XDK110: Migration Guide

Cross-Domain Development Kit XDK110

Platform for Application Development

XDK110: Migration Guide	
Document revision	1.0
Document release date	**11. May 2018**
Workbench version	3.0.0 and above
Document number	BCDS-XDK110-GUIDE MIGRATION
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. This document is confidential and under NDA inherent with the purchase of an XDK110. **Advance information – Subject to change without notice**

Table of contents

This guide postulates a basic understanding of the XDK and according Workspace. For new users we recommend going through the following guides at xdk.io/guides first:

- *Workbench Installation*
- *Workbench First Steps*
- *XDK Guide FreeRTOS*
-

XDK110
Platform for Application Development

1. General Description

1.1 Introduction

With every new version of the XDK-Workbench, major changes to the SDK and the Workbench itself are made.
This document provides information on the most significant changes that were made from one Workbench version to another.
The purpose is to allow for migration of projects from older to newer versions without much effort.

First, some important concepts on how versions are determined and what changes are generally made between versions are described in the chapter *General*.
In chapter *How To Migrate*, some basic ideas of how to migrate a project efficiently are introduced, that explain how to use this guide as a basis.
Apart from that, there is one chapter for every version change that was determined to have significant changes that require detailed explanation and mention.

For every such chapter, there may be a sub-chapter called *All* that describes changes that affect all projects, regardless from the modules that will be eventually used.

Other than that, there will be a sub-chapter for every noteworthy module or API library that has changed.

2. General

In general, changes are distinguished between how much they affect project code that is based on the previous version of the SDK.
In this context, project code refers to any piece of code written by XDK-Workbench users for applications that run on the XDK.

The three main types of changes are **breaking changes**, **new features** and **bugfixes**.
Breaking changes are the focus of this document.

- **Breaking Changes** (C) break code that depends on APIs or modules, if a project from the previous version is using the new version's SDK. Such a change requires either an adaption of the code using the respective API, or a complete rewrite. An example for this is BLE from 2.0.1 to 3.0.0, where the old API had been closed off, in favor of a platform specific API.
- **New Features** (F) are new endpoints and functions in an API. They do not change anything, and therefore do not cause breaks in the code. New features usually do not have to included in the old code, and that is the case, the new feature would also be mentioned as a *breaking change*. An example for a new feature is the possibility of adding custom headers using the HTTP API.
- **Bugfixes** (B) are changes to the source code that do not break existing project code. They do usually do not require an adaption in project code, but they may make a previously implemented workaround for a certain issue unnecessary.

The XDK-Workbench version is described as a combination of three numbers *x.y.z*, for example 3.0.0.

- The ***x*** number usually increments when major breaking changes are introduced, indicating that migration of older projects may be necessary. New features and bugfixes are also part of the version change.
- The ***y*** number usually increments on major changes and when new features are added. These changes do not break most projects. Bugfixes are also part of the version change.
- The ***z*** number usually increments on minor changes, such as minor features, and bugfixes in the SDK.

Each of the following chapters describes one major version change that requires extensive adaption of most/many older projects, if they were to be migrated from one specific version to the version that follows chronologically.

3. How to migrate

3.1 General Introduction

The SDK offers three main directories. They are **Common**, **Libraries** and **Platform**.

- The **Common** directory contains relevant makefiles, that can be freely edited (only recommended for advanced users). Additionally, the directory contains files for configuring certain parts of the SDK.
- The **Libraries** directory contains third party API, such as FreeRTOS, ServalStack, EMlib and FATfs. The implementation of most of these libraries stays unchanged, and if changes are made, they are usually, such as new features. These libraries are independent from the XDK, and are mostly build to last. The only library actively receiving changes is the *ServalStack*.
- The **Platform** directory contains API provided by BCDS. These API are most commonly designed to offer a layer between third party libraries for hardware and the XDK's system. For example, the sensors of the XDK have hardware-specific API in the 3rd-party library at Libraries/BSTLibTo use the sensors, the physical sensor itself must first be enabled and connected on a very low API level. The sensor values would have to be read from the sensor, converted and made available. The Platform-specific API handles all this, and only requires the user to call very few functions to enable the sensors and retrieve their data.

Generally, as implied by how likely the three directories are subject to change, the content of **Platform** is the most likely source of breaking changes between major updates of the XDK-Workbench.

Every sub-chapter of this guide covers one specific directory and package/module. When migrating a project from one version to another, first look at which sub-chapters are available. If any of the covered modules is used in the project code, check if the change is a breaking change, and modify the code accordingly.

For that, first consider the nature of the change. If it is a simple rename, then modification should be straightforward. If parameters changed, take a look at the header-file that contains the function and see the comment above the function's declaration. Usually, the parameters and the effect of the function are described there. If one or more functions are removed, usually another one is added, that accounts for the removed function. This will usually be mentioned in the corresponding sub-chapter.

Unfortunately, documentation may not be available for every change currently.

If the project does not compile properly after migrating the project, then most likely an unmentioned API from the **Platform** directory had been changed. In that case, see where that API is located in the SDK in the previous version and try to find a corresponding API in the next version.

If the project is based on one of the Workbench-Examples, please see the corresponding sub-chapter *Workbench-Examples* for general information. If an example is mentioned there, it may require changes.

4. Version 1.3.0 to 1.5.2

In the following text, abbreviations are used for breaking changes, new features, and bugfixes as follows.

- (C) - Breaking Change
- (F) - New Feature
- (B) - Bugfixes

4.1 General steps for Migration

The following steps serve as a guideline for attempts in migrating a 1.3.0 project to 1.5.2.

1. Restructure the project directory and use the new correct Makefile, by:
2. Either:
- Creating a new project
- Copying added code into new project from old project

3. Or:
- Delete the "make" folder containing the project application makefile. (Preferably, make a backup beforehand)
- Copy the application makefile from any one of the example application into the application project directory
- Copy the main.c file from any of example application into the application project directory
- Replace your required application makefile changes in the copied example makefile.

4. Apply API changes, when modules, that are mentioned within this chapter, are used.
5. (Re-)compile and verify until the build gets successfully completed

4.2 Common

4.2.1 Application Startup

From 1.3.0 to 1.5.2, the way applications start has been changed.

Table 1. Chanes to XDK Startup from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above
Location of main function	Inside SSU_systemStartUp module and it is common for all application	Inside the application's source directory in *main.c*
Location of application makefile	"make" folder within the "application project directory"	Inside the application's directory, called *Makefile*
First API called after system booting	`void SSU_initSystem(OS_timerHandle_tp xTimer) {}`	`void appInitSystem(xTimerHandle xTimer) {...}`

4.2.2 Deprecations & Renaming

Table 2. Platform SDK changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above	Comments
Platform/BST	Not Deprecated	Deprecated	Platform/SensorUtils implements this functionality now
Platform/OS	Not Deprecated	Deprecated	Direct usage of Free RTOS Library API recommended
Platform/Power	Not Present	Newly Added	Implemented FreeRTOS Sleep Management
Platform/Utils/Source/ Userpage	Not Deprecated	Deprecated	Userpage has been deprecated from the Utils Package
Platform/MiscDrivers/ Source/NVM	Not Present	Present	NVM has been newly implemented instead of Userpage
Platform/Utils/URU	Not Present	Present	Moved to directory xdk110\Common\source in SDK

4.2.3 Renamed / Removed Packages

Table 3. Package changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above
Platform/WIFI	Not Renamed	Renamed to Platform/Wlan
Platform/SDcard	Not Moved	Moved to Platform/MiscDrivers

4.2.4 Removed Files

Table 4. Header File name changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above
XDK device handles header files	`xdk_board.h`	`XdkBoardHandle.h`
XDK device handles source files	`xdk_board.c`	`BoardHandle.c`
XDK custom startup header file	`CSU_chipStartUp_ih.h`	`XdkBoardInitialization.h`
XDK custom startup source file	`CSU_chipStartUp_cc.c`	`BoardInitialization.c`
XDK sensor handles header file	`xdk_sensors.h`	`XdkSensorHandle.h`
XDK sensor handles source file	`xdk_sensors.c`	`SensorHandle.c`
XDK system startup header file	`SSU_systemStartUp_ch.h`	`XdkSystemStartup.h`
XDK system startup source file	`SSU_systemStartUp_cc.c`	`SystemStartup.c`
XDK gpio configuration header file	`PDC_pinDefaultConfig_ih.h`	`XdkGpioConfig.h`
XDK gpio configuration source file	`PDC_pinDefaultConfig_cc.c`	`GpioConfig.c`

4.3 Libraries

4.3.1 FreeRTOS

In 1.3.0, the module *OS*, used to abstract some of the FreeRTOS functions, was part of the Platform modules. This is removed in 1.5.2. Therefore, instead of the abstraction functions, the FreeRTOS calls should be used.

The following tables show which functions from the OS module correspond to which FreeRTOS functions.

4.3.2 Timers

Table 5. Timer function changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above	Remarks
RTOS Timer Creation	`OS_timerCreate((const int8_t *) "Timername", 2, 1, NULL, AppHandler)`	`xTimerCreate((const char * const ) "Timername", Ticks, 1, NULL, AppHandler)`	In XDK 1.3.0 the RTOS timer ticks is converted as part of abstraction
Return type of Timer create	`OS_timerHandle_tp`	`xTimerHandle`	
Pend using timer	`OS_timerPendFunctionCallFromISR(appDeferredISRCallback, NULL, UINT32_C(0))`	`xTimerPendFunctionCallFromISR(appDeferredISRCallback, NULL, UINT8_C(0), &xHigherPriorityTaskWoken)`	In XDK 1.3.0 the task yield from ISR is done as part of abstraction
Timer Start	`OS_timerStart(timerHandle,3)`	`xTimerStart(timerHandle, 3/portTICK_RATE_MS)`	In XDK 1.3.0 the RTOS timer ticks is converted as part of abstraction
Timer Stop	`OS_timerStop(timerHandle, 3)`	`xTimerStop(timerHandle, 3/portTICK_RATE_MS)`	In XDK 1.3.0 the RTOS timer ticks is converted as part of abstraction

When timer ticks need to be specified as input to FreeRTOS API, then it has to be explicitly converted as shown below.

Code 1. Example implementation for converting integer values into freeRTOS ticks

```
uint32_t Ticks = 2;

if (Ticks != UINT32_MAX) /* Validated for portMAX_DELAY to assist the task to wait
Infinitely (without timing out) */
{
    Ticks /= portTICK_RATE_MS;
}
if (UINT32_C(0) == Ticks) /* ticks cannot be 0 in FreeRTOS timer. So ticks is
assigned to 1 */
{
    Ticks = UINT32_C(1);
}
xTimerStop(timerHandle, Ticks);
```

When using the FreeRTOS API to defer from ISR using pend timer, refer to the example below.

Code 2. Example implementation for using pend timer called from an ISR

```
portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
if (xTimerPendFunctionCallFromISR(appDeferredISRCallback, NULL, UINT8_C(0),
&xHigherPriorityTaskWoken) == pdPASS){
    returnValue = INT32_C(0);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
else{
    Retcode_raiseError(returnValue);
}
```

4.3.3 Tasks

Table 6. Operating task function changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above	Remarks
RTOS Task Creation	`OS_taskCreate((OS_taskCode_tp) AppHandler, (const int8_t *) "Timername", 1024, (uint32_t) 2, &TaskHandle)`	`xTaskCreate(AppHandler, (const char * const ) "Timername", 1024, NULL, (uint32_t) 2, &TaskHandle)`	
Return type of Task create	`OS_taskHandle_tp`	`xTaskHandle`	
Task Suspend	`OS_taskSuspend(&TaskHandle)`	`vTaskSuspend(&TaskHandle)`	
Task Delay	`OS_taskDelay(1)`	`TaskDelay((portTickType) 1 / portTICK_RATE_MS)`	In XDK 1.3.0 the RTOS ticks is converted as part of abstraction
Task Delay until	`OS_taskDelayUntil(&timeToDelayTask, OS_getMsDelayTimeInSystemTicks(100))`	`vTaskDelayUntil(&timeToDelayTask, PowerMgt_GetMsDelayTimeInSystemTicks(100))`	In XDK 1.3.0 to get time in milliseconds OS_getMsDelayTimeInSystemTicks() is used, but in XDK 1.5.2 PowerMgt_GetMsDelayTimeInSystemTicks() present in Platform/Power package is used

Below is an example for *Task Delay until*.

Code 3. Example implementation for delaying an operating task using vTaskDelayUntil()

```
#include "BCDS_PowerMgt.h"
#include <FreeRTOS.h>
#include <task.h>

static uint32_t timeToDelayTask = UINT32_C(0); /* variable to delay OS task */

void Task_mainProcess(void *pvParameters){
    timeToDelayTask = xTaskGetTickCount();
    for (;;){
    vTaskDelayUntil(&timeToDelayTask,PowerMgt_GetMsDelayTimeInSystemTicks(100));
    }
}
```

4.3.4 Semaphores

Table 7. Semaphore function changes from XDK-Workbench 1.3.0 to 1.5.2 and above

Description	XDK-1.3.0	XDK-1.5.2 & above
RTOS Semaphore Handle	OS_semaphoreHandle_tp	SemaphoreHandle_t
RTOS Semaphore Create	OS_semaphoreCreate()	xSemaphoreCreateMutex()
RTOS Semaphore Take	OS_semaphoreTake(*handle, OS_MAX_DELAY)	*xSemaphoreTake(*handle, UINT32_MAX)
RTOS Semaphore Give	OS_semaphoreGive(*handle)	xSemaphoreGive(*handle)

4.4 Platform

4.4.1 Sensors

All sensor API has been changed to unify return codes and the way functions are called. The new sensor API is presented in the current version of the Sensor Guide. Module names have been renamed as well.

As an example, see the following table of changes for the Accelerometer.

Table 8. Accelerometer function changes from XDK-Workbench 1.3.0 to 1.5.2 and above

XDK-1.3.0	XDK-1.5.2 & above
Module name: "accelerometer.h"	Module name: "BCDS_Accelerometer.h"
`sensor_errorType_t accelerometerInit(AccelerometerHandle_t )`	`Retcode_T Accelerometer_init(Accelerometer_HandlePtr_T)`
`sensor_errorType_t accelerometerSetBandwidth(Accelerometer Handle_t, accelerometer_bandwidth_t)`	`Retcode_T Accelerometer_setBandwidth(Accelerometer_HandleP tr_T,Accelerometer_Bandwidth_T)`
`sensor_errorType_t accelerometerGetBandwidth(Accelerometer Handle_t, accelerometer_bandwidth_tp)`	`Retcode_T Accelerometer_getBandwidth(Accelerometer_HandleP tr_T, Accelerometer_BandwidthPtr_T)`
`sensor_errorType_t accelerometerSetRange(AccelerometerHand le_t, accelerometer_range_t)`	`Retcode_T Accelerometer_setRange(Accelerometer_HandlePtr_T , Accelerometer_Range_T)`

XDK-1.3.0	XDK-1.5.2 & above
sensor_errorType_t accelerometerGetRange(AccelerometerHandle_t, accelerometer_range_tp)	Retcode_T Accelerometer_getRange(Accelerometer_HandlePtr_T, Accelerometer_RangePtr_T)
sensor_errorType_t accelerometerSetSleepDuration(AccelerometerHandle_t, accelerometer_sleepduration_t)	Retcode_T Accelerometer_setSleepDuration(Accelerometer_HandlePtr_T, Accelerometer_SleepDuration_T)
Not Available	Retcode_T Accelerometer_getSleepDuration(Accelerometer_HandlePtr_T, Accelerometer_SleepDurationPtr_T)
sensor_errorType_t accelerometerGetRange(AccelerometerHandle_t, accelerometer_range_tp)	Retcode_T Accelerometer_getRange(Accelerometer_HandlePtr_T, Accelerometer_RangePtr_T)
sensor_errorType_t accelerometerSetSleepDuration(AccelerometerHandle_t, accelerometer_sleepduration_t)	Retcode_T Accelerometer_setSleepDuration(Accelerometer_HandlePtr_T, Accelerometer_SleepDuration_T)
Not Available	Retcode_T Accelerometer_getSleepDuration(Accelerometer_HandlePtr_T, Accelerometer_SleepDurationPtr_T)
sensor_errorType_t accelerometerSetMode(AccelerometerHandle_t, accelerometer_powermode_t)	Retcode_T Accelerometer_setMode(Accelerometer_HandlePtr_T, Accelerometer_Powermode_T)
sensor_errorType_t accelerometerGetMode(AccelerometerHandle_t, accelerometer_powermode_tp)	Retcode_T Accelerometer_getMode(Accelerometer_HandlePtr_T, Accelerometer_PowermodePtr_T)
sensor_errorType_t accelerometerReadXyzLsbValue(AccelerometerHandle_t, accelerometer_xyzData_tp)	Retcode_T Accelerometer_readXyzLsbValue(Accelerometer_HandlePtr_T, Accelerometer_XyzDataPtr_T)
sensor_errorType_t accelerometerReadXyzGValue(AccelerometerHandle_t, accelerometer_xyzData_tp)	Retcode_T Accelerometer_readXyzGValue(Accelerometer_HandlePtr_T, Accelerometer_XyzDataPtr_T)
Not Available	Retcode_T Accelerometer_deInit(Accelerometer_HandlePtr_T)

XDK-1.3.0	XDK-1.5.2 & above
sensor_errorType_t accelerometerConfigureSlopeInterrupt(AccelerometerHandle_t, accelerometer_ConfigureSlopeInterrupt_t)	Retcode_T Accelerometer_configInterrupt(Accelerometer_HandlePtr_T, Accelerometer_InterruptChannel_T, Accelerometer_InterruptType_T, void *)
Not Available	Retcode_T Accelerometer_disableInterrupt(Accelerometer_HandlePtr_T, Accelerometer_InterruptChannel_T, Accelerometer_InterruptType_T)
sensor_errorType_t accelerometerRegisterRealTimeCallback(AccelerometerHandle_t,interruptCallback)	Retcode_T Accelerometer_regRealTimeCallback(Accelerometer_HandlePtr_T, Accelerometer_InterruptChannel_T, accelRealTimeCallback)
sensor_errorType_t accelerometerRegisterDeferredCallback(AccelerometerHandle_t,deferredInterruptCallback)	Retcode_T Accelerometer_regDeferredCallback(Accelerometer_HandlePtr_T, Accelerometer_InterruptChannel_T, accelDeferredTimeCallback)
sensor_errorType_t accelerometerUnregisterCallback(AccelerometerHandle_t)	Not Available

4.4.2 SDC (SD Card Driver)

- (C) The module SDC has been renamed to SDCardDriver
- (C) API function names have been changed to reflect the new module name
- (C) Return Types, Input Types and Enumerations have been changed for more uniformity
- (C) The interface is now provided by "BCDS_SDCardDriver.h" (previously "`SDC_sdCardDriver_ih.h`")

See the complete list of changes for further information:

Table 9. SD card driver function changes from XDK-Workbench 1.3.0 to 1.5.2 and above

XDK-1.3.0	XDK-1.5.2 & above
`SDC_return_t SDC_init(void)`	`Retcode_T SDCardDriver_Init(void)`
`SDC_Status_t SDC_getSdCardDetectStatus(void)`	`SDCardDriver_Status_T SDCardDriver_GetDetectStatus(void)`
`void SDC_sdCardDisconnect(void)`	`void SDCardDriver_Disconnect(void)`
`SDC_return_t SDC_sdCardConnect(void)`	`Retcode_T SDCardDriver_Connect(void)`
`SDC_diskStatus_t SDC_diskInitialize(uint8_t sdDrive)`	`Retcode_T SDCardDriver_DiskInitialize(uint8_t sdDrive)`
`SDC_diskResult_t SDC_sdCardDiskWrite(uint8_t drive, const uint8_t *writeBuffer,uint32_t sector, uint8_t writeCount)`	`Retcode_T SDCardDriver_DiskWrite(uint8_t drive, const uint8_t *writeBuffer,uint32_t sector, uint32_t writeCount)`
`SDC_diskResult_t SDC_sdCardDiskRead(uint8_t drive, uint8_t *readBuffer,uint32_t sector, uint8_t readCount)`	`Retcode_T SDCardDriver_DiskRead(uint8_t drive, uint8_t *readBuffer,uint32_t sector, uint32_t readCount)`
`SDC_diskResult_t SDC_diskIoctl(uint8_t drive, uint8_t control, void *buffer)`	`Retcode_T SDCardDriver_DiskIoctl(uint8_t drive, uint8_t control, void *buffer)`
`SDC_diskStatus_t SDC_getDiskStatus(uint8_t drive)`	`Retcode_T SDCardDriver_GetDiskStatus(uint8_t drive)`

4.4.3 FatFs

- (C) changed function signature `FRESULT f_mount (BYTE, FATFS*)` to `FRESULT f_mount (FATFS* fs, const TCHAR* path, BYTE opt)`

4.4.4 Workbench-Examples

All the examples have been renamed and substantially changed in accordance with the new startup procedure of applications and the respective API changes.

5. Version 1.6.0 to 1.7.0

In the following text, abbreviations are used for breaking changes, new features, and bugfixes as follows.

- (C) - Breaking Change
- (F) - New Feature
- (B) - Bugfixes

5.1 Platform

5.1.1 WLI/ NCI (WIFI and Networking)

- (C) The modules WLI and NCI have been renamed to WlanConnect and NetworkConfig respectively
- (C) API function names have been changed to reflect the new module names
- (C) Return Types, Input Types and Enumerations have been changed for more uniformity

See the complete list of changes for further information:

Table 10. Wi-Fi function changes from XDK-Workbench 1.6.0 to 1.7.0 and above

XDK-1.6.0	XDK-1.7.0
`WLI_return_t WLI_init(void)`	`Retcode_T WlanConnect_Init(void);`
`WLI_deinit WLI_deinit(void)`	`Retcode_T WlanConnect_DeInit(void);`
`WLI_return_t WLI_connectOpen(WLI_connectSSID_t connectSSID,WLI_connectCallback_t connectCallback);`	`Retcode_T WlanConnect_Open(WlanConnect_SSID_T connectSSID, WlanConnect_Callback_T connectCallback)`
`WLI_return_t WLI_connectWEP_Open(WLI_connectSSID_t connectSSID,WLI_connectPassPhrase_t connectPass,WLI_connectCallback_t connectCallback);`	`Retcode_T WlanConnect_WEP_Open(WlanConnect_SSID_T connectSSID, WlanConnect_PassPhrase_T connectPass,uint8_t passPhraseLength, WlanConnect_Callback_T connectCallback);`
`WLI_return_t WLI_connectWPA(WLI_connectSSID_t connectSSID, WLI_connectPassPhrase_t connectPass,WLI_connectCallback_t connectCallback);`	`Retcode_T WlanConnect_WPA(WlanConnect_SSID_T connectSSID,WlanConnect_PassPhrase_T connectPass,WlanConnect_Callback_T connectCallback);`
`WLI_return_t WLI_connectWPS_PBC(WLI_connectCallback_t connectCallback);`	`Retcode_T WlanConnect_WPS_PBC(WlanConnect_Callback_T connectCallback);`

XDK-1.6.0	XDK-1.7.0
`WLI_return_t WLI_connectWPS_PIN(WLI_connectCallback_t connectCallback);`	`Retcode_T WlanConnect_WPS_PIN(WlanConnect_Callback_T connectCallback);`
`WLI_return_t WLI_deleteAllProfiles(void);`	`Retcode_T WlanConnect_DeleteAllProfiles(void);`
`WLI_return_t WLI_disconnect(WLI_disconnectCallback_t disconnectCallback);`	`Retcode_T WlanConnect_Disconnect(WlanConnect_DisconnectCallback_T disconnectCallback);`
`WLI_scanReturnCode_t WLI_scanNetworks(WLI_scanInterval_t f_scanInterval, WLI_scanList_t* f_scanList);`	`Retcode_T WlanConnect_ScanNetworks(WlanConnect_ScanInterval_T scanInterval, WlanConnect_ScanList_T* scanList);`
`WLI_currentStatus_t WLI_getCurrentNetworkStatus(WLI_disconnectCallback_t allDisconnectCallback);`	`WlanConnect_CurrentNwStatus_T WlanConnect_GetCurrentNwStatus(void);`
`WLI_connectStatus_t WLI_getConnectionStatus(void);`	`WlanConnect_Status_T WlanConnect_GetStatus(void);`
`NCI_return_t NCI_getIpSettings(NCI_ipSettings_t* myIpSettings);`	`Retcode_T NetworkConfig_GetIpSettings(NetworkConfig_IpSettings_T* myIpSettings);`
`NCI_return_t NCI_setIpStatic(NCI_ipSettings_t myIpSettings);`	`Retcode_T NetworkConfig_SetIpStatic(NetworkConfig_IpSettings_T myIpSettings);`
`NCI_return_t NCI_setIpDhcp(NCI_ipCallback_t myIpCallback);`	`Retcode_T NetworkConfig_SetIpDhcp(NetworkConfig_IpCallback_T myIpCallback);`
`uint32_t NCI_ipv4Value(uint32_t add3, uint32_t add2, uint32_t add1, uint32_t add0);`	`uint32_t NetworkConfig_Ipv4Value(uint8_t add3, uint8_t add2, uint8_t add1, uint8_t add0);`
`uint32_t NCI_ipv4Byte(uint32_t ipValue, uint8_t index);`	`uint8_t NetworkConfig_Ipv4Byte(uint32_t ipValue, uint8_t index);`
`NCI_ipStatus_t NCI_getIpObtainedStatus(void);`	`NetworkConfig_IpStatus_T NetworkConfig_GetIpStatus(void);`

6. Version 2.0.1 to 3.0.0

In the following text, abbreviations are used for breaking changes, new features, and bugfixes as follows.

- (C) - Breaking Change
- (F) - New Feature
- (B) - Bugfixes

6.1 Basic

The way the application is started on the XDK has changed significantly, from version 2.0.1 to 3.0.0. In that regard, changes have to be made to the implementation file which declares the function `appInitSystem()` as well as the entire content of *main.c*, which is automatically generated in every XDK Project.

An easy way to solve this, would be by creating a new project in the new Workbench version and to copy only the code written by the user.

Alternatively, the changes can be made manually, as follows.

First, adapt the declaration of `appinitSystem()` as shown in Code 4 (old) and Code 5 (new)

Code 4. appInitSystem function signature of XDK-Workbench 2.0.1 and below

```
void appInitSystem(xTimerHandle xTimer){
   (void) (xTimer);
   // Your Application Code
}
```

Code 5. appInitSystem function signature of XDK-Workbench 3.0.0 and above

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

The function `appInitSystem()` was called within the context of a timer before. This has been changed in favor of a new feature called *Command Processor*, which is essentially a combination of tasks and queues. More information regarding Command Processors can be found in the FreeRTOS guide.

As a result, the main-function within the implementation file *main.c* has changed to accommodate that. As such, the content of *main.c* has to be adapted as seen in Code 6 (old) and Code 7 (new).

Code 6. Main.c implementation of XDK-Workbench 2.0.1 and below

```
/* basic header files */
#include "BCDS_Basics.h"

/* additional interface header files */
#include "XdkSystemStartup.h"

/* functions */
int main(void){
    systemStartup();
}
```

Code 7a. Main.c implementation of XDK-Workbench 3.0.0 and above (Part 1)

```
#include "stdio.h"

/* system header files */
#include <BCDS_Basics.h>
#include "BCDS_CmdProcessor.h"
#include "BCDS_Assert.h"
#include "FreeRTOS.h"
#include "task.h"
#include "HAL_UART.h"

/* additional interface header files */
#include "XdkSystemStartup.h"

/* global variables ********************************************************** */

#define TASK_PRIO_MAIN_CMD_PROCESSOR                (UINT32_C(1))
#define TASK_STACK_SIZE_MAIN_CMD_PROCESSOR          (UINT16_C(700))
#define TASK_Q_LEN_MAIN_CMD_PROCESSOR               (UINT32_C(10))

void appInitSystem(void * CmdProcessorHandle, uint32_t param2);
static CmdProcessor_T MainCmdProcessor;
```

Code 7b. Main.c implementation of XDK-Workbench 3.0.0 and above (Part 2)

```
#include "stdio.h"

/* system header files */
#include <BCDS_Basics.h>
#include "BCDS_CmdProcessor.h"
#include "BCDS_Assert.h"
#include "FreeRTOS.h"
#include "task.h"
#include "HAL_UART.h"

/* additional interface header files */
#include "XdkSystemStartup.h"

/* global variables ********************************************************** */

#define TASK_PRIO_MAIN_CMD_PROCESSOR                UINT32_C(1)
#define TASK_STACK_SIZE_MAIN_CMD_PROCESSOR          UINT16_C(700)
#define TASK_Q_LEN_MAIN_CMD_PROCESSOR               UINT32_C(10)

void appInitSystem(void * CmdProcessorHandle, uint32_t param2);
static CmdProcessor_T MainCmdProcessor;
```

The three defines control how the Main Command Processor is initialized. They determine the underlying task's priority, its stack size, and how many functions can be in queue at the same time. These defintions should be changed according to the user's requirements.

Code 8. Command processor specific configuration parameters

```
#define TASK_PRIO_MAIN_CMD_PROCESSOR                UINT32_C(1)
#define TASK_STACK_SIZE_MAIN_CMD_PROCESSOR          UINT16_C(700)
#define TASK_Q_LEN_MAIN_CMD_PROCESSOR               UINT32_C(10)
```

6.2 Common

Most Platform-specific configurations have been moved to Common/config. These configurations allow for enabling/disabling certain modules and features. Some of the modules that can be configured are:

- Platform/Drivers
- Platform/Essentials
- Platform/FOTA
- Libraries/FreeRTOS
- Libraries/FATfs

See SDK/xdk110/Common in the XDK-Workbench for more information.

6.2.1 Essentials / BSP / HAL Config

It is possible to configure which BSP modules are activated in *Common/config/Essentials/BCDS_HALConfig.h* via `#define` statements

For example:

Code 9. BSP module activation defines

```
#define BCDS_FEATURE_BSP_SD_CARD            1   /* Feature for SD CARD  */
#define BCDS_FEATURE_I2C                    1   /* Feature for I2C */
#define BCDS_FEATURE_BSP_USB                1   /* Feature for USB */
#define BCDS_FEATURE_SPI                    1   /* Feature for SPI */
#define BCDS_FEATURE_BSP_LED                1   /* Feature for LED */
#define BCDS_FEATURE_BSP_BUTTON             1   /* Feature for BUTTON */
```

6.3 Libraries

6.3.1 ServalStack

Some portions of the API may be unavailable, if it had been disabled during compilation. API is considered disabled, if the corresponding *_ENABLE_* define is set to 0 within the header-file `Serval_Defines.h`. Changing the corresponding defines does not have an effect on the availability of a module currently.

- HTTP
  - (F) Custom Headers can be serialized
  - (F) Host-Header and Range-Header can now be set

- REST
  - (F) Host-Header and Range-Header can now be set
  - (C) Host-Header and Range-Header must be explicitly set to `NULL` if not used, otherwise they may produce random header values.

- MQTT
  - (F) added MQTT module, available as of version 3.1.0

6.4 Platform

6.4.1 Module Organization

As Modularity Improvements, some modules have been reorganized.

- Platform/Essentials now contains:
  - Basics (previously Platform/Basics)
  - Control (previously Platform/Control)
- Platform/Drivers now contains:
  - MiscDrivers (previously Platform/MiscDrivers)
  - SensorDrivers (previously Platform/SensorDrivers)
  - portable and abstract API (regardless of the type of underlying hardware)
- Platform/BSP (*Board Support Package*) contains software for hardware-specific drivers.
- Platform/Peripherals removed, part of BSP and Drivers now
- Platform/Power removed, power configurations are part of the respective modules, instead of general power options.
- Platform/Utils now contains the following APIs:
  - BCDS_CmdProcessor.h
  - BCDS_EventHub.h
  - BCDS_GuardedTask.h
  - BCDS_Queue.h
  - BCDS_SleepControl.h
  - BCDS_TaskMonitor.h

6.4.2 FOTA

- (C) Introduced Event Driven Base implementation
  - (C) Renamed `FotaStateMachine_CancelDownload()` to `FotaStateMachine_ControlDownload()`
- (C) Introduced additional parameters for the following functions:
- FotaStateMachine_Init()`
FotaStateMachine_GetState()`
  * `DownloadClient_SetupNewDownload`
* (F) New function provided in validation agent to verify the integrity of the firmware

6.4.3 SD Card Driver

- API Interface has been moved to Platform/Drivers
- (C) Renamed `SDCardDriver_Init(void)` to `SDCardDriver_Initialize(void)`
- (C) Removed `SDCardDriver_Disconnect(void)`
- (F) Added `Retcode_T SDCardDriver_Deinitialize(void)`
- (C) Removed `SDCardDriver_Connect(void)`
- (C) Changed Return-Type of `SDCardDriver_GetDiskStatus(..)` from `Retcode_T` to `SDCardDriver_DiskStatus_T`

6.4.4 WLAN Driver

- (C) Removed `WlanDriver_InterruptEnable()`
- (C) Removed `WlanDriver_InterruptDisable()`

6.4.5 BLE

- (C) The entire BLE API has been replaced by a new event-driven implementation. Please refer to the BLE Guide for more information

6.4.6 Utils

- Some API from 2.0.1 Platform/Control has been moved here (see chapter Platform - Module Organization)
- (F) Added `BCDS_XProtocol.h` interface
- (C) The following interface files have been renamed:
  - `LB2C_lb2cProtocol_ih.h` to `BCDS_LeanB2CAP.h`
  - `LOG_module_ih.h` to `BCDS_Logging.h`
  - `RB_ringBuffer_ih.h` to `BCDS_RingBuffer.h`
  - `TLV_dataHandler_ih.h` to `BCDS_TLV.h`
- `DBG_assert_ih.h` has been removed.

6.5 Workbench-Examples

6.5.1 SendAccelerometerDataOverBle & SendAccelDataOverUdpAndBle

Since the entire BLE API has been replaced, it is recommended that users who created a project based on one of these examples create a new one in the workbench, while only copying the user-specific code into the new project.

For most use cases, sending and receiving are the most important functionalities for BLE. As such, the following list describes, which functions handled which functionality within the respective example.

- SendAccelerometerDataOverBle
  - Sending
    - Previously: `bleAccelDataTransmit(..)`
    - Now: `BleAccelDataTransmit(..)`
  - Receiving
    - Previously: `bleAlpwDataExchangeService(..)`
    - Now: `BleDataReceivedCallBack(..)`

- SendAccelDataOverUdpAndBle
  - Sending
    - Previously: `sendAccelData(..)`
    - Now: `SendAccelDataoverBle(..)`
  - Receiving
    - Previously: `bleAlpwDataExchangeService(..)`
    - Now: `BleDataReceivedCallBack(..)`

6.5.2 SdCardExample

See chapter 2.0.1 to 3.0.0 - Platform - SD Card Driver for relevant changes

7. Document History and Modification

REV. NO.	CHAPTER	DESCRIPTION OF MODIFICATION/CHANGES	EDITOR	DATE
1.0		Version 1.0 initial release	AFS	2018-05-11