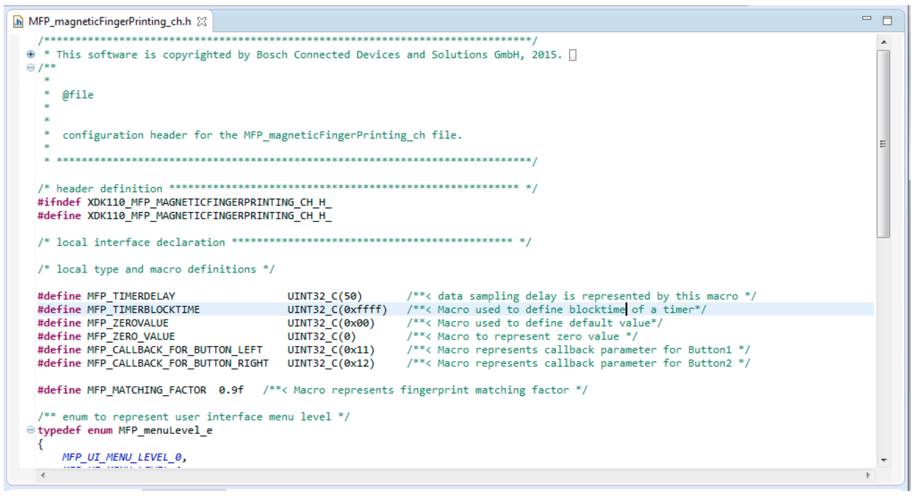
Picture 12: Project Explorer Tile



The Project Explorer helps you to manage your projects. You can easily browse through your project folder and choose a file. Double-clicking will make an editable file appear inside the editor tile. More functions are available by using the context menu.

3.3.3 Editor

Picture 12: Editor Tile

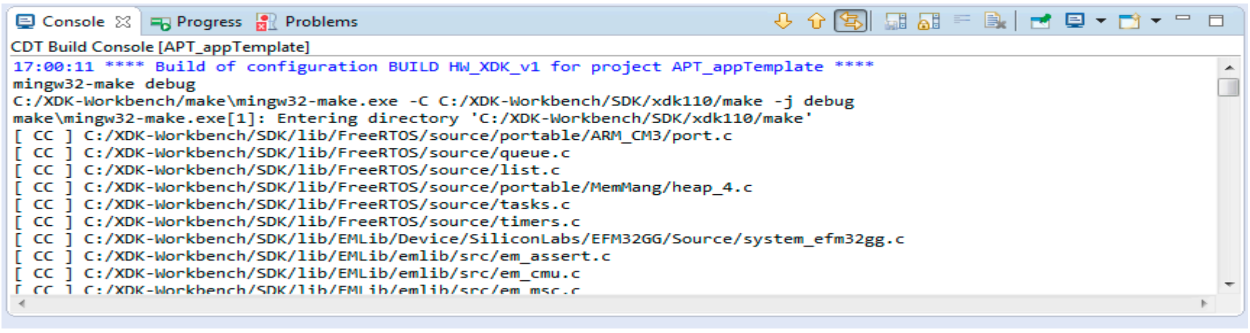


```
/********************************************************************************/
 * This software is copyrighted by Bosch Connected Devices and Solutions GmbH, 2015.
/**
 *
 *  @file
 *
 *
 *  configuration header for the MFP_magneticFingerPrinting_ch file.
 *
 * ******************************************************************************/

/* header definition ********************************************************* */
#ifndef XDK110_MFP_MAGNETICFINGERPRINTING_CH_H_
#define XDK110_MFP_MAGNETICFINGERPRINTING_CH_H_

/* local interface declaration ********************************************* */

/* local type and macro definitions */

#define MFP_TIMERDELAY                UINT32_C(50)       /**< data sampling delay is represented by this macro */
#define MFP_TIMERBLOCKTIME            UINT32_C(0xffff)   /**< Macro used to define blocktime of a timer*/
#define MFP_ZEROVALUE                 UINT32_C(0x00)     /**< Macro used to define default value*/
#define MFP_ZERO_VALUE                UINT32_C(0)        /**< Macro to represent zero value */
#define MFP_CALLBACK_FOR_BUTTON_LEFT  UINT32_C(0x11)     /**< Macro represents callback parameter for Button1 */
#define MFP_CALLBACK_FOR_BUTTON_RIGHT UINT32_C(0x12)     /**< Macro represents callback parameter for Button2 */

#define MFP_MATCHING_FACTOR  0.9f   /**< Macro represents fingerprint matching factor */

/** enum to represent user interface menu level */
typedef enum MFP_menuLevel_e
{
    MFP_UI_MENU_LEVEL_0,
```

The Editor tile allows creating and changing files.
More functions are available by using the context menu.

3.3.4 Console

Picture 13: Console Tile



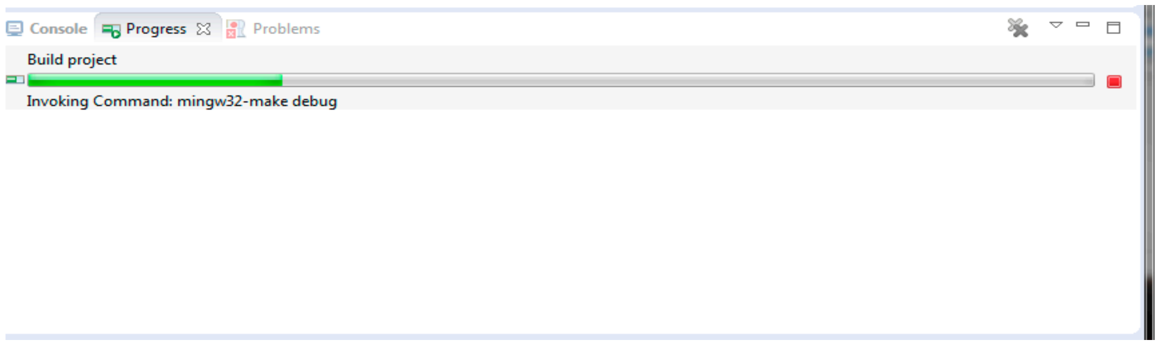
CDT Build Console [APT_appTemplate]

```
17:00:11 **** Build of configuration BUILD HW_XDK_v1 for project APT_appTemplate ****
mingw32-make debug
C:/XDK-Workbench/make\mingw32-make.exe -C C:/XDK-Workbench/SDK/xdk110/make -j debug
make\mingw32-make.exe[1]: Entering directory 'C:/XDK-Workbench/SDK/xdk110/make'
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/portable/ARM_CM3/port.c
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/queue.c
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/list.c
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/portable/MemMang/heap_4.c
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/tasks.c
[ CC ] C:/XDK-Workbench/SDK/lib/FreeRTOS/source/timers.c
[ CC ] C:/XDK-Workbench/SDK/lib/EMLib/Device/SiliconLabs/EFM32GG/Source/system_efm32gg.c
[ CC ] C:/XDK-Workbench/SDK/lib/EMLib/emlib/src/em_assert.c
[ CC ] C:/XDK-Workbench/SDK/lib/EMLib/emlib/src/em_cmu.c
[ CC ] C:/XDK-Workbench/SDK/lib/EMLib/emlib/src/em_msc.c
```

The “Console” tab shows an overview of the status of the Workbench, allows sending data to the XDK and shows output coming from XDK via USB. Depending on the current status, some buttons are available on the upper right side of the tile. These buttons contain functions like “go to next error”, “clear console”, etc.

There are two additional tabs. “Progress” shows the current status of e.g. flashing.

Picture 14: Progress Tile



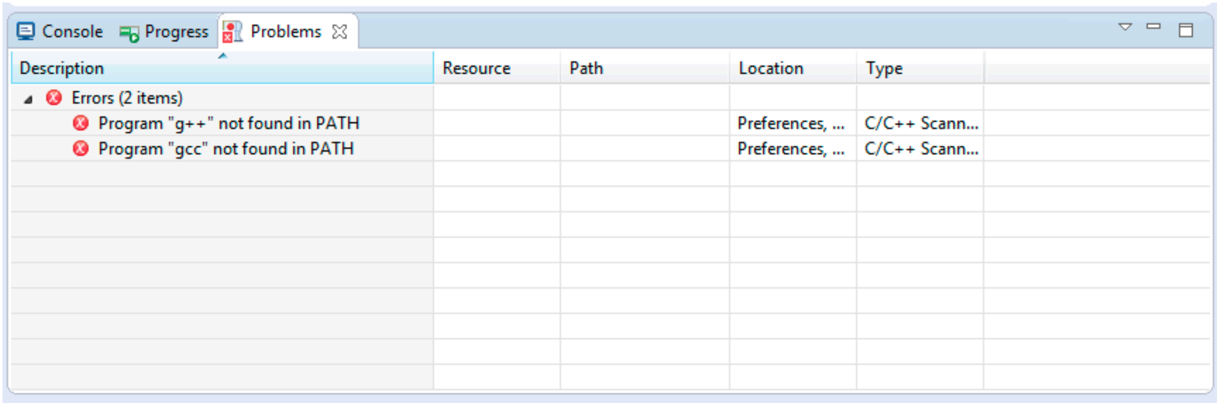
Build project

Invoking Command: mingw32-make debug

You can cancel the operation by clicking on the red button on the right side of the progess bar.

The “Problems” tab displays existing problems detected by the XDK Workbench.

Picture 15: Problems Tile



Description	Resource	Path	Location	Type
Errors (2 items)				
Program "g++" not found in PATH			Preferences, ...	C/C++ Scann...
Program "gcc" not found in PATH			Preferences, ...	C/C++ Scann...

3.4 Build System

3.4.1 Makefile

XDK uses a make-based build system. Make is a powerful tool that uses so-called Makefiles to specify which files are required to compile a project.

The Manual of Make (http://www.gnu.org/software/make/manual/make.html) contains all the details that may be required to understand the behavior and inner workings of make.

Each XDK application contains a subfolder called "make", which in turn contains the "Makefile". The Makefile includes "xdk110/make/application.mk", which contains all the common elements of the application Makefiles. While the overall build system of XDK works out of the box, certain modifications to the Makefile of your applications may be required, as outlined in this page.

Each Makefile contains at least the following three targets:

Debug:	Enables assertions and includes debug symbols
Release:	Disables assertions and does not include debug symbols
All:	Produces a release build

Which variables should I touch in the makefile?

Here you see an exemplary excerpt from an XDK application makefile:

Code 1: application makefile

```
include $(BCDS_BASE_DIR)/Common/application.mk

BCDS_CFLAGS_COMMON = -DMY_DEFINE

PROJECTNAME = XDK110_myExample
override BCDS_CFLAGS_DEBUG_COMMON += $(BCDS_CFLAGS_COMMON)
override ASMFLAGS += $(ASMFLAGS_APP)
```

Below you will find some elements from the Makefile:

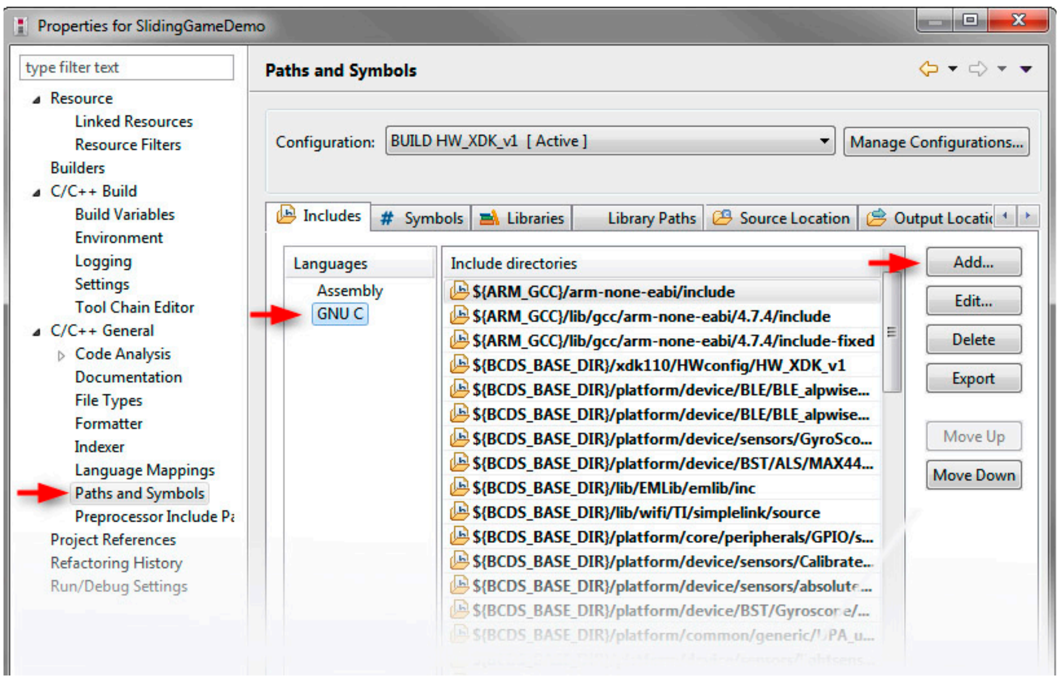
Table 11: Elements from the Makefile

Variable	Meaning
BCDS_CFLAGS_COMMON	Defines application-specific CFLAGS (see below)
XDK_APP_EXECUTABLE_NAME	Defines the project name, which is used to determine the name of your executable file
BCDS_XDK_INCLUDES	Defines the search path' for include files (How Can I Add a New Source File to the Project?)
BCDS_XDK_APP_SOURCE_FILES	Defines the c-files that are taken into account when compiling your project (How Can I Add a New Source File to the Project?)
BCDS_SYSTEM_STARTUP_METHOD	Defines whether XDK uses the standard or a custom initialization routine (see BCDS_SYSTEM_STARTUP_METHOD

How Can I Add a New Source File to the Project?

To add a source file, you have to update the BCDS_XDK_APP_SOURCE_FILES variable to include all files that you want to compile. Furthermore, the BCDS_XDK_INCLUDES variable must be updated to include all path' in which include files may be residing. After adding new include path', make sure the change in the XDK Workbench project configuration (see below) is reflected.

Picture 16: Paths and Symbols



Configuration: BUILD HW_XDK_v1 [Active]

Include directories
${ARM_GCC}/arm-none-eabi/include
${ARM_GCC}/lib/gcc/arm-none-eabi/4.7.4/include
${ARM_GCC}/lib/gcc/arm-none-eabi/4.7.4/include-fixed
${BCDS_BASE_DIR}/xdk110/HWconfig/HW_XDK_v1
${BCDS_BASE_DIR}/platform/device/BLE/BLE_alpwise...
${BCDS_BASE_DIR}/platform/device/BLE/BLE_alpwise...
${BCDS_BASE_DIR}/platform/device/sensors/GyroSco...
${BCDS_BASE_DIR}/platform/device/BST/ALS/MAX44...
${BCDS_BASE_DIR}/lib/EMLib/emlib/inc
${BCDS_BASE_DIR}/lib/wifi/TI/simplelink/source
${BCDS_BASE_DIR}/platform/core/peripherals/GPIO/s...
${BCDS_BASE_DIR}/platform/device/sensors/Calibrate...
${BCDS_BASE_DIR}/platform/device/sensors/absolute...
${BCDS_BASE_DIR}/platform/device/BST/Gyroscope/...
${BCDS_BASE_DIR}/platform/common/generic/UPA_u...

1. Click on File\Properties (or ALT + ENTER).
2. Click on Paths and Symbols.
3. Choose the language.
4. Add the desired directories.

How Can I Change CFLAGS in my Project?

CFLAGS are flags passed on to the compiler. They allow you to configure various things in your software and in the behavior of the compiler. To add specific CFLAGS for your application, please create a new line in the Makefile of your project:

Code 2: CFLAGS

```
BCDS_CFLAGS_COMMON = -DMY_DEFINE
```

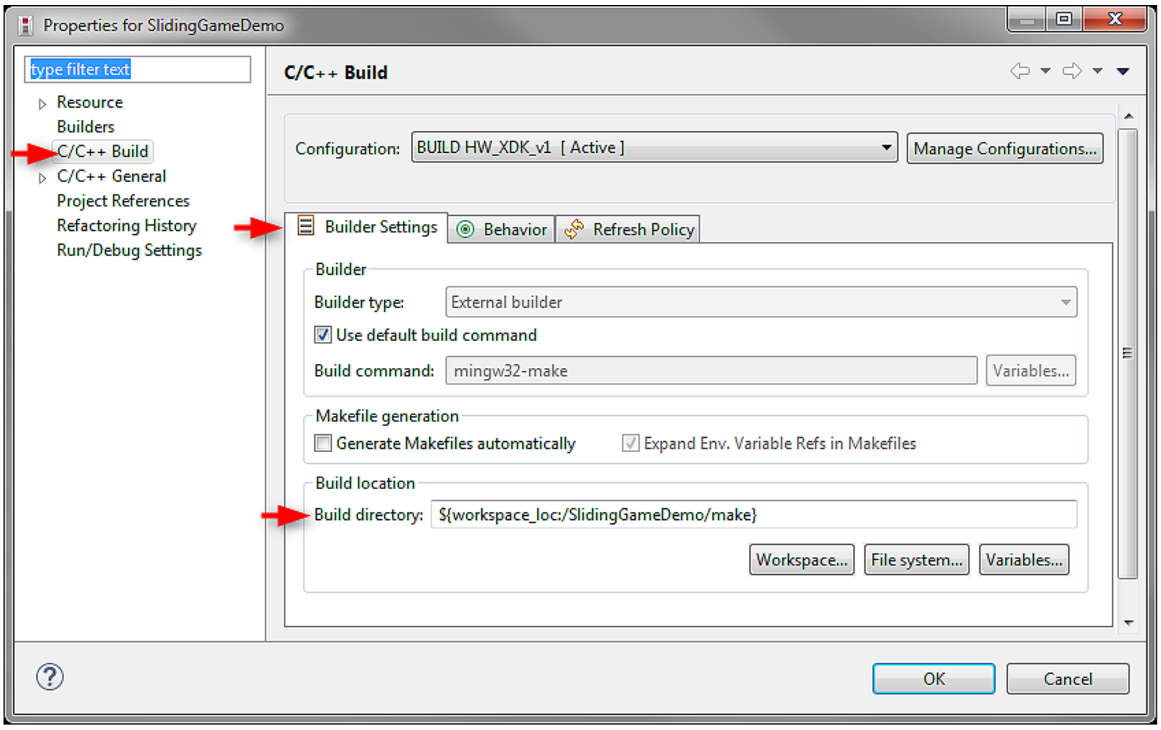
In this line, you can add your own CFLAGS.

How Can I Rename a Project in XDK Workbench?

If you rename a project in XDK Workbench, the project configuration as well as the Makefile has to be modified.

1. Click your project in XDK Workbench and select "Properties".
2. Go into the "C/C++ Build" tab and update the "BuildDirectory". To do this, click on "Workspace" and select the "make" folder inside your renamed application.

Picture 17: C/C++ Build


Properties for SlidingGameDemo

C/C++ Build

Configuration: BUILD HW_XDK_v1 [Active]

Builder type: External builder

☑ Use default build command

Build command: mingw32-make

☐ Generate Makefiles automatically

☑ Expand Env. Variable Refs in Makefiles

Build directory: ${workspace_loc:/SlidingGameDemo/make}

3. Open up the Makefile in your renamed application. You may want to modify the BCDS_APP_NAME, as it defines the name of your executable file afterwards. Furthermore, you may want to add and/or change the files listed in BCDS_XDK_APP_SOURCE_FILES.

3.5 Operating XDK

The XDK Workbench indicates in which operation mode XDK is. XDK must be in bootloader mode to flash via XDK Workbench. The XDK Workbench will automatically put XDK into bootloader mode before flashing. If this does not work, XDK must be placed manually into bootloader.

3.5.1 USB Port

XDK is equipped with a micro USB port to connect it to the PC. This port is directly connected to the USB controller on the MCU of XDK and its function is hence dependent on the software running on XDK. XDK SDK defines a standard behavior of the USB port and allows the user to send and receive data via USB.

Unique Serial Number

Each XDK has a unique iSerialNumber which is identical to the MCU ID.

Device String Allows Mode Detection

- XDK has different iProduct strings for bootloader and application modes
- Refer to: iProduct: http://www.beyondlogic.org/usbnutshell/usb5.shtml
- XDKs iProducts
    - Bootloader mode: XDK Bootloader
    - Application mode: XDK Application

CDC ACM Device

- XDK is recognized as CDC/ACM device via USB
    - Refer to: http://en.wikipedia.org/wiki/USB_communications_device_class
    - On Windows, this creates a virtual COM port
-
- XDK USB IDs
    - idProduct: 0x017B
    - idVendor: 0x108C
-
- Bidirectional Communication
    - XDK is recognized as a virtual COM port on Windows
    - printf outputs of XDK are redirected to the COM port
    - "USB" module of XDK API can be used to receive data from the PC

When connected to a PC, XDK is recognized as a CDC ACM device. CDC stands for "communications device class", while ACM stands for "Abstract Control Model". Essentially, this means, that XDK emulates a serial port (so-called virtual serial port) over the USB connection. Operating systems such as Mac OS X and Linux ship with standard drivers for such devices. For Windows, XDK Workbench automatically installs the necessary drivers. Windows assigns a port identifier "COMx" to XDK, where the number x is incremented when new devices are connected. Each XDK has a unique USB serial number. This ensures that no matter to which USB port of your PC you connect XDK; the assigned port identifier will always be the same.
When running software on XDK, issuing a printf command will send data over the virtual serial port. This can be used for simple debugging purposes as well as for sending data over the virtual serial port. Furthermore, the XDK SDK offers a USB module that also allows receiving data over the serial port. Using the function USB_callBackMapping, you can easily register a callback that is called when data is received over the virtual serial port.

Reset/Reboot XDK

As outlined in Approach 3 in chapter 3.5.2, XDK can receive commands from XDK Workbench over the virtual serial port. Commands are parsed and executed by the URU module in XDK SDK. At the moment, XDK understands the following commands:

#reBoot$:	Reboot XDK and go into bootloader mode
#reSet$:	Reboot XDK and go into application mode

Sending these commands to XDK via the virtual serial port will either get XDK into bootloader mode or restart XDK into application mode. When sending application data to XDK, please make sure you do not use the reserved strings, as XDK might restart unexpectedly.

XDK Windows Driver

Unfortunately, the driver for CDC ACM devices in Microsoft Windows (which is also used by XDK) misbehaves under certain circumstances. When a USB CDC ACM device is connected and the respective port identifier is in use (by Tera Term for example) and then the device suddenly disappears, the driver keeps the port identifier blocked. When the device reappears and the identifier is still in use, the device will not be usable. To work around this issue, the port has to be unused (e.g. by closing Tera Term) and the device has to be reconnected.

Note: Disappearing Device

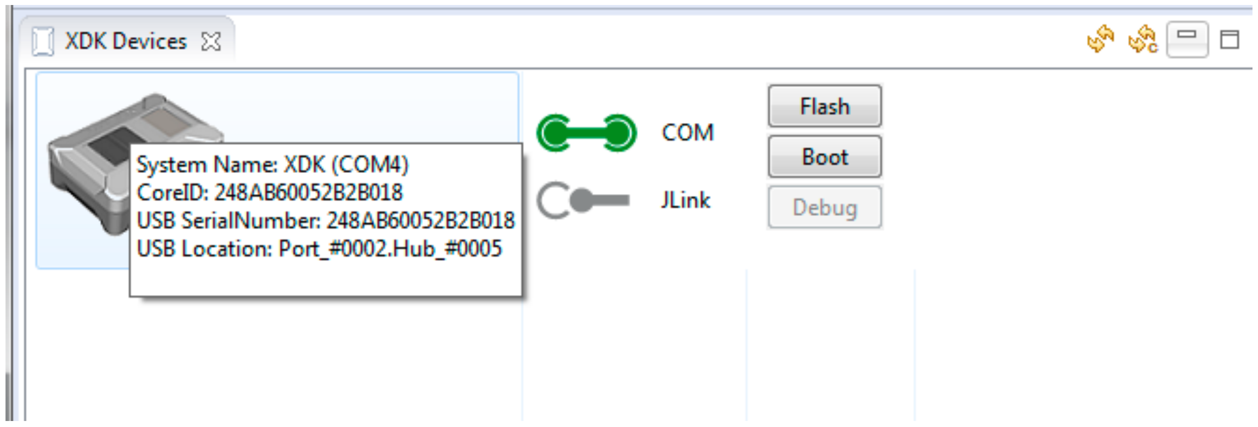
Due to the design of XDK's USB interface, reboot or resetting XDK as well as switching it off and on again make the device disappear from Windows' perspective.

XDK Workbench works around this issue by closing the port identifier whenever XDK is disappearing. However, third party applications usually will not do this. So, whenever XDK disappears, please close your terminal session and restart it once XDK is available again.
The default path for the driver is C:\XDK-Workbench\SDK\drivers_win32.

Using a Serial Terminal

To communicate with XDK, a standard serial terminal application can be used. For example using the third party Tera Term allows opening the virtual serial port of XDK and communicating with XDK. Please make sure that XDK Workbench does not run when you want to communicate with XDK with a third party program, as the two might interfere with each other. When working with XDK, please also pay close attention to the problem outlined in chapter 3.6.1 “XDK Windows Driver”.
To figure out what the port identifier of your XDK is, simply move your mouse over your XDK in XDK Workbench. A mouseover as shown in the picture will show up and give you the "COMx" identifier of your XDK.

Picture 18: Mouseover for XDK Port Identifier



System Name: XDK (COM4)
CoreID: 248AB60052B2B018
USB SerialNumber: 248AB60052B2B018
USB Location: Port_#0002.Hub_#0005

3.5.2 Operation Modes

This chapter describes the different operation modes of the XDK, how to determine in which mode XDK is and how to switch between them. Please note that in normal operation, XDK Workbench will automatically detect the mode XDK currently is in. Furthermore, XDK Workbench will try to automatically perform the mode switch that is required for its correct function. However, there may be certain situations in which the automatic detection or the automated switching does not work. In general, XDK knows the following modes:

- Bootloader Mode
- Application Mode
- Assertion
- Stack Overflow

Bootloader Mode

The XDK bootloader is stored in the first 64 kB of XDKs flash memory. It is write protected and represents the only piece of software on XDK that shall not be modified by an XDK user. When the bootloader is no longer present or becomes corrupted, an update or recovery is only possible via the J-Link JTAG Adapter (see below).
The bootloader allows uploading XDK applications via USB (see XDK Bootloader User Guide for further information). The bootloader is write-protected and can only be overwritten or updated using the J-Link JTAG Adapter (sold separately) via XDK Workbench.
When powering up, XDK will automatically go into the bootloader mode, which is indicated by the red LED. If XDK finds a valid application, the bootloader will automatically turn off the red LED and start the application.
In case no valid application is found or XDK is forced to go into bootloader mode, the yellow LED indicates whether XDK is successfully connected to a PC. Once the red and yellow LEDs are solid on, XDK shows up in the device view of XDK Workbench and can be programmed.

How to engage the Bootloader?

To get XDK into the bootloader, XDK Workbench can be used. Right-click on the XDK and select "Goto Bootloader". In case XDK does not respond (and is possibly not even recognized over USB), then you can manually force XDK into bootloader mode:

- Switch off XDK
- Press and hold Button 1
- Turn on XDK
- Release Button 1 as soon as red LED turns on

This approach is the "last resort" when XDK does not respond. Even when XDK is not recognized over USB, approach 1 will still work. There are two more possibilities to set XDK into bootloader mode.

Approach 2:

- Get XDK into application mode (typically by switching it on)
- Connect it to your PC
- Start XDK Workbench
- Right-click on your XDK in the XDK Device View and select "Go to Bootloader"

The second approach will set a flag in the user page of the MCU of XDK. When XDK reboots, the bootloader reads the flag and engages itself. Only after booting an application, the flag is reset.

Approach 3:

- Get XDK into application mode (typically by switching it on)
- Connect it to your PC
- Connect to XDK with a serial terminal program (see XDK_USB_DEVICE_HANDLING_Terminal)
- Send the following string: #reBoot$
- XDK will automatically reboot and go into the bootloader

This is technically the same as Approach 2. More information on the commands that can be sent by USB can be found in chapter 3.5.1.

Binary images uploaded via the bootloader must be transferred in the XMODEM-CRC format.

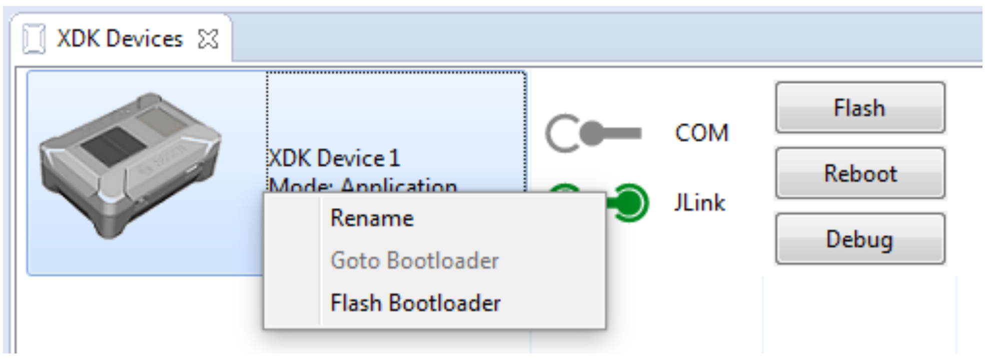
Table 12: Explanation and Meaning of Commands

Command	Meaning	Explanation
u	Upload application	This command lets the user upload an application to the flash, while keeping the bootloader intact. For an application to work correctly it must use a linker file which places the application start address at 0x00010000 for the EFM32GG. The application is transferred using the XMODEM-CRC protocol.
i	Information	This command returns the bootloader version and unique chip id.
b	Boot application	This command will start the uploaded application.
v	Verify flash checksum	This command calculates the CRC-16 checksum of the entire flash and prints it.
c	Verify application checksum	This command calculates the CRC-16 checksum of the application and prints it.
n	Verify user page checksum	This command calculates the CRC-16 checksum of the user page and prints it.
n	Verify lockpage checksum	This command calculates the CRC-16 checksum of the lockpage and prints it.
r	Reset	This command resets the XDK device.

Bootloader Update

Updating the bootloader requires a J-Link JTAG Adapter (sold separately). To perform the update please connect your J-Link adapter to your XDK as well as your PC via USB.
Start your XDK Workbench and make sure that your XDK shows up as having a J-Link connection. Right clicking on the XDK will open a context menu (Picture 28).

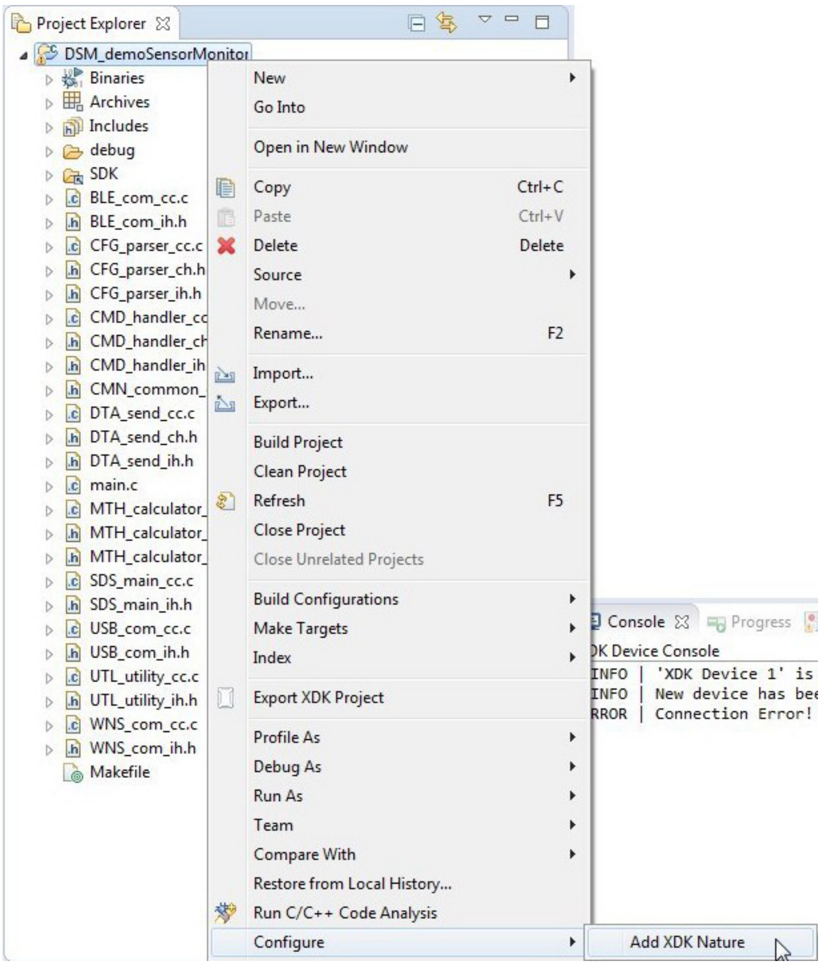
Picture 19: XDK Devices Context Menu



Select „Flash Bootloader“ and then select the bootloader you want to flash onto your XDK Device. The XDK Workbench thereupon updates the bootloader and re-establishes the write protection for the bootloader section.

To ensure that your applications remain working after a bootloader update you have to add a XDK nature to your projects by right clicking on every existing project > *Configure* > *Add XDK Nature* in your *Project Explorer* as shown in the screenshot below.

Picture 20: Add XDK Nature



```
INFO | 'XDK Device 1' is
INFO | New device has bee
RROR | Connection Error!
```

Application Mode/Startup Guide

This simple guide is intended for developers who are going to develop applications for XDK, to know the basic startup procedure of the system before they start writing their application.

XDK software can be configured to start in two ways as described below. The configuration can be done using the BCDS_SYSTEM_STARTUP_METHOD macro present in the application Makefile. By default XDK is shipped with DEFAULT_STARTUP enabled. Refer the code block below representing the change.

Code 3: XDK Custom Startup

```
export BCDS_SYSTEM_STARTUP_METHOD=CUSTOM_STARTUP
```

Default Startup

The `Main()` function implemented in the SystemStartUp module will be the first C function executed during power ON and it does the following:

1. EFM32 chip is configured to a proper state with the help of the library function exported by emlib.
2. All interrupt sources are configured to maximum priority to ensure that ISR runs at highest priority.
3. System peripherals like GPIO, I2C and USB are initialized to a proper state.
4. GPIO pins are configured to its default value.
5. The user page module is initialized. It is used to save configuration information like Wi-Fi MAC address, Bluetooth MAC address, etc., in the user page area of flash.
6. Creates a default application specific initialization CmdProcessor and start it.
7. Gives control to the operating system by starting the task scheduler of FreeRTOS.

Custom Startup

The custom startup procedure will exclude steps 3 to 5 of the default startup procedure described above, to give users the flexibility to have their own way of initializing system peripherals. Users can choose to initialize and configure the peripherals which they want to use in their application. If the custom start-up procedure is configured, users have to ensure that the GPIO pins they are using are initialized to a proper default state. In both of the above configurations XDK's SystemStartUp module will schedule a default application-specific initialization function "appInitSystem" which will run in the CmdProcessor context with task priority 2.

Note: Operation System Tag Priority

When creating the OS task inside the place holder "appInitSystem"" the priority of OS tasks created should be less than or equal to 2, because assigning a priority greater than 2 may block all timer tasks having lower priority.

So all user applications are expected to have the implementation of the task "appInitSystem", and this will be the place holder for the application-specific initialization function. The application-specific initialization function can further create either "timers" or "OS tasks" as suitable for its purpose.

Task Creation and Their Priority

Users can find configuration related to FreeRTOS in a file called FreeRTOSConfig.h and it is present in path "xdk110\Common\config". This contains information like maximum task priority, stack size, heap size, software timer configurations, CPU clock and interrupt priority configurations. As explained, users can create "Timer" or "OS Task" for their application use. Since the creation of OS task requires the user to specify the task priority and stack size, we expect the user to understand the task priorities of FreeRTOS, if, they prefer to create and use OS task for their application (see FreeRTOS documentation). The maximum priority of a task in XDK software is presently configured to 5, which is controlled by the configMAX_PRIORITIES macro. Considering the fact that the interrupt routines should run in the highest task priority, XDK assigns maximum priority to its interrupt sources. So, any task running in XDK should have the priority less than the configMAX_PRIORITIES that is 5. For a user who does not want to get into the complexities of handling independent OS tasks, we suggest to use timers to realize their use cases.

Assertion

Assertions are used in XDK to ensure certain conditions that must be met when calling a function or executing a given piece of code. The developer uses assertions to indicate explicit assumptions that he makes during the development of the code.

In case one of these assumptions is violated during the execution of the application, XDK will stop and turn on all LEDs. Furthermore, the following message will be printed out on the console:

```
asserted at Filename lib/FreeRTOS/source/queue.c, line no 599
```

XDK will have to be reset now, before it can be used again. Either you simply switch it off and on again, and XDK will then execute the installed application. Or you force it into bootloader mode (see 3.6.2) to be able to install a new application onto XDK.

To debug an assertion, dig into the code and find the place where your program asserted. The assertion will tell you which assumption was not fulfilled. Check your code to make sure that all assumptions are met when calling another function.

Stack Overflow

The call stack of a program is the memory that is allocated for holding runtime information that is mostly produced while calling functions. Each task in freeRTOS has a limited stack size. A task that exceeds its own reserved stack size corrupts the memory of other tasks, which will lead to unpredictable behavior. This situation is called stack overflow and must be avoided. XDK uses a heuristic to detect whether a stack overflow has occurred. If this is the case, XDK will turn on the red and yellow LED and print out the following message on the console:

```
----- STACK OVERFLOW -----Task Name: LCA_application -----Current Task Handle: 0xAB
```

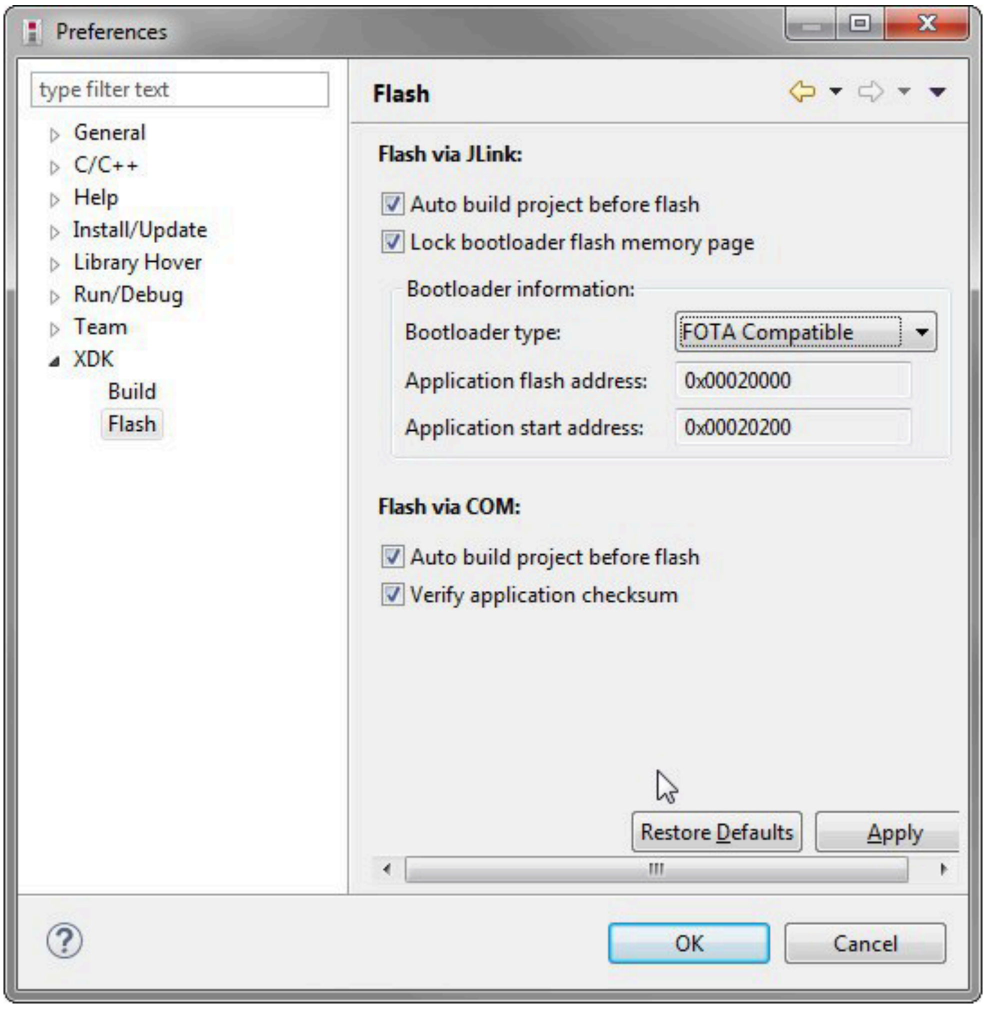
XDK will have to be reset now, before it can be used again. Either you simply switch it off and on again, and XDK will then execute the installed application; or you force it into bootloader mode (see 3.5.2) to be able to install a new application onto XDK.

3.6 Firmware Over the Air (FOTA)

3.6.1 Introduction

Available with the XDK Workbench Version 2.0.0 onwards, the XDK Workbench ships with an additional feature called Firmware Over the Air (FOTA). FOTA is a method to update your firmware wirelessly, i.e. over Wi-Fi. To do so, please make sure that your bootloader is FOTA compatible by overwriting your current bootloader as described in chapter 3.5.2 (if necessary) and your current bootloader type is set to „*FOTA Compatible*“ in the XDK preferences. The following screenshot shows where the configuration can be found in *Window* > *Preferences* > *XDK* > *Flash*.

Picture 21: Bootloader type



Flash

Flash via JLink:

- [✓] Auto build project before flash
- [✓] Lock bootloader flash memory page

Bootloader information:

Bootloader type:	FOTA Compatible
Application flash address:	0x00020000
Application start address:	0x00020200

Flash via COM:

- [✓] Auto build project before flash
- [✓] Verify application checksum

Note that FOTA uses the LWM2M application protocol. In the following, a reference to a LWM2M resource is represented in the notation *LWM2MObjectName.LWM2MResourceName*. Be aware that the implementation of the FirmwareUpdate object is subject to interpretation and that an applicable version of the LWM2M specification is still not defined. To get familiar with LWM2M we recommend the LWM2M Guide on the XDK Help&Learning Page (xdk.io/guides).

3.6.2 FOTA Container

The firmware has to be provided in a FOTA container that consist of a container header and the firmware binary. Please be aware that all fields that are currently used by the bootloader can't be be changed in the future due to the prevailing write-protection of the bootloader section.

FOTA Container Format

Table 13: FOTA Container Format

Section	Name	Length	Default Value	Processing Relevance	Description
Header	Container Header Version	2 bytes (little-endian)	0x0100	Ignored	Used by the application to define supported application versions.
Header	Container Header Size	2 bytes (little-endian)	0x0200	Ignored	Determines the starting position of the firmware binary
Header	Product Class	2 bytes (little-endian)	0x001X	Ignored	Used to check whether the application is downloading firmware for the right product.
Header	Product Variant	2 bytes (little-endian)	0x00	Ignored	Used to check whether the application is downloading firmware for the right region.
Header	Reserved for future	232 bytes (little-endian)	0xFF	Ignored	Currently not in use
Header	Firmware Version	4 bytes (little-endian)	0x0000 01	Yes	Used to check whether the downloaded firmware version is newer than the current firmware version - Firmware with the same version or higher is accepted
Header	Firmware Size	4 bytes (little-endian)	-	Yes	Size of the executable binary - Firmware max size 600 kB
Header	Reserved	256 bytes	0xFF	Ignored	Currently not in use
Header	Firmware CRC	4 bytes (little-endian)	CRC-32	Yes	Used to ensure that the firmware was not corrupted during copy operations

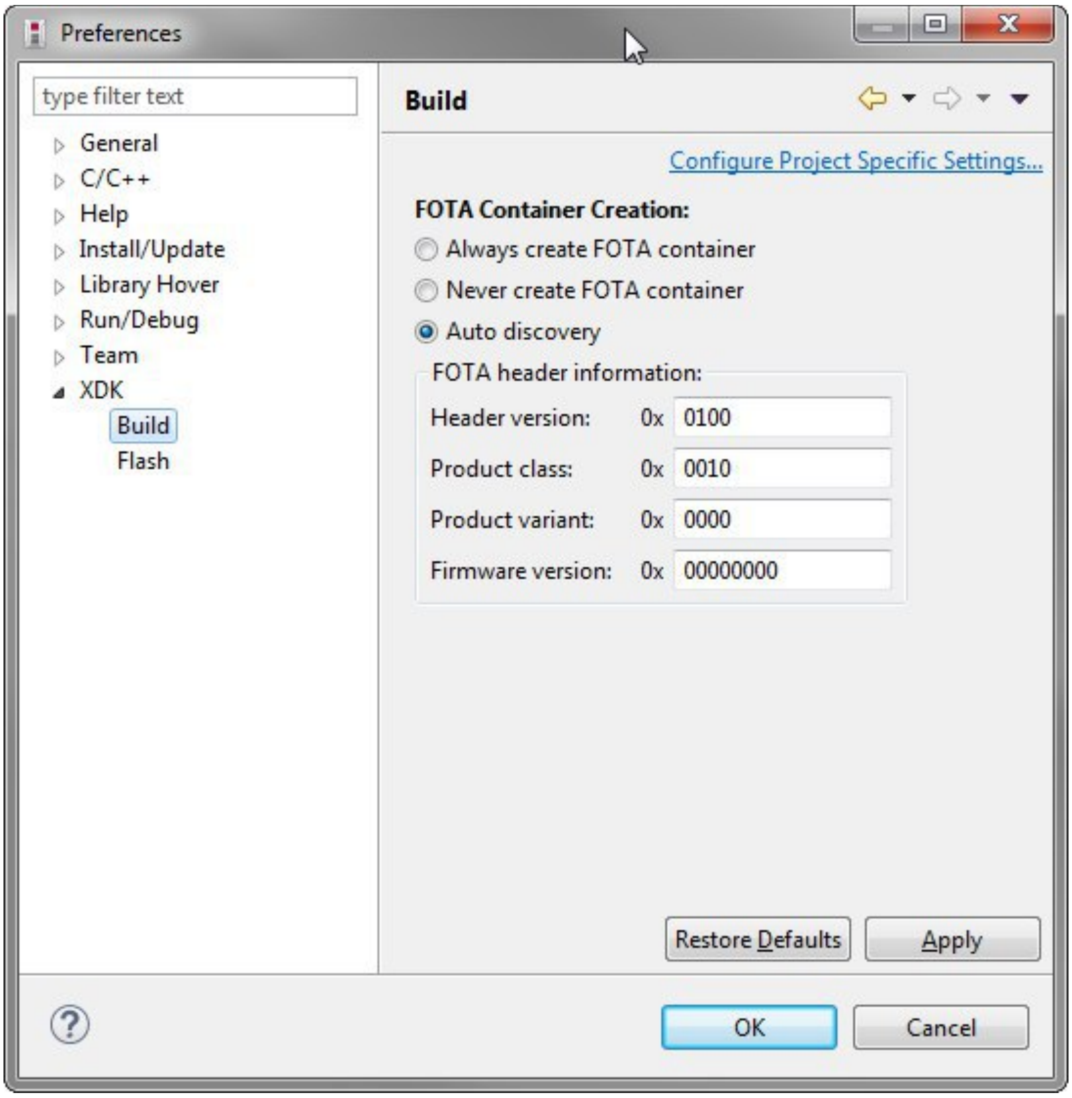
Header	Container Header CRC	4 bytes (little-endian)	CRC-32	Yes	Used to ensure that the container header was not corrupted during copy operations
Firmware	Firmware Binary	n bytes (byte stream)	-	Ignored	Executable code
Footer	Firmware Signature	256 bytes (byte stream)	-	Ignored	Used to ensure that the downloaded firmware is signed by an authority

3.6.4 FOTA Container Creation

Using the XDK Workbench

The FOTA Container can be created using the XDK Workbench. The following screenshot shows where the configuration of the „*FOTA Container Creation*“ can be found in *Window* > *Preferences* > *XDK* > *Build*.

Picture 22: FOTA Container Creation



FOTA Container Creation:
- Always create FOTA container
- Never create FOTA container
- [✓ Auto discovery]

FOTA header information:

Header version:	0x 0100
Product class:	0x 0010
Product variant:	0x 0000
Firmware version:	0x 00000000

Using the Command Line Tools

Instead of using the XDK Workbench, the FOTA container can also be created by using the command line tools „*create_fota_container.jar*“ and „*append_signature.jar*“ that can be found inside the folder „fota“ in your XDK Workbench installation directory.

Table 14: Command Line arguments create_fota_container.jar

Argument	Relevance	Meaning
-fv, --firmwareVersion <arg>	Required	Firmware Version
-h	-	Shows help content
-hv, --headerVersion <arg>	Required	The version of FOTA container header
-i, --inputFile <arg>	Required	Path of the input binary file
-o, --outputFile <arg>	Optional	Path of the output binary file (if not set, the input file will be overwritten)
-pc, --productClass <arg>	Required	Product Class
-hv, --headerVersion <arg>	Required	Product Variant

Table 15: Command Line arguments append_signature.jar

Argument	Relevance	Meaning
-h	-	Shows help content
-i, --inputFile <arg>	Required	Path of the input binary file
-o, --outputFile <arg>	Optional	Path of the output binary file (if not set, the input file will be overwritten)
-s, --signature <arg>	Required	Firmware Signature as hexadecimal string

Note that the FOTA container file should be copied to the MCU with negative offset, so that the starting address of the executable part of the binary is at the correct MCU flash address and the container header is located at the addresses right before that.

3.6.5 FOTA Update Process

1. Initiation

To initiate the firmware update process, a sensor has to be registered at the LWM2M server and expose the *FirmwareUpdate* object. The server then sends the *FirmwareUpdate.PackageURI* to the LWM2M client that serves the FOTA container. The client reacts to a change of the Package URI by setting *FirmwareUpdate.UpdateResult* = 0 (Default) and checking the validity of the provided Package URI. In case the URI is valid, the LWM2M client sets *FirmwareUpdate.State* = 1 (Downloading) to initiate the download process. Otherwise the client sets *FirmwareUpdate. UpdateResult* = 7 (Invalid URI) and aborts the further update process.

2. Download

While *FirmwareUpdate.State* = 1 (Downloading) the main application task periodically requests the next firmware package until all packets are received, unless a communication sequence is running within normal operation (park notification, LWM2M update registration).
Upon receipt of a package, its content is stored sequentially to update the firmware starting at a specific download address. Once the container header has been fully downloaded, the LWM2M client checks whether enough memory is available and verifies the version of the downloaded firmware binary. In case there is not enough memory available or the version of the downloaded firmware binary is not greater than the currently running version the update result is set to FirmwareUpdate.UpdateResult = 2 (Not enough storage) or *FirmwareUpdate.UpdateResult* = 6 (unsupported package type) and the further update process is aborted. Once the binary is completely downloaded, the update result is set to *FirmwareUpdate.State* = 2 (Downloaded).
Note that the firmware download will automatically resume whenever the node was restarted during the download process.

3. Verification

The verification process starts as soon as the LWM2M server calls *FirmwareUpdate.Update*. The node then verifies the CRC of both the firmware and the container header. If the CRC is invalid the node sets *FirmwareUpdate.UpdateResult* = 5 (CRC check failure).

4. Booting

If performing the update resource was successful and the application has rebooted properly and resumed normal operation, it needs to set *FirmwareUpdate.UpdateResult* = 1 (Firmware updated successfully) and *FirmwareUpdate.State = 1* (Idle).

4. Interfaces

4.1 Extension Interface

The connection interface allows access to additional functionality of the MCU, such as radios or sensors. This way, developers can use the hardware and software of XDK for testing out new components with minimal application effort. All MCU pins are GPIOs and can be configured freely within the limitations of the MCU. Suggestions for usage are given to highlight dedicated functions available on these pins in the Table 7. For further information on the MCU pin functionality, please refer to the MCU data sheet.

Warning: Damage due to incorrect Voltage

The equipment can be damaged by using an inadequate power source. This can cause serious injury and damage to materials.
Do not connect the power supply pins to an external power source!

4.2 Connecting External Components to the XDK

4.2.1 Connecting the “XDK Gateway” Extension Board

Connect the 26-pin cable (included in delivery) to the “XDK Gateway” board and to the 26-pin connector of the XDK 110. The “XDK Gateway” offers a simple way to implement additional functions. It is optimized for the use with breadboards.

4.2.2 Connecting Your Own Components to the XDK Gateway Board

The XDK Gateway board I/O pins are labeled with the respective MCU pin designator. Some of the pins can be configured to cater several functions. However, due to shared resources with internal components, restrictions apply to the possible configurations of the extension bus.

Warning: Function Limitations

Before you start connecting external components to the XDK, please note the following limitations:

GPIO pins voltage is 2.5V. If you connect external components that work on a different voltage level, a level shifter has to be used to ensure communication with the MCU. **Failing to do so can lead to permanent damage of the XDK!**
The microcontroller supports four GPIO pin drive strength settings: 0.5, 2, 6 and 20 mA. The default drive strength is set to 6 mA and an alternative level can be defined for each port. Pins of a given port can select the configured alternative drive strength individually. While each pin can receive or provide maximum 20 mA current, there is a limit on the maximum combined current sourced or sank by all GPIOs. As the XDK board itself has many useful peripherals connected to the MCU that may require specific alternative drive strength settings if enabled, it is recommended to use the default **6 mA drive strength for all IO pins** of the extension board connector and to limit the total amount of current sourced or sank through the extension board bus to **50 mA.**
Exceeding these limits might permanently damage the microcontroller!

Furthermore, please consider the following characteristics:

- By default, the 3V3 power is in a disabled state. In your application, please use the function Board_EnablePowerSupply3V3(EXTENSION_BOARD) to enable the 3V3 on the extension bus.
- The pins of the extension bus are protected against ESD up to 4kV. This includes a serial resistance of 40ohm into the signal path. While this is no problem for communication, it needs to be considered when powering external components via GPIO, e.g. an LED.
- Any pin that you intend to use will have to be configured accordingly. Please refer to the GPIO module for doing so. All pins are assigned with default modes and values in file *xdk110/Platform/BSP/source/BSP_BoardSettings.h.* Please refer to the interface for further information.
- Most of the pins provided on the extension bus can be freely configured, e.g. as GPIO, or following the recommendations in the table below. However, the following pins differ electrically from the standard:
    - Pins B9 and B10 are equipped with an internal pull-up resistor of 3.32kOhm and are therefore usable for I2C communication only.
    - Pins PD5 and PD6 have anti-alias (low-pass) filters with a cut off frequency of ~400 MHz

Table 15: Connector Pin Assignment of the Extension Bus

Connector Pin	MCU Pin	Suggested Use	Multiplex location	Macro
A1	PA0	Timer0 compare	TIMER0 #4	EXTENSION_TIM0_CC0
A2	PC0	Timer0 compare	TIMER0 #4	EXTENSION_TIM0_CC1
A3	PC1	Timer0 compare	TIMER0 #4	EXTENSION_TIM0_CC2
A4	PC2	Timer0 Dead time insertion	TIMER0 #4	EXTENSION_TIM0_CDTIO
A5	PC3	Timer0 Dead time insertion	TIMER0 #4	EXTENSION_TIM0_CDTI1
A6	PC4	Timer0 Dead time insertion	TIMER0 #4	EXTENSION_TIM0_CDTI2
A7	PC8	Timer2 Capture operations	TIMER2 #2	EXTENSION_ TIM2_CC0
A8	PC9	Timer2 Capture operations	TIMER2 #2	EXTENSION _TIM2_CC1
A9	PC10	Timer2 Capture operations	TIMER2 #2	EXTENSION_ TIM2_CC2
A10	PD6	ADC0 Channel6	n/a	EXTENSION _ADC0_CH6
A11	PD5	ADC0 Channel5	n/a	EXTENSION _ADC0_CH5
A12	PA1	General purpose I/ O	n/a	EXTENSION _GPIO_IN_OUT_0

A13	PE2	General purpose I/O	n/a	EXTENSION _GPIO_IN_OUT_1
B1	PB9	UART1 TX	UART1 #2	EXTENSION _UART1_TX
B2	PB10	UART1 RX	UART1 #2	EXTENSION _UART1_RX
B3	PB2	UART1 RTS	n/a	EXTENSION _UART1_RTS
B4	PF6	UART1 CTS	n/a	EXTENSION _UART1_CTS
B5	PB4	SPI MISO	USART2 #1	EXTENSION _US2_MISO
B6	PB3	SPI MOSI	USART2 #1	EXTENSION _US2_MOSI
B7	PB5	SPI Clock	USART2 #1	EXTENSION _US2_SCK
B8	PD8	SPI Chip select	n/a	EXTENSION _US2_CS
B9	PB11	I2C1 data line (pull-up)	I2C1 #1	EXTENSION _I2C1_SDA
B10	PB12	I2C1 clock line (pull-up)	I2C1 #1	EXTENSION _I2C1_SCL
B11	2V5	Power ; Limit 100mA continuous/ peak	n/a	
B12	3V3	Power ; Limit 100mA continuous/ peak	n/a	
B13	GND	Power	n/a	

For details on multiplexing, please refer to the MCU datasheet.

4.2.3 Software Example

This example outlines how to realize LED blinking functionality on the XDK Extension Bus.

Code 5: Led task

```
#include "BCDS_MCU_GPIO.h"
#include "BCDS_MCU_GPIO_Handle.h"

void extensionLedTask(void)
{
    MCU_GPIO_Handle_T GPIOA;

    /* initialize local variables */
    int count = 0;

    GPIOA.Port = gpioPortA;
    GPIOA.Pin = 1;
    GPIOA.Mode = gpioModePushPull;
    GPIOA.InitialState = MCU_GPIO_PIN_STATE_HIGH;

    /* Initialization activities for PTD driver */
    MCU_GPIO_Initialize(&GPIOA);
    /* blinking functionality */
    while(count < 10)
    {
        /* Set data out register for the pin to 1 */
        MCU_GPIO_WritePin(&GPIOA,MCU_GPIO_PIN_STATE_HIGH);
        vTaskDelay (1000);
        /* use suitable delay API. This delay API is for demo only. */
        /* Set data out register for the pin to 0 */
        MCU_GPIO_WritePin(&GPIOA,MCU_GPIO_PIN_STATE_LOW);
        vTaskDelay(1000);
        /* use suitable delay API. This delay API is for demo only. */
        count++;
    }
}

void appInitSystem(void * CmdProcessorHandle, uint32_t param2)
{
    if (CmdProcessorHandle == NULL)
    {
        printf("Command processor handle is null \n\r");
        assert(false);
   }
    BCDS_UNUSED(param2);

   extensionLedTask();
}
```

4.2.4 Mechanical design of extension boards

The extension bus connector of XDK is a 26-pin male connector of type ERNI Male 054595. For your extension boards, please use the same connector and use the flat ribbon cable to connect your extension board to XDK.

Certification Limits

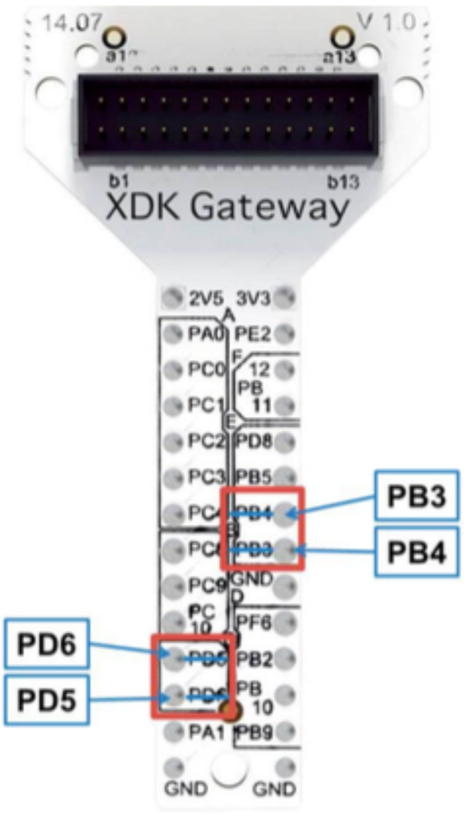
XDK certification does not apply to extension boards other than the “XDK gateway”. If you use your own hardware connected to XDK, you need to check the compliance to legal standards according to your region. Customer-made extension boards are not supported by BCDS and any damage to the XDK caused by connecting improper hardware to XDK will void the warranty.

The MCU pins refer to the Giant Gecko microcontroller on the XDK. For details on the configuration, please refer to the MCU datasheet and MCU Reference Manual.

4.2.5 Errata

Unfortunately there is an error in the labeling of the “XDK Gateway” (Version V1.0) extension board delivered with XDK. Some of the pin names were swapped. The correct pin assignment is shown in the picture below. We apologize for any inconvenience caused by this.

Picture 23: XDK Gateway



3.JTAG Debug Interface

XDK110 can be programmed via the USB connector as described above. For advanced users, it also has the possibility to access the MCU debug port via standard JTAG debug interface for Cortex-M MCUs. You can choose the debugger that matches the JTAG pin configuration in the table below.

Picture 24: JTAG Debug Interface

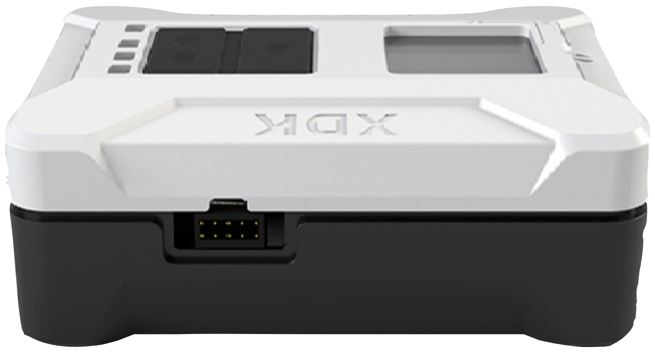


Table 16: XDK JTAG Interface

Pin	Signal	Description
1	POWER	Reference voltage
2	SWDIO	Bi-directional data pin
3	GND	Ground pin
4	SWCLK	Clock signal to target CPU
5	GND	Ground pin
6	SWO	Serial wire output
7	GND	Ground pin
8	NOT USED	
9	GND	Ground pin
10	RESET	Reset

If you do not have a suitable debug probe and feel the need to access the debug port, you can buy a “J-link LITE CortexM-9” debug probe from SEGGER (https://www.segger.com/jlink-lite-cortexm.html) by contacting the SEGGER sales team. Please mention that you will use the debug probe for XDK, as J-link LITE is not sold separately.

Note Cable Configuration

Please note that the Segger “J-link LITE CortexM-9” debug probe will have one pin blocked by a plastic plug. Make sure your remove this plug on the XDK side before connecting the cable.

5. Regulatory information

5.1 RoHS

The XDK110 meets the requirements of the directive regarding restriction of hazardous substances (RoHS), see also:
Directive 2011/65//EU of the European Parliament and of the Council of 8 September 2011 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

5.2 Telecommunication

The XDK110 meets the requirements of the directive on radio equipment and telecommunications terminal equipment (R&TTE) 1999/5/EC.

5.3 Radiofrequency Radiation Exposure and further Information

The radiated output power of the device is far below the FCC radio frequency exposure limits. Nevertheless, the device shall be used in such a manner that the potential for human contact during normal operation is minimized.

This device complies with Part 15 of the FCC Rules and with Industry Canada license-exempt RSS standard(s). Operation is subject to the following three conditions:

- This device may not cause harmful interference, and
- This device must accept any interference received, including interference that may cause undesired operation.
- The minimum distance between body and device should be 20 cm.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux trois conditions suivantes:

- L'appareil ne doit pas produire de brouillage, et
- L'utilisateur de l'appareil doit accepter tout brouillage radioélectrique
- subi, même si le brouillage est susceptible d'en compromettre le
- fonctionnement.
- La distance minimum entre appareil et corps est 20 cm.

5.4 Certification

Europe	CE
U.S	FCC ID 2ADSJXDK110 FC
Canada	IC ID: 12595A-XDK110

Note Forfeiting the Warranty

Opening the casing without authorization could present risks to the user and will void any warranty. Changes or modifications made to this equipment not expressly approved by Bosch may void the declarations of conformity and the FCC authorization to operate this equipment.
Do not open the casing.
Use only original or approved accessories.

---

6. Legal Disclaimer

XDK housing is not to be opened or tampered with. In case of XDK in a single box package a breakout board has been provided with delivery so that the user can access the extension bus of XDK without opening the housing.

XDK is designed for use within environmental conditions as further detailed in the technical handbook. Any use or operation under deviating environmental conditions must be validated and tested on customer’s own responsibility.

Manufacturing of XDK or derivations, adaptations thereof may only be done by BCDS.
XDK is not intended to represent a final design recommendation for any particular application.
Final device in an application has to be designed, tested and validated as per customer’s own defined series production process (e.g. proper circuit board design and layout, heat-sink, proper supply concept etc.) taking into account the applicable state of the art in technology.

The customer shall be responsible for conducting the design of customer’s application or project within the scope of the Internet of Things (hereinafter referred to “Customer Application”) as well as the integration of the XDK into such Customer Application and any adequate operating safeguards taking account of the state of the art in technology and all applicable statutory regulations and provisions. In order to avoid/minimize risks associated with a Customer Application, sufficient validation and testing must be provided by the customer to avoid inherent or procedural hazards.

Bosch assumes no responsibility for the consequences of use of the Customer Application or for any infringement of patents or other rights of third parties which may result from its use in combination with any technology developed or added by the customer.

Neither the XDK nor a potential product derivation, nor the break-out board or any other XDK component adaption are designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Bosch product could create a situation where personal injury or death may occur. The same applies for any kind of nuclear systems or weapons.

In the event of any use of XDK outside the BCDS expressly specified environmental or installation conditions customer shall indemnify and hold BCDS and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use.

7. Document History and Modification

Rev. No.	Chapter	Description of modification/changes	Editor	Date
2.0		Version 2.0 initial release	AFS	2017-08-17