MediaTek LinkIt™ Assist 2502 Developer's Guide

Version: 1.1

Release date: 17 June 2015

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	9 April 2015	Initial release
1.1	17 June	Update for board availability

Table of contents

Lists of tables and figures

1. Introduction

1.1. What is MediaTek LinkIt?

MediaTek LinkIt™ is a collection of development platforms designed for the prototyping of Wearables and Internet of Things (IoT) devices. Each development platforms provide a collection of tools, hardware and related resources to enable developers to address one of the three Wearables and Internet of Things (IoT) device sectors, as shown in Figure 1.

	One Application Use (OAU)	Simple Application Use (SAU)	Rich Application Use (RAU)
Examples	Fitness Tracker Health Tracker Simple Bluetooth device Smart bulbs Smart appliances	Smart Wristband Smart Watch Child/Elderly Safety watch/ wristband/tracker	High-end Smart Watch Smart Glasses
Hardware (optional)	MCU (<100 MHz) Wi-Fi or Bluetooth Sensor LED Display	MCU (100~300 MHz) Bluetooth Sensors LED or TFT Display GSM/GPRS GPS Wi-Fi	AP (> 1GHz with multi-core) Bluetooth Sensors TFT Display or See-Through Display GSM/GPRS GPS Wi-Fi
OS		Mostly use RTOS as Kernel	Mostly use branded Linux as Kernel
Price Point	Lowest	Middle	Highest
Battery life	Long (>7 days)	Medium 5-7 days	Short (2-3 days)
Characteristics	Limited computing power, focusing on 1 task (such as sports, health, find device, switching, setting) Mostly non-display or with very simple LED display	May have multiple functions and can update apps Also need outdoor/indoor positioning	Multiple apps and functions Sophisticated UI with more powerful graphics and multimedia features
	Focus segment for MediaTek LinkIt Connect 7681 development platform and the MT7681 chipset	Focus segment for MediaTek LinkIt Assist 2502 and LinkIt ONE development platforms and MT2502 chipset	

Figure 1 Wearables and IoT product segments

Each development platform turn offers one or more chipset and API variants designed to meet specific development and device requirements. To enable the creation devices prototypes, each developer platform includes:

- One or more HDKs consisting of a development board and one of more chipset modules to enable the prototyping of devices.
- An SDK to enable the creation of firmware or software for devices.
- One or more hardware reference designs that can be used as the basis for board layouts of final products.
- Comprehensive documentation, such as API references, developer guides, chipset descriptions and pin-out diagrams.
- Support forums.

1.2. MediaTek LinkIt Assist 2502 development platform

The MediaTek LinkIt Assist 2502 development platform is designed to enable the prototyping of Simple Application Use (SAU) Wearables and IoT devices. These devices include smartwatches, toddler and senior trackers, smart wrist bands and healthcare wearable devices.

As shown in Figure 2 this development platform consists of a SOC, API, hardware development kit (HDK), software development kit (SDK) and is supported by comprehensive documentation.

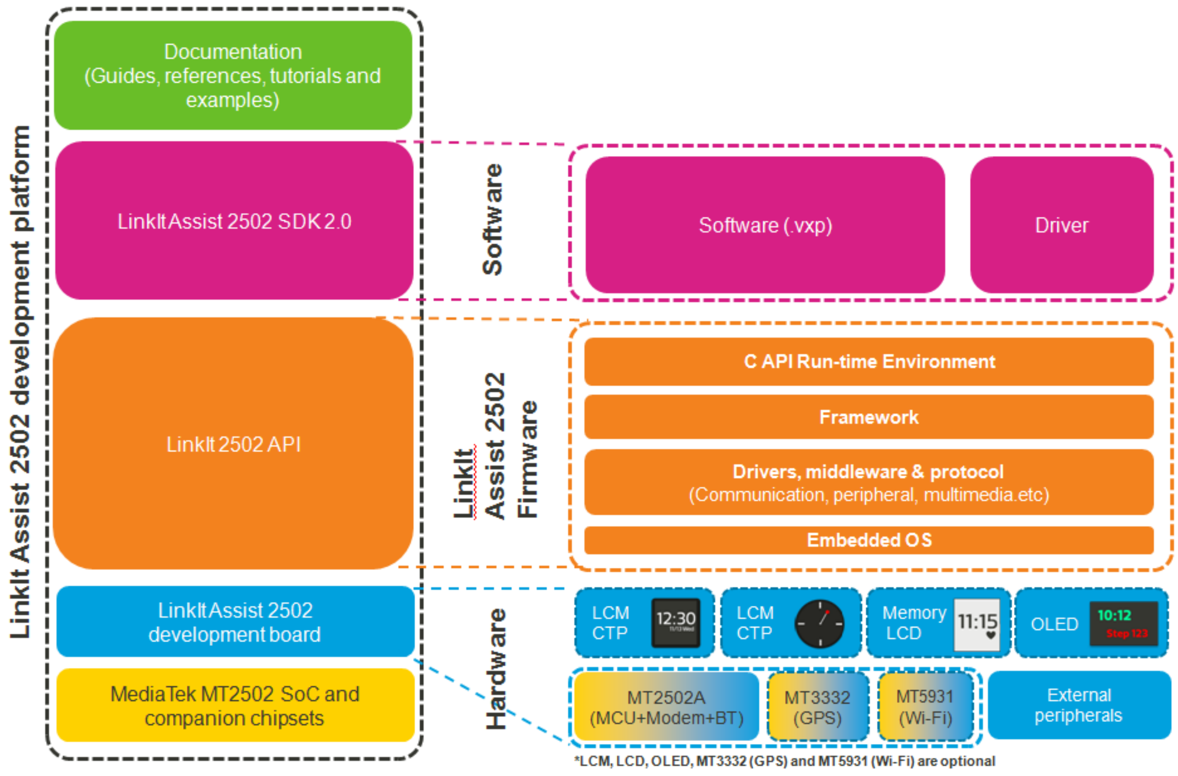

LinkIt Assist 2502 development platform
Documentation
(Guides, references, tutorials and examples)
LinkIt Assist 2502 SDK 2.0
LinkIt 2502 API
LinkIt Assist 2502 development board
MediaTek MT2502 SoC and companion chipsets
Software
LinkIt Assist 2502 Firmware
Hardware
Software (.vxp)
Driver
C API Run-time Environment
Framework
Drivers, middleware & protocol
(Communication, peripheral, multimedia.etc)
Embedded OS
LCM CTP
12:30
LCM CTP
Memory LCD
11:15
OLED
10:12
MT2502A (MCU+Modem+BT)
MT3332 (GPS)
MT5931 (Wi-Fi)
External peripherals
*LCM, LCD, OLED, MT3332 (GPS) and MT5931 (Wi-Fi) are optional

Figure 2 The components of the MediaTek LinkIt Assist 2502 development platform

1.2.1. System-on-Chip

The LinkIt development platform is built around the world's smallest commercial System-on-chip (SOC) for Wearables, MediaTek MT2502 (Aster).

This SOC also works with MediaTek's energy efficient Wi-Fi (MediaTek MT5931) and GNSS (MediaTek MT3332) companion chipsets.

1.2.2. LinkIt Assist 2502 Hardware Development Kit (HDK)

The LinkIt Assist 2502 HDK consists of:

- LinkIt Assist 2502 development board from Seeed. The board provides access to the hardware LCD engine of MediaTek MT2502 SOC, making it suitable for prototyping wearable devices.
- LinkIt Assist 2502 modules by AcSiP that includes:
  - LinkIt Assist 2502 module (AcSiP module model number AI2502S05)
  - LinkIt Assist 2502 GNSS module (AcSiP module model number CW03S)
  - LinkIt Assist 2502 Wi-Fi module (AcSiP module model number CW01S)

1.2.2.1. LinkIt Assist 2502 development board

The LinkIt Assist 2502 development board incorporates the three LinkIt Assist 2502 modules, as shown in Figure 3, and display, pins and buttons, as shown in Figure 4.

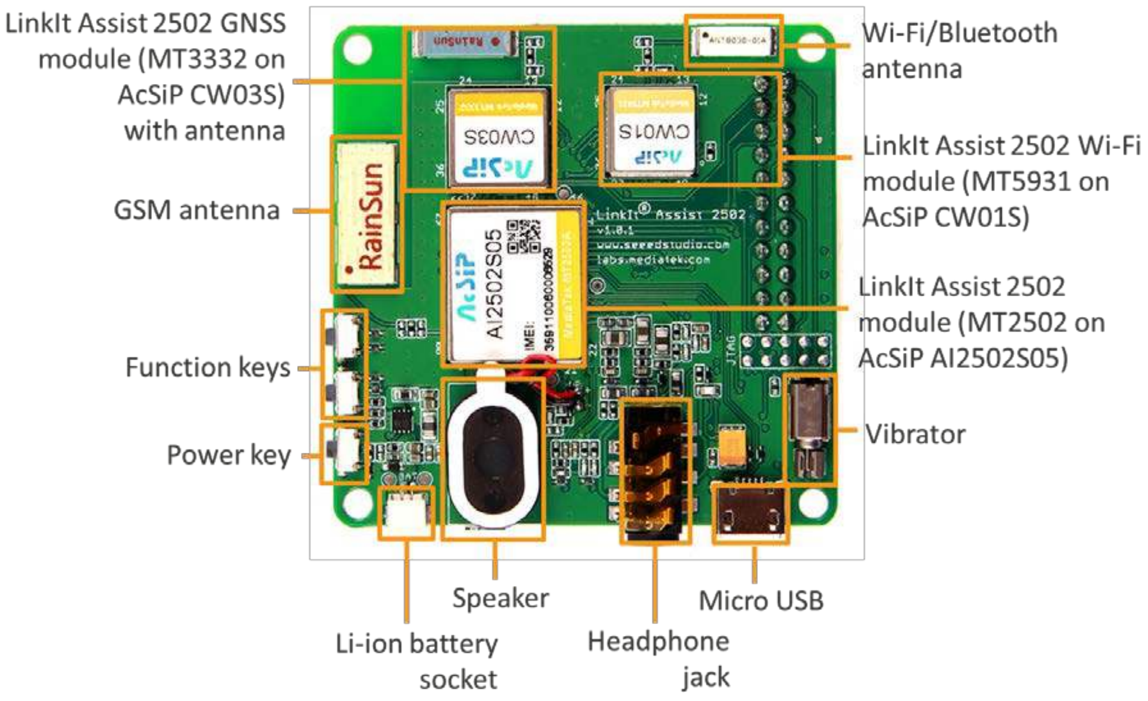

LinkIt Assist 2502 GNSS module (MT3332 on AcSiP CW03S) with antenna
GSM antenna
Function keys
Power key
Wi-Fi/Bluetooth antenna
LinkIt Assist 2502 Wi-Fi module (MT5931 on AcSiP CW01S)
LinkIt Assist 2502 module (MT2502 on AcSiP AI2502S05)
Vibrator
Speaker
Li-ion battery socket
Headphone jack
Micro USB

Figure 3 LinkIt Assist 2502 development board (back view)

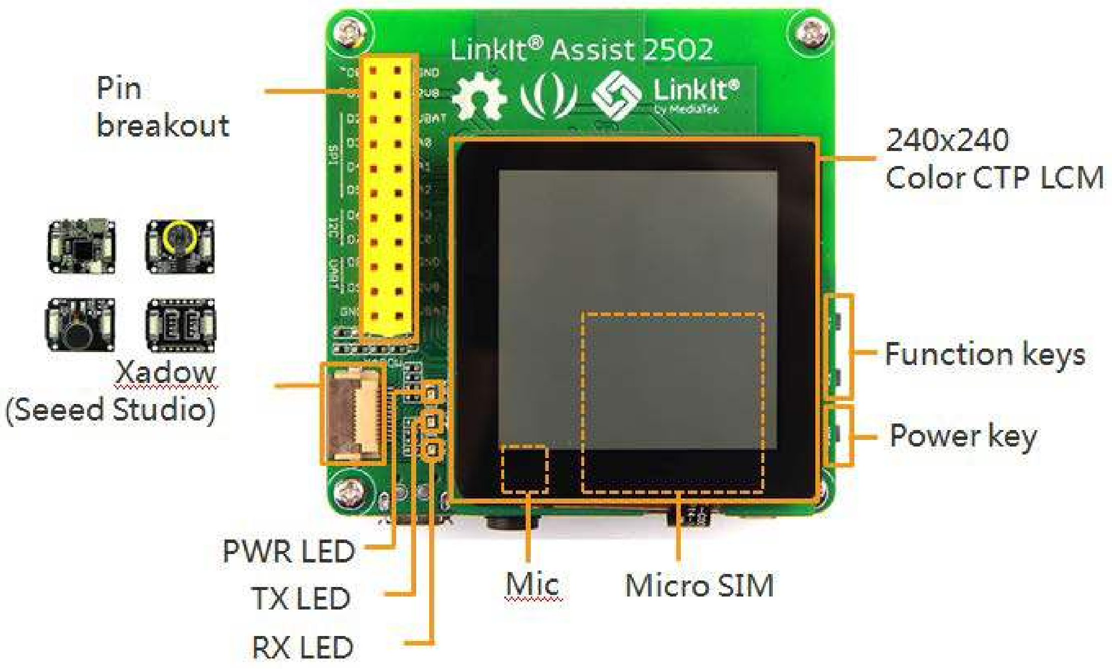
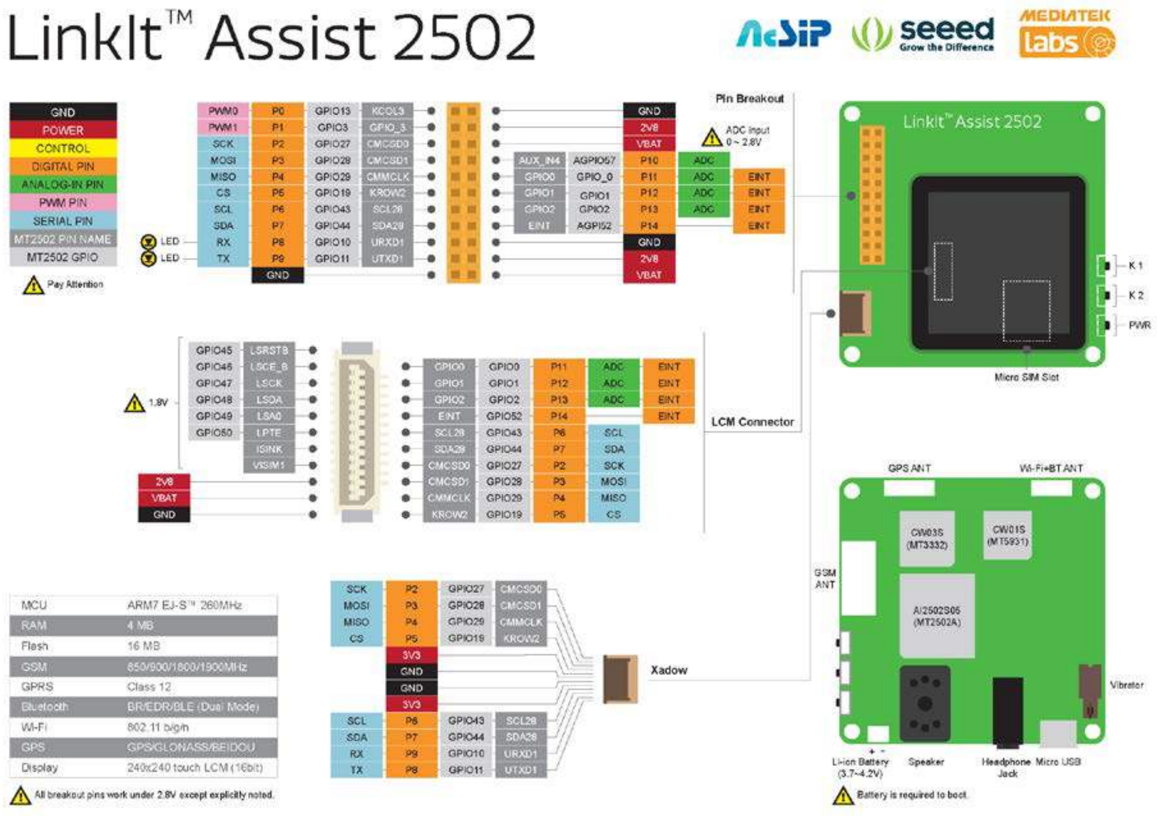


Figure 4 LinkIt Assist Development Board (front view)

Each MediaTek Assist 2502 module include a MediaTek SOC and related RF components to provide an easy-to-use hardware interface. The schematic, layout and BOM for each module are available for you to use in designing your own MT2502 product using modules. Hardware development cycle can be shortened by reference to this reference design based on modules, and make shorter time-to-market possible.

This platform provides a full-range of communication, positioning and multimedia features. This make it suitable for design and development an wide array of wearable devices such as watches, wristbands, trackers, and other mobile IoT devices. Communication protocols GSM, GPRS, Bluetooth 2.1 and 4.0 and Wi-Fi are supported. GNSS positioning feature include supports for GPS, GLONASS and BeiDou systems.

It is also easy to attach the peripheral devices that might be needed to prototype the final product through interfaces that include GPIO, analog, PMW, I2C, SPI, UART and Xadow. The available pins are illustrated in Figure 5.



MCU	ARM7 EJ-S™ 260MHz
RAM	4 MB
Flash	16 MB
GSM	850/900/1800/1900MHz
GPRS	Class 12
Bluetooth	BR/EDR/BLE (Dual Mode)
Wi-Fi	802.11 b/g/n
GPS	GPS/GLONASS/BEIDOU
Display	240x240 touch LCM (16bit)

All breakout pins work under 2.8V except explicitly noted.

Figure 5 Pin-out diagram of the LinkIt Assist 2502 development board

The full specification of the LinkIt Assist 2502 development board is provided in Table 1.

Table 1 Specification of the LinkIt Assist 2502 development board

Category	Feature	Spec
Microcontroller Module	Module Name	AcSiP AI2502S05
	Chipset	MT2502A (Aster)
	Core	ARM7 EJ-S™
	Clock Speed	260MHz
Wi-Fi Module	Module Name	AcSiP CW01S
	Chipset	MT5931
GPS Module	Module Name	AcSiP CW03S
	Chipset	MT3332
PCB Size	Dimensions	2.1 x 2.1 inches
Memory	Flash	16MB
	RAM	4MB
Power	Battery Jack Charger	3.7~4.2V Li-ion battery Micro USB
Digital IO Pins	Pin Count	14
	Voltage	2.8V
Analog Input Pins	Pin Count	4
	Voltage	0~2.8V(1)
PWM Output Pins	Pin Count	2
	Voltage	2.8V
External Interrupts	Pin Count	2
	Voltage	2.8V
I2C (master only)	Set Count	1 (SDA, SCL)
	Speed	100Kbps, 400Kbps, 3.4Mbps
SPI (master only)	Set Count	1 (MOSI, MISO, SCK)
	Speed	104Kbps~26Mbps
UART	Set Count	1 (RX, TX)
UART on USB(2)	Set Count	1
Xadow (Seeed Studio)	Set Count	1
	Voltage	Signal 2.8V, Power 3.3V
Communications	GSM	850/900/1800/1900 MHz
	GPRS	Class 12
	Bluetooth	2.1 SPP and 4.0 GATT (Dual Mode)
	Wi-Fi	802.11 b/g/n
Positioning	GNSS	GPS/GLONASS/BeiDou
Display Module	LCD	Truer 240x240 transflective
	Driving IC	ST7789S

(1) Warning: Exceeding the maximum allowed voltage damages the pin.
(2) This UART on USB is used by the Monitor tool included in the SDK to display application logs and system logs.

1.2.2.2. Related HDKs

The LinkIt ONE development board is also based on the MediaTek MT2502 SOC. It's possible to use this board to run MediaTek LinkIt Assist 2502 applications, with some limitations.

A comparison of the development boards is provided in Table 2.

	LinkIt ONE	LinkIt Assist 2502
Purpose	Proof of concept	Prototyping
LinkIt SDK 1.0 (for Arduino) support	Yes	No
LinkIt Assist 2502 SDK support	Yes	Yes
Board Pin-out Design	Arduino-style Pin Layout	Proprietary Design
Size	84 x 53mm	53 x 53mm
Pin voltage	3.3V	2.8V
Extra Peripheral Interface	Grove (Seeed Studio)	Xadow (Seeed Studio)
Build-in Key	0	2
SD Card	Yes	No
SIM	SIM	Micro SIM
Power Source	Micro USB (5V) Battery (optional)	Micro USB (5V) Battery (required)
Audio	Headphone (inc. mic)	Headphone, speaker, mic
Vibrator	No	Yes
Antenna	External	Build-in ceramic antenna

Table 2 Comparison of the LinkIt ONE and Assist 2502 development boards

For more information on the LinkIt ONE development board, please see the MediaTek LinkIt™ ONE Developer's Guide.

To be compliant with Arduino, the design of the LinkIt ONE development board includes special circuits, for example to shift pin voltage from 2.8V to 3.3V, which would add unnecessary cost if incorporated into a final product. Those circuits are therefore not included in the LinkIt Assist 2502 development board.

1.2.3. LinkIt Assist 2502 API

The LinkIt Assist 2502 API provides a collection of C-based modules that enable programmatic access to the features of the platform's SOC, companion chipsets, displays and connected peripherals. Further details on the API is provided in 5, "API Guide".

As shown in Figure 2 the LinkIt Assist 2502 API is used to create a LinkIt Assist 2502 application in the LinkIt Assist 2502 SDK. Coding is undertaken using the C programming language based API functions, as well as third party libraries and device drivers. The application is compiled into a LinkIt Assist 2502 executable file (VXP file) and executes in the runtime environment provided by the firmware of the LinkIt Assist 2502 development board. Dynamic loading is supported to run applications from flash storage.

It's possible to develop LinkIt Assist 2502 software in C++ as the toolchain supports compilation in both C and C++ files. If you want to code in C++, rename the application source code file from `*.c` to `*.cpp` after generating your project. However, note that the LinkIt Assist 2502 API provides C functions rather than C++ functions and some C++ libraries may rely on runtime functions that are not supported by the LinkIt Assist 2502 development platform.

LinkIt Assist 2502 applications are executed in a multi-threaded environment. In this environment, there are system-managed threads such as driver threads that are responsible for managing integrated hardware such as the GSM modem and Bluetooth protocol stack. A LinkIt Assist 2502 application runs on a specific thread called *main thread*. LinkIt Assist 2502 applications can also create new threads. However, most LinkIt Assist 2502 SDK 2.0 API functions can only be invoked from the main thread. Functions that are called on created threads are explained in detail in the API reference documentation.

Due to the nature of multi-thread, multi-tasking environment, LinkIt Assist 2502 applications adopt an event-based programming model. More information about this programming model is provided in 5.5.5, "Threads".

1.2.3.1. API Summary

Table 3 shows a summary of the modules available in the LinkIt 2502 API.

Table 3 Summary of LinkIt 2502 API modules

Functions	Features
Core	
Memory	Provides functions for memory allocation, deallocation and alike.
Timer	Provides three timers: precise timer, non-precise timer and HISR timer.
Thread	Provides functions for threads, signals and mutex related activities.
Character Set Conversion	Character conversion functions among the character sets of ASCII, USC-2, UTF-8 and UTF-16BE.
File System	A file system that provides file and folder related operations.
Date & Time	Sets and retrieves system time, or gets the microsecond counts since the system was powered on.
Log	Outputs log items with severity to the Monitor tool provided by the SDK.
Resource	Accesses resources in VXP file, e.g. strings, image and audio data.
Firmware	API to query firmware information and trigger the firmware update process.
Power Management	Enables shut down and reboot of the system, also provides battery and charging information.

Functions	Features
User interaction	
Keypad	Registers keypad event handlers, if the HDK provides keypad hardware.
Touch	Registers touch event handlers, if supported by HDK and firmware.
Interfaces and network	
Driver Common Layer	Provides functions to control and access (read and write) to the hardware devices, including ADC, GPIO, I2C, SPI, UART, EINT, PMU and PWM.
Network Protocol	Provides TCP, UDP, SSL, HTTPS and HTTP support.
Telephony and mobile data	
SIM	Functions to access the information from the SIM card in the device.
Telephony	Functions to control voice calls.
SMS	Functions to send, read, delete and cancel SMS messages.
Cellular Info	Functions to obtain current or neighboring cell information.
GPRS	Configures the GPRS connection.
Wi-Fi	
WLAN	Configures WLAN (Wireless LAN).
Location	
GPS	GPS and Extended Prediction Orbit (EPO) functionality
Bluetooth	
Bluetooth	Functions for applications to configure, monitor and search for Bluetooth 2.1 connections. The Bluetooth SPP API supports client and server functionality for one-to-one connections.
Notification Service	Functions to receive notifications from iOS and Android devices.
Bluetooth Low Energy	Provides Bluetooth 4.0 Generic Attribute Profile (GATT) support.
Graphics	
Graphics	Performs pixel operations Draws points, lines, rectangles and ellipses Decodes and paints images Outputs text using vector font Performs LCD operations on supported hardware
Audio	
Audio	Play and record audio files

1.2.3.2. Related APIs

In addition to the core API, the LinkIt Assist 2502 development platform can also make use of:

- LinkIt Assist 2502 Smartphone Notification API, which provides for the receipt of notifications through a Bluetooth connection, including notifications from the Apple Notification Center Service (ANCS). For more information see LinkIt Assist 2502 Smartphone Notifications Developer's Guide.

- LinkIt Smart Connection libraries for Android and iOS, which provide for broadcasting wireless AP setting from a smartphone or tablet to a LinkIt 2502 development board or device. For more information see LinkIt Smart Connection Developer's Guide.

1.2.4. LinkIt Assist 2502 Software Development Kit (SDK)

The MediaTek LinkIt Assist 2502 SDK 2.0 enables the creation of application (software) for the development platform. A details guide to using the SDK is provided in 2, "Getting started".

1.2.4.1. Feature summary

The SDK consists of the following:

- **Firmware Updater** that uploads matching version of the firmware to the development boards. The SDK provides firmware in binary format.
- A plug-in for the Eclipse IDE (with CDT) that provides:
  - **Eclipse Project Launcher Wizard** that creates a project skeleton for you to start programming your LinkIt Assist 2502 applications.
  - **Eclipse Tool Bar** that defines the proper application settings and compile options for the compiler to build the VXP executable for your LinkIt Assist 2502 applications.
  - **Resource Editor** that enables you to embed images and strings into the VXP executable that can be retrieved by the Resource module of the SDK.
- **Uploader** that uploads the VXP executable to your connected development board through the USB port.

1.2.4.2. Development process overview

You start creating a LinkIt 2502 applications using the Eclipse Project Launcher Wizard to create a project skeleton or open an example project. The application is codes with the Eclipse CDT tools and the compiled and linked from options on the Eclipse Tool Bar.

The generated executable (VXP) is uploaded to the internal storage of the development board with the Uploader tool. Next, the firmware on the development board reboots, and the uploaded application executes to enable access to the features of the underlying frameworks and built-in drivers.

1.2.5. Documentation and code examples

There are five references available to assist with the development of software for LinkIt Assist 2502 prototypes:

- This developer's guide, the latest copy of which is available here on the MediaTek Labs website.
- LinkIt Smart Connection Developer's Guide: This document provides information on how to use Smart Connection to provision the Wi-Fi connection on a LinkIt development board and connect to a wireless AP. It includes details on how to use the Android and iOS libraries to build a smartphone app that is used to define an AP and send details to a development board.
- LinkIt Assist 2502 Smartphone Notifications Developer's Guide: This document provides details of the API that can be used to receive smartphone notification on a LinkIt Assist 2502 device.

- LinkIt Assist 2502 development board pin-out diagram: This diagram provides details of the pin breakout on the development board.
- LinkIt Assist 2502 API reference: Details of the various modules, their functions and parameters, available for coding software for the development platform.
- Example code supplied as part of the SDK and accessible through the Eclipse Project Wizard.

In addition there are a number of resources available to assist with the creation of final device hardware boards:

- LinkIt Assist 2502 Hardware Reference Design: This file includes:
    - LinkIt Assist 2502 development board schematic and layout
    - LinkIt Assist 2502 development board pin-out diagram
    - LinkIt Assist 2502 pin mux table
    - LinkIt Assist 2502 module data sheets (AcSiP)
    - Chipset datasheets

      Please note: PCB layout and board schematics have been created using Eagle. They can be viewed using the freeware Eagle Light Edition.
- MT2502 (Aster) chipset technical brief
- MT3332 (GNSS) chipset technical brief
- MT5931 (Wi-Fi) chipset technical brief
- LinkIt Assist 2502 module data sheet (AcSiP AI2502S05)
- LinkIt Assist 2502 GNSS module data sheet (AcSiP CW03S)
- LinkIt Assist 2502 Wi-Fi module data sheet (AcSiP CW01S)

Additional documentation may become available from time to time and can be found on the development platforms documentation page on the MediaTek Labs website.

1.3. Related development platforms

The MediaTek LinkIt ONE development platform is also based on the MediaTek MT2502 chipset. However, LinkIt ONE development platform provides for use of a set of APIs similar to those provided on the Arduino development boards, implements through a porting layer between the Arduino API and the Runtime environment. Software development is enabled by an SDK that offers a plug-in to the Arduino software.

There is also a fundamental between the software architecture of LinkIt ONE application and those for LinkIt Assist 2502. LinkIt ONE programs consists of `setup()` and `loop()` functions and it's common for the system to be in an endless polling state in the `loop()` function. This programming style is very straightforward, but has the disadvantage that it keeps the system in a busy state for much of the time, which leads to increased power consumption. The LinkIt Assist 2502 programming style however, uses event-based programming that uses power more efficiently. Please see 4.3, "Event-driven Programming Model" for details on the event-based programming model

The LinkIt ONE development board offers features compatible with the LinkIt ONE API. Software created for the LinkIt Assist 2502 development board can be run on the LinkIt ONE development board, however some features may not work due to differences in the hardware capabilities.

For information on the LinkIt ONE development platform, please see the MediaTek LinkIt™ ONE Developer's Guide.

1.4. Joining the MediaTek Labs Ecosystem

Wearables and Internet of Things are the next wave in the consumer gadget revolution. MediaTek Labs is a key player in this field, combining the best of two worlds — the existing MediaTek ecosystem of phone manufacturers, electronic device manufacturers and telecom operators combined with an open, vibrant maker and developer community.

No matter whether you're a maker, device manufacturer, student, DIY hobbyist, or programmer, you will find a development platform from MediaTek to match your requirements to create something innovative. Then you'll be able to take advantage of the MediaTek Labs Partner Connect program to take your idea to market.

You can join the MediaTek Labs ecosystem by registering on labs.mediatek.com.

2. Getting started

This chapter provides a guide to getting started with the LinkIt Assist 2502 development platform and covers the following items:

- The supported environment for development.
- Installing the Eclipse IDE for C/C++ Developers.
- Installing and configuring the LinkIt Assist 2502 SDK 2.0.
- Building your first project and running it on the development board.
- Exploring example code to accomplish specific tasks.
- Using other tools in the SDK.

2.1. Environment

To make use of the MediaTek LinkIt 2502 SDK and development board please ensure you have a PC with the following:

- Operating system: Microsoft Windows XP, Vista, 7 or 8.
- Eclipse IDE: Indigo (3.7) with CDT 8.0.2.

2.2. Installing Eclipse IDE for C/C++ Developers

The LinkIt Assist 2502 SDK 2.0 includes a plug-in for Eclipse IDE and is compatible with Eclipse IDE Indigo or higher versions. Please download and install Eclipse IDE from the official Eclipse website making sure it has the C/C++ development (CDT) plug-in. The IDE installation may require an appropriate JAVA SE version to be installed on your PC.

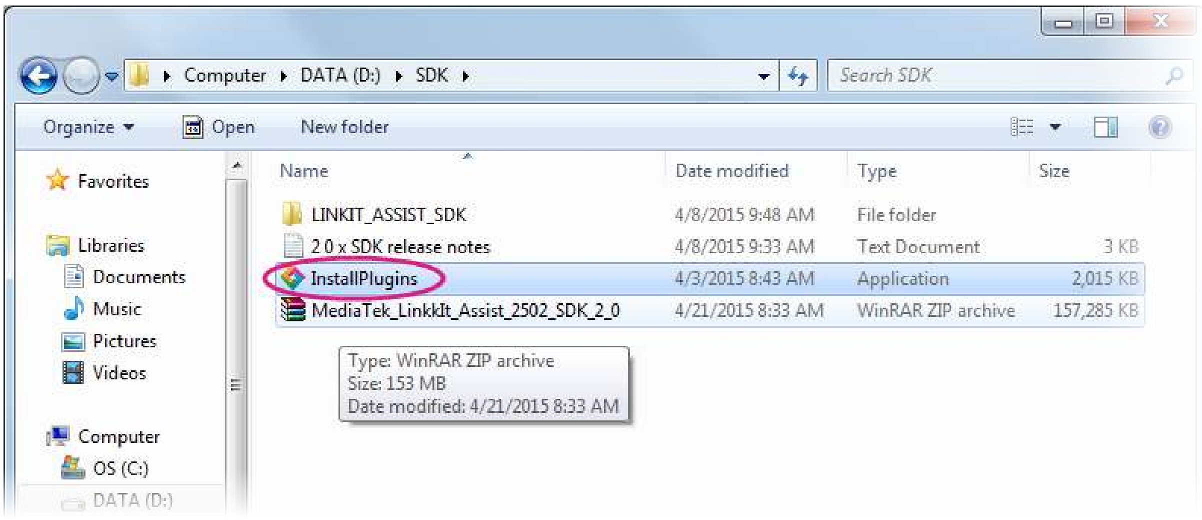
The LinkIt Assist 2502 Eclipse plug-in needs to be installed in the same folder as the Eclipse IDE, where `eclipse.exe` is located.

2.3. Installing LinkIt Assist 2502 SDK 2.0

To install LinkIt Assist 2502 SDK 2.0 you need to do the following:

1) Download the LinkIt Assist 2502 SDK 2.0
2) Extract the content of the LinkIt Assist 2502 SDK 2.0 zip file. There are tools in addition to the Eclipse plug-in included in the SDK so consider extracting the content to a permanent location. If you extract the content to a temporary location the installer will give you the option to move it to a permanent location.
3) Make sure that the Eclipse IDE is not running.

4) Double click the LinkIt Assist 2502 SDK 2.0 InstallPlugins as shown in Figure 6.

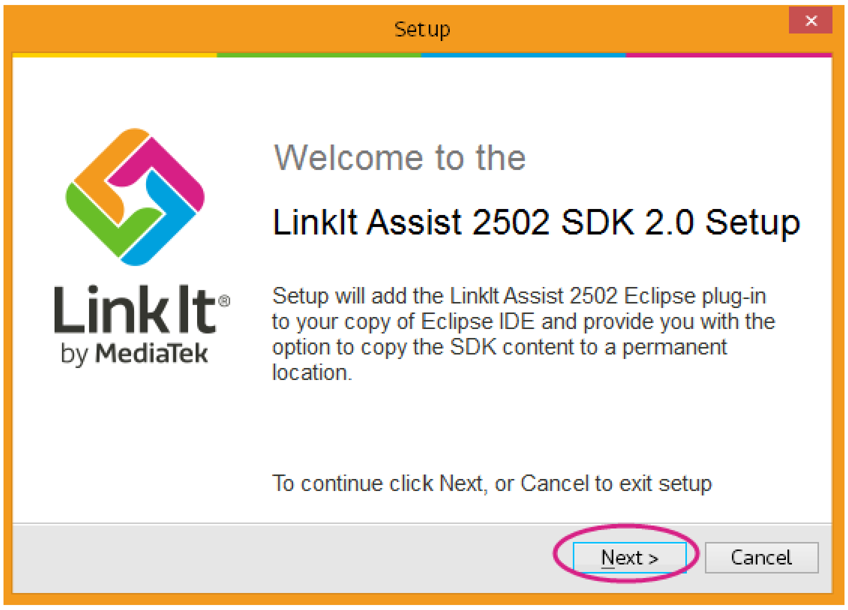


Name	Date modified	Type	Size
LINKIT_ASSIST_SDK	4/8/2015 9:48 AM	File folder	
2.0.x SDK release notes	4/8/2015 9:33 AM	Text Document	3 KB
InstallPlugins	4/3/2015 8:43 AM	Application	2,015 KB
MediaTek_LinkIt_Assist_2502_SDK_2_0	4/21/2015 8:33 AM	WinRAR ZIP archive	157,285 KB

Type: WinRAR ZIP archive
Size: 153 MB
Date modified: 4/21/2015 8:33 AM

Figure 6 The LinkIt Assist 2502 SDK 2.0 installation application

5) In the Welcome page, Figure 7, click **Next**.



Welcome to the

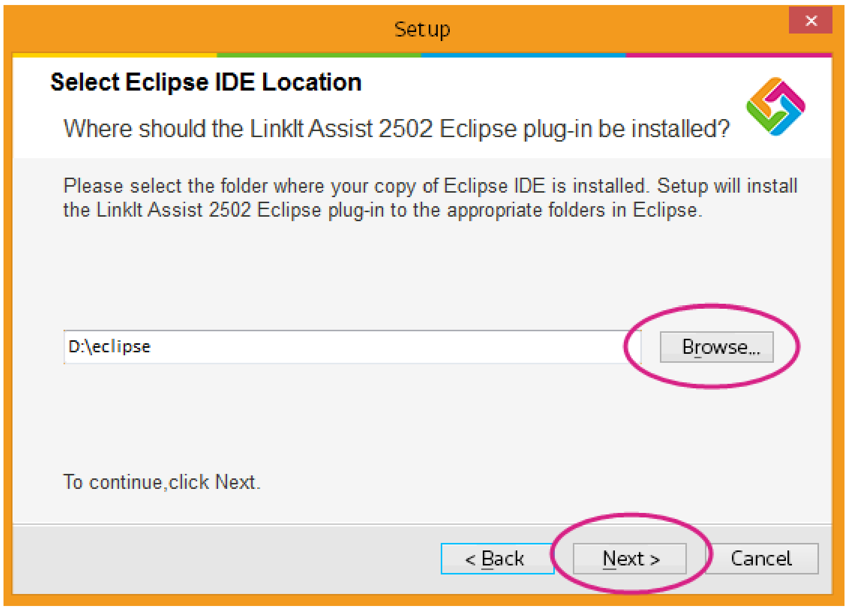
LinkIt Assist 2502 SDK 2.0 Setup

Setup will add the LinkIt Assist 2502 Eclipse plug-in to your copy of Eclipse IDE and provide you with the option to copy the SDK content to a permanent location.

To continue click Next, or Cancel to exit setup

Figure 7 The welcome page of the SDK setup wizard.

6) In **Select Eclipse IDE Location**, click **Browse** and locate the folder in which you installed Eclipse IDE (the designated folder mentioned in 2.2, "Installing Eclipse IDE for C/C++ Developers") as shown in Figure 8.


Setup

Select Eclipse IDE Location

Where should the LinkIt Assist 2502 Eclipse plug-in be installed?

Please select the folder where your copy of Eclipse IDE is installed. Setup will install the LinkIt Assist 2502 Eclipse plug-in to the appropriate folders in Eclipse.

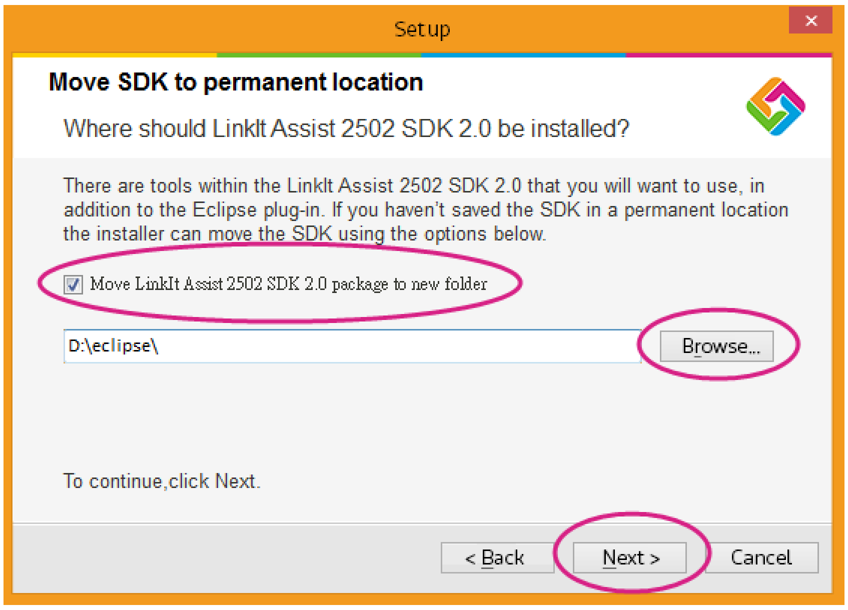
D:\eclipse

To continue,click Next.

Figure 8 Selecting the folder in which Eclipse IDE is installed

Click **Next.**

7) In **Move SDK to permanent location** check **Move LinkIt SDK 2.0 package to new folder** if you extracted the SDK to a temporary location. Click **Browse** to locate a permanent location in which to store the SDK as shown in Figure 9.


Setup

Move SDK to permanent location

Where should LinkIt Assist 2502 SDK 2.0 be installed?

There are tools within the LinkIt Assist 2502 SDK 2.0 that you will want to use, in addition to the Eclipse plug-in. If you haven't saved the SDK in a permanent location the installer can move the SDK using the options below.

[✓] Move LinkIt Assist 2502 SDK 2.0 package to new folder

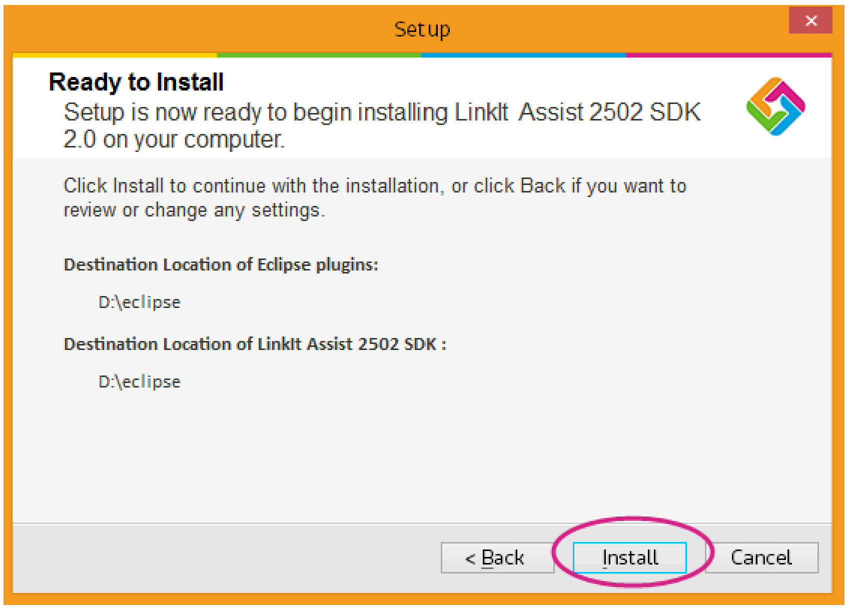
D:\eclipse\

To continue,click Next.

Figure 9 Option to move SDK to a permanent location

Click **Next.**

8) In **Ready to Install**, Figure 10, review the selected file locations and if they are OK click **Install**.



Ready to Install

Setup is now ready to begin installing LinkIt Assist 2502 SDK 2.0 on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination Location of Eclipse plugins:

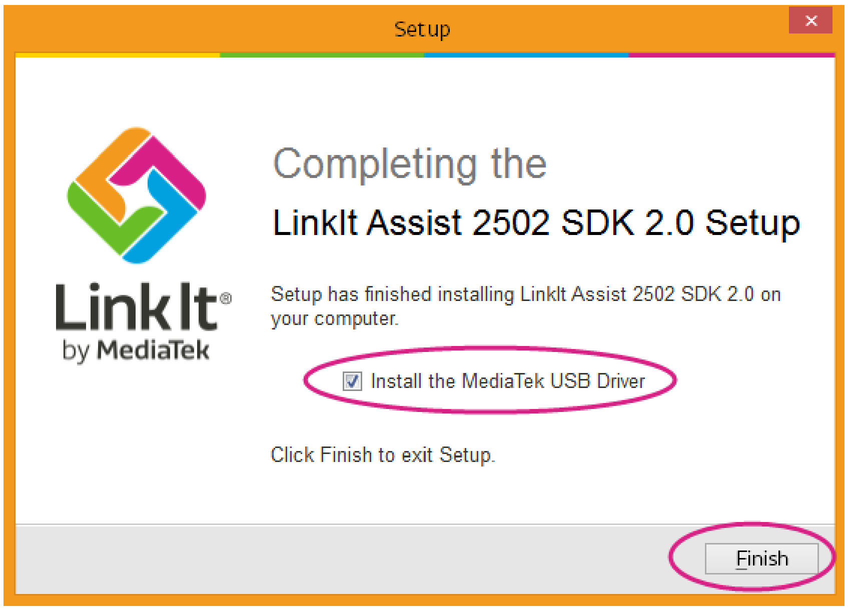
D:\eclipse

Destination Location of LinkIt Assist 2502 SDK :

D:\eclipse

Figure 10 The Ready to Install page of the setup wizard

9) Once the plug-in is installed and, if the option was taken, the SDK is moved the **Completing the LinkIt Assist 2502 SDK 2.0 Setup** page, Figure 11, displays. Ensure that **Install the MediaTek USB Driver** is checked, then click **Finish**.



Completing the LinkIt Assist 2502 SDK 2.0 Setup

Setup has finished installing LinkIt Assist 2502 SDK 2.0 on your computer.

[✓] Install the MediaTek USB Driver

Click Finish to exit Setup.

Figure 11 Final page of the SDK installer

10) The MediaTek USB driver is now installed and the setup completed.

The MediaTek USB Driver shouldn't need to be installed again should you upgrade the SDK in the future.

2.4. Updating the Firmware

It's essential to make sure that the firmware of the development board corresponds to the SDK you just installed, before you start programming applications in the Eclipse IDE. To do this you use the Firmware Updater' to update firmware after installing new versions of SDK. To get started, please make sure the following is ready:

1) If you've:
   a) A LinkIt Assist 2502 development board, you are good to go.
   b) A LinkIt ONE development board, check that it's switched into mass storage mode, (if you're using USB as the power source, make sure the USB switch is selected as well) as shown in Figure 12.

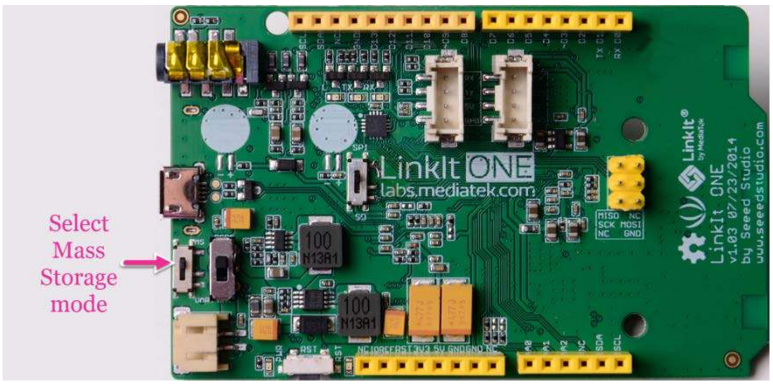
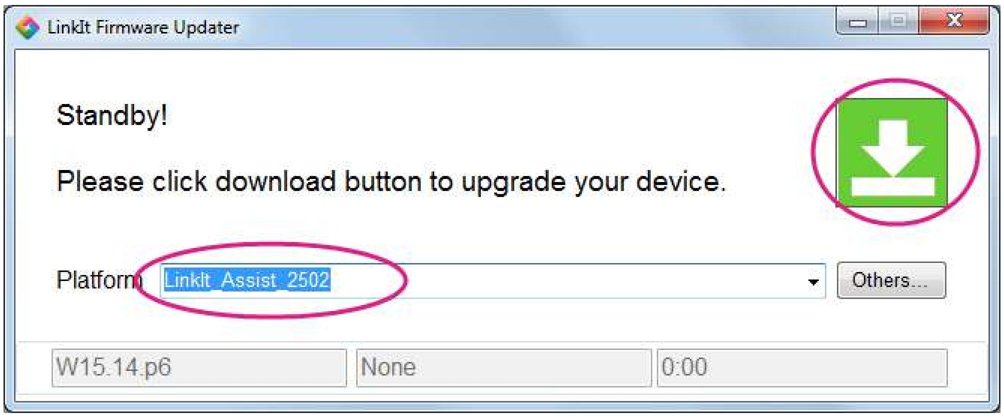


Figure 12 LinkIt ONE in mass storage mode

2) Disconnect your LinkIt development board from your PC.
3) Launch `LinkIt Firmware Updater.exe`, which is located in the SDK folder under `{SDK_path}\tools\FirmwareUpdater\`
4) In the **LinkIt Firmware Updater** window, in **Platform** select your development board. Then click the **green update** button, as shown in Figure 13.



Standby!

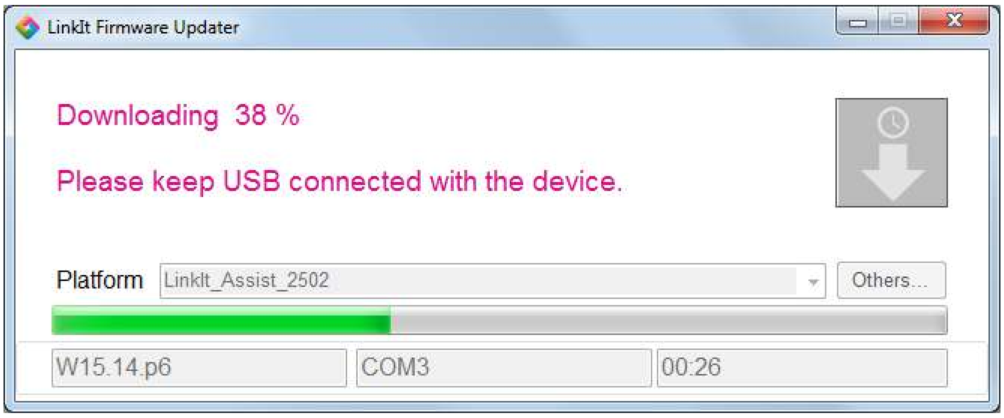
Please click download button to upgrade your device.

Platform LinkIt_Assist_2502

W15.14.p6 None 0:00

Figure 13 LinkIt Firmware Updater launch screen

5) The firmware is now downloaded to the development board, as shown Figure 14. Do not disconnect the development board from your PC during this process.


LinkIt Firmware Updater

Downloading 38 %

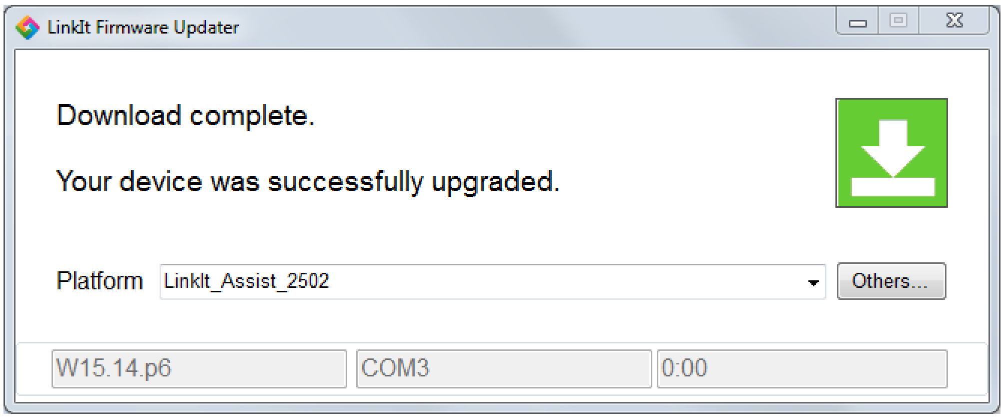
Please keep USB connected with the device.

Platform LinkIt_Assist_2502

W15.14.p6 | COM3 | 00:26

Figure 14 The firmware being uploaded to your device

6) When the firmware update is complete this will be confirmed in the **Download Complete** page, as shown in Figure 15.


LinkIt Firmware Updater

Download complete.

Your device was successfully upgraded.

Platform LinkIt_Assist_2502

W15.14.p6 | COM3 | 0:00

Figure 15 Firmware Updater Download Complete screen

7) Disconnect your LinkIt Assist 2502 development board (if you were updating a LinkIt ONE, switch it back to UART mode) and re-connect it to your PC.

With the latest SDK firmware installed, you're ready to proceed to create your first application in Eclipse IDE.

2.5. Creating Your First Project

Software development and project creation on either LinkIt ONE or LinkIt Assist 2502 development boards requires the same steps, but the results are different.

1) The first step is to create a new LinkIt Assist 2502 application. In the Eclipse IDE **File** menu, point to **New** and click **Other**, as shown in Figure 16, or use the CTRL+N keyboard shortcut.

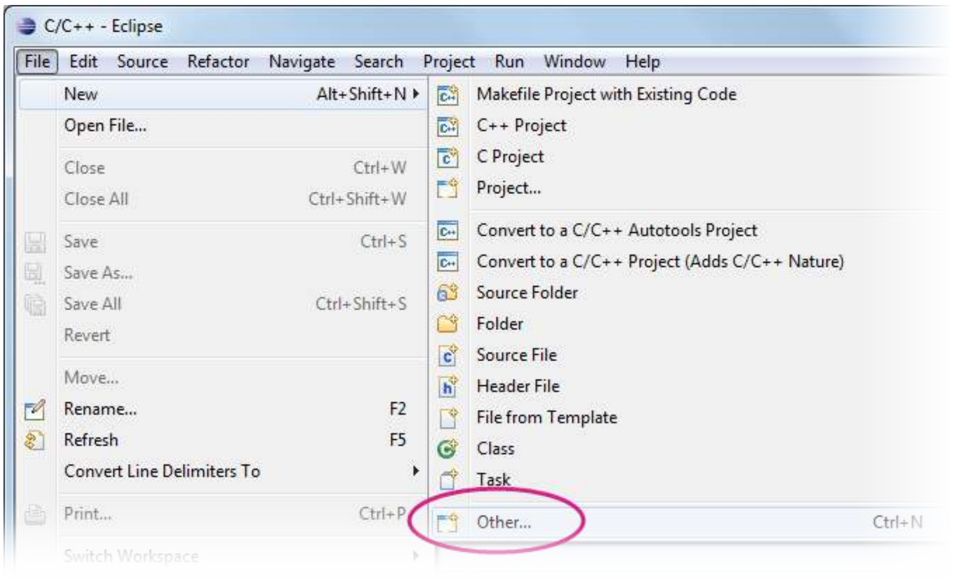


Figure 16 Creating new LinkIt Assist 2502 application

2) The **New** window displays. Expand the **LinkIt 2.0** folder, select **Application (*.vxp)** and click **Next** as shown in Figure 17.

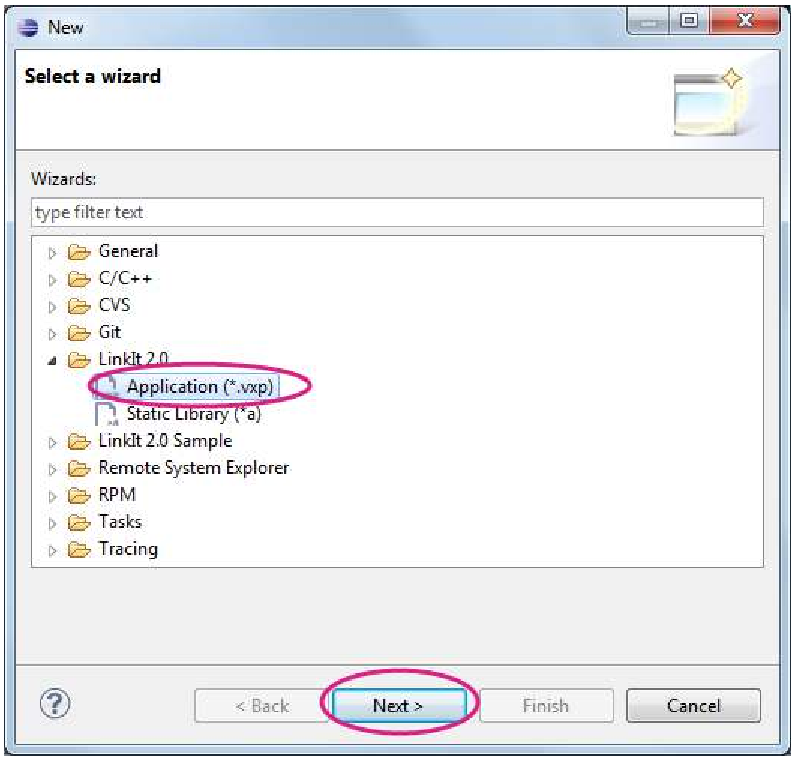
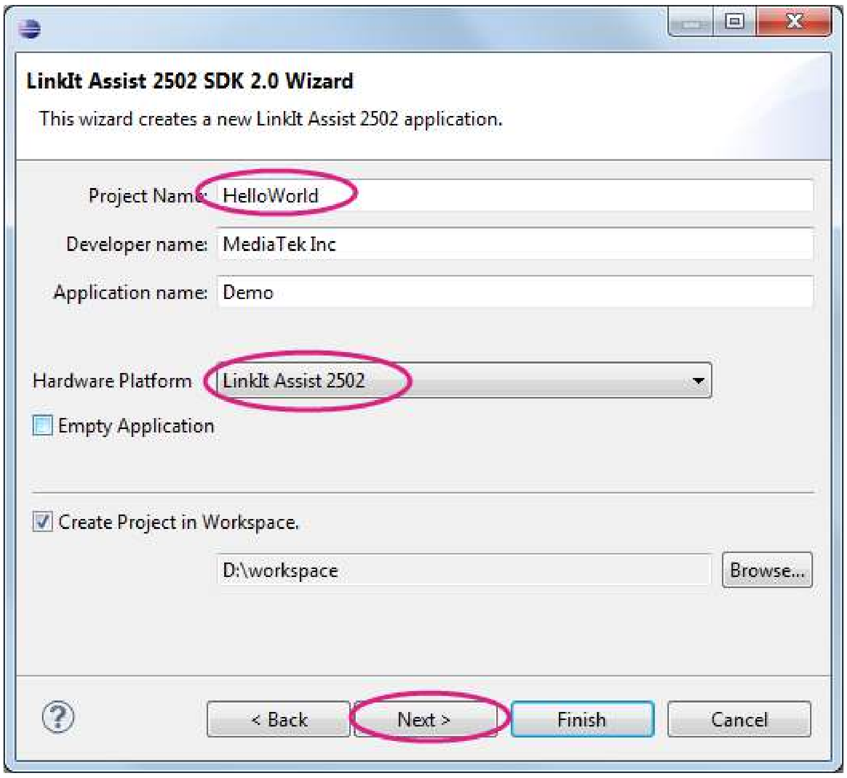


Figure 17 Selecting the LinkIt 2.0 application

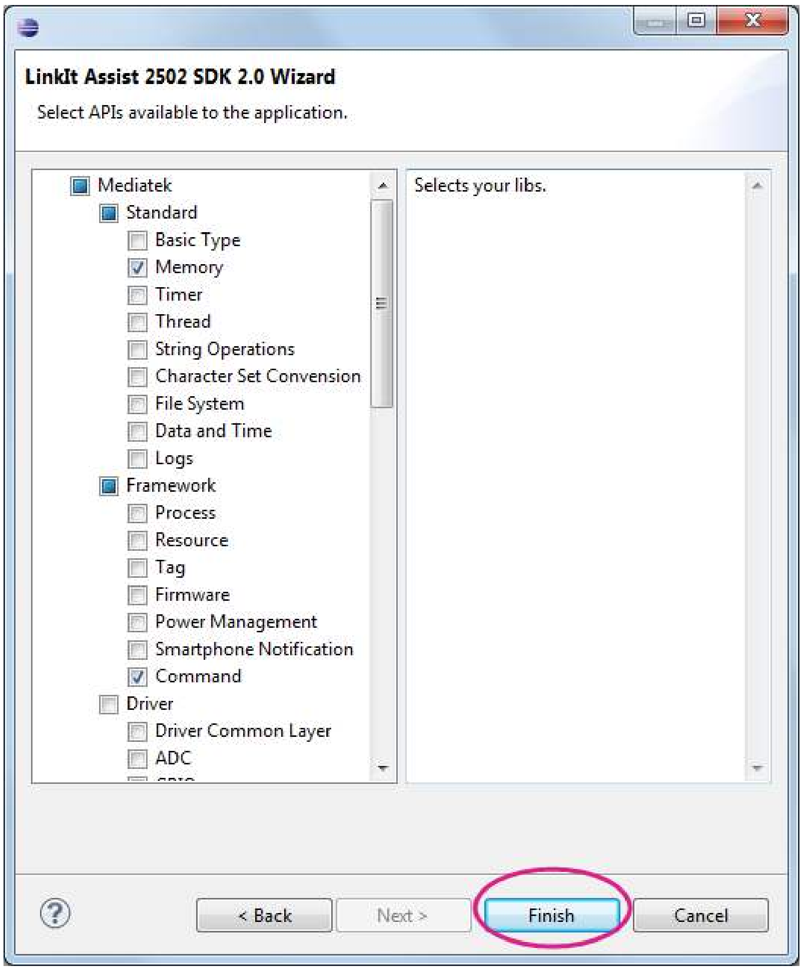
3) The **LinkIt Assist 2502 SDK 2.0 Wizard** window displays. Change the project name and make sure the **Hardware Platform** matches your development board, as shown in Figure 18. Click **Next**.


LinkIt Assist 2502 SDK 2.0 Wizard
This wizard creates a new LinkIt Assist 2502 application.

Project Name: HelloWorld
Developer name: MediaTek Inc
Application name: Demo
Hardware Platform: LinkIt Assist 2502
Empty Application: ☐
Create Project in Workspace. ☑
D:\workspace

Figure 18 Selecting the right hardware platform

4) The library selection page displays, as shown in Figure 19. The library selection page enables you to include library header files into your project. You can always manually add header files later using the `#include` command. You can proceed with default libraries for the first project and click **Finish**.


LinkIt Assist 2502 SDK 2.0 Wizard

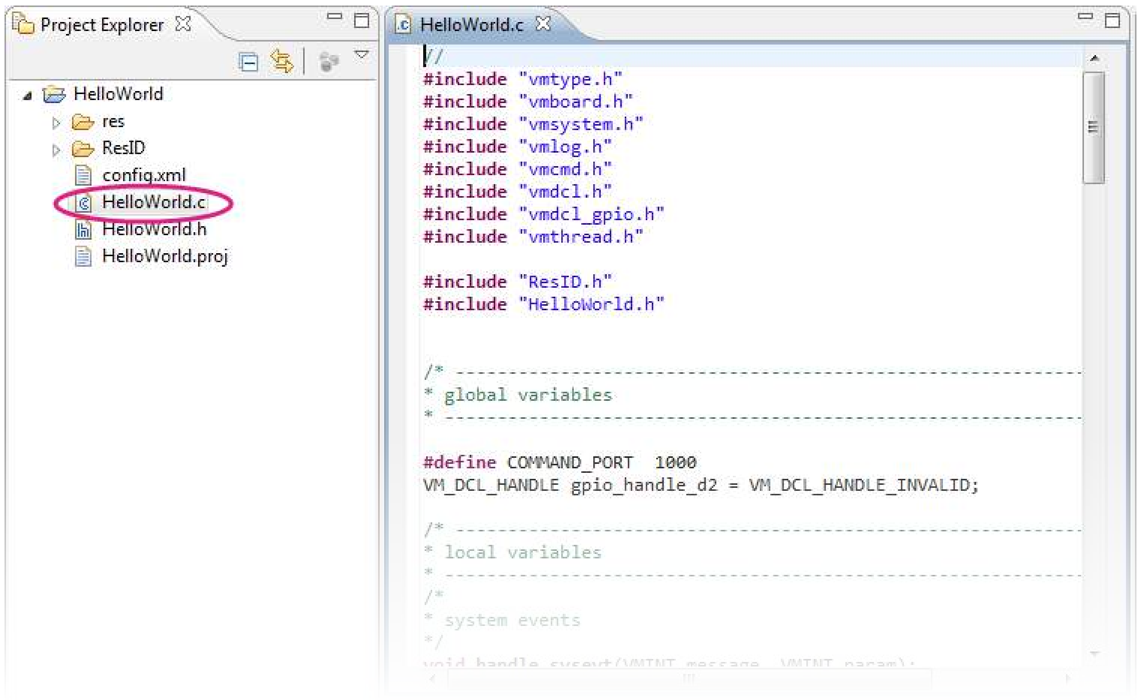
Select APIs available to the application.

- [■] Mediatek
  - [■] Standard
    - [ ] Basic Type
    - [✓] Memory
    - [ ] Timer
    - [ ] Thread
    - [ ] String Operations
    - [ ] Character Set Convension
    - [ ] File System
    - [ ] Data and Time
    - [ ] Logs
  - [■] Framework
    - [ ] Process
    - [ ] Resource
    - [ ] Tag
    - [ ] Firmware
    - [ ] Power Management
    - [ ] Smartphone Notification
    - [✓] Command
  - [ ] Driver
    - [ ] Driver Common Layer
    - [ ] ADC

Selects your libs.

Figure 19 LinkIt Assist 2502 libraries

You've now created your first LinkIt Assist 2502 application. Open the project from the project explorer pane and double click the `*.c` file, as shown in Figure 20.



```
//
#include "vmtype.h"
#include "vmboard.h"
#include "vmsystem.h"
#include "vmlog.h"
#include "vmcmd.h"
#include "vmdcl.h"
#include "vmdcl_gpio.h"
#include "vmthread.h"

#include "ResID.h"
#include "HelloWorld.h"


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

#define COMMAND_PORT  1000
VM_DCL_HANDLE gpio_handle_d2 = VM_DCL_HANDLE_INVALID;

/* ----------------------------------------------------------------
 * local variables
 * ----------------------------------------------------------------
/*
 * system events
 */
```

Figure 20 Your First LinkIt Assist 2502 Application

2.6. Compiling and Uploading to Hardware

An important thing to note before starting this section is that the **Build** and **Debug** menu commands and related toolbuttons in Eclipse IDE don't work for LinkIt Assist 2502 applications. These controls are for C/C++ applications that will run on a PC. Instead, to perform the necessary development tasks use the plug-in toolbar, as shown in Figure 21.

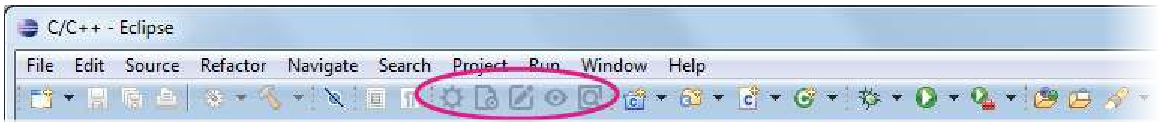


Figure 21 LinkIt toolbar in Eclipse IDE

For more details on the features of the LinkIt toolbar see 2.9, "Using the LinkIt Toolbar".

To build and upload your application:

1) Make sure your LinkIt Assist 2502 development board is connected to your PC and click the Build Application toolbutton, as shown in Figure 22.

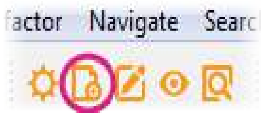


Figure 22 Build Application toolbutton in the LinkIt toolbar

2) The Eclipse IDE will build and upload the application. After it's finished, the results are displayed in the console pane. To see the build and upload log, change the console view to **LinkIt Console**, as shown in Figure 23.

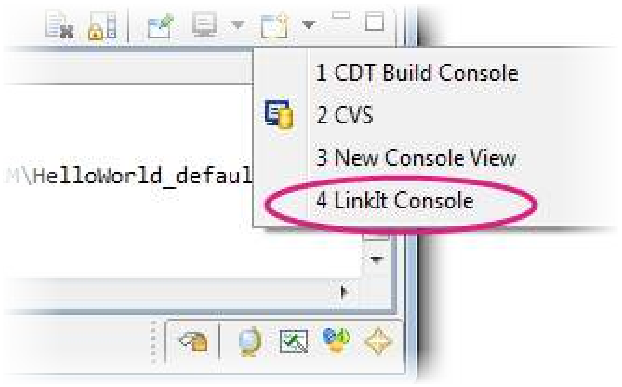


Figure 23 LinkIt Console in Eclipse

2.7. Running your Project

Once Eclipse has built and installed your application:

- On the LinkIt ONE development board: the LED starts blinking.
- On the LinkIt Assist 2502 development board: the message **HelloWorld** is shown on the display panel.

Congratulations, you have successfully completed your first LinkIt Assist 2502 SDK 2.0 project.

You can now explore the example applications provided in the SDK.

2.8. Using the example applications

The LinkIt Assist 2502 SDK 2.0 provides example application projects that you can use to further understand the API or uses as the basis of your own prototypes. Example applications are grouped by folder under `{SDK_installation_path}\examples\API.`

The example applications are listed alphabetically and named after function modules and sub-modules. For example, the `DCL_GPIO_Output` example shows how to use the GPIO function as an output for your hardware.

You can import the examples directly into Eclipse as follow:

1) In the Eclipse IDE **File** menu, point to **New** and click **Other**, or use the CTRL+N keyboard shortcut. The **New** window displays. Expand the **LinkIt Assist 2502 SDK 2.0 Example** folder, select **Example Application (*.vxp)** and click **Next** as shown in Figure 24.

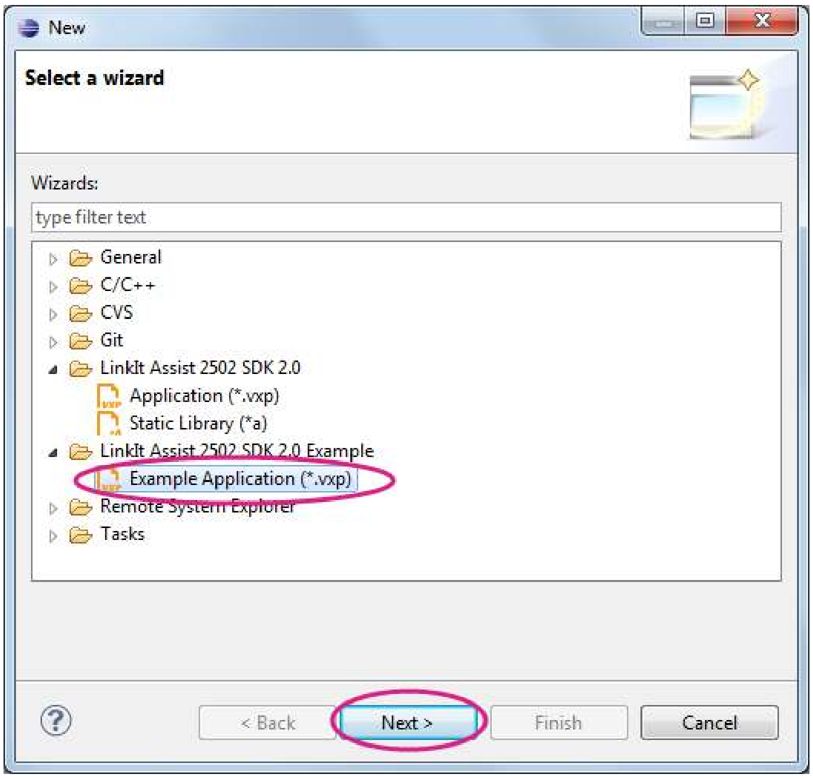
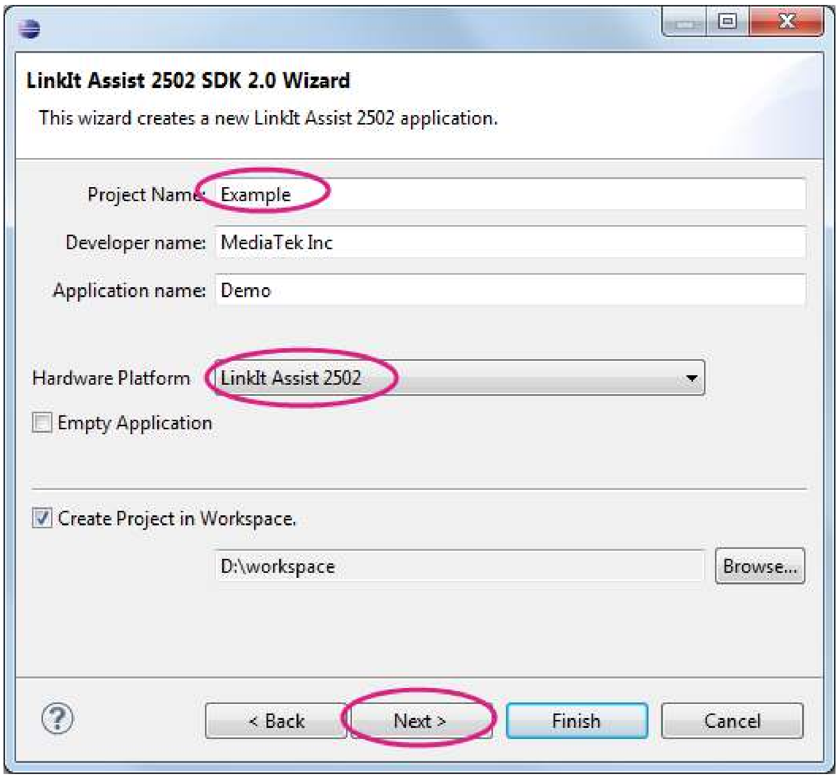


Figure 24 Selecting LinkIt Assist 2502 example code

2) In the **LinkIt Assist 2502 SDK 2.0 Example Wizard** give your project a new name and select the right hardware platform, as shown in Figure 25. Click **Next**.



LinkIt Assist 2502 SDK 2.0 Wizard

This wizard creates a new LinkIt Assist 2502 application.

Project Name: Example

Developer name: MediaTek Inc

Application name: Demo

Hardware Platform: LinkIt Assist 2502

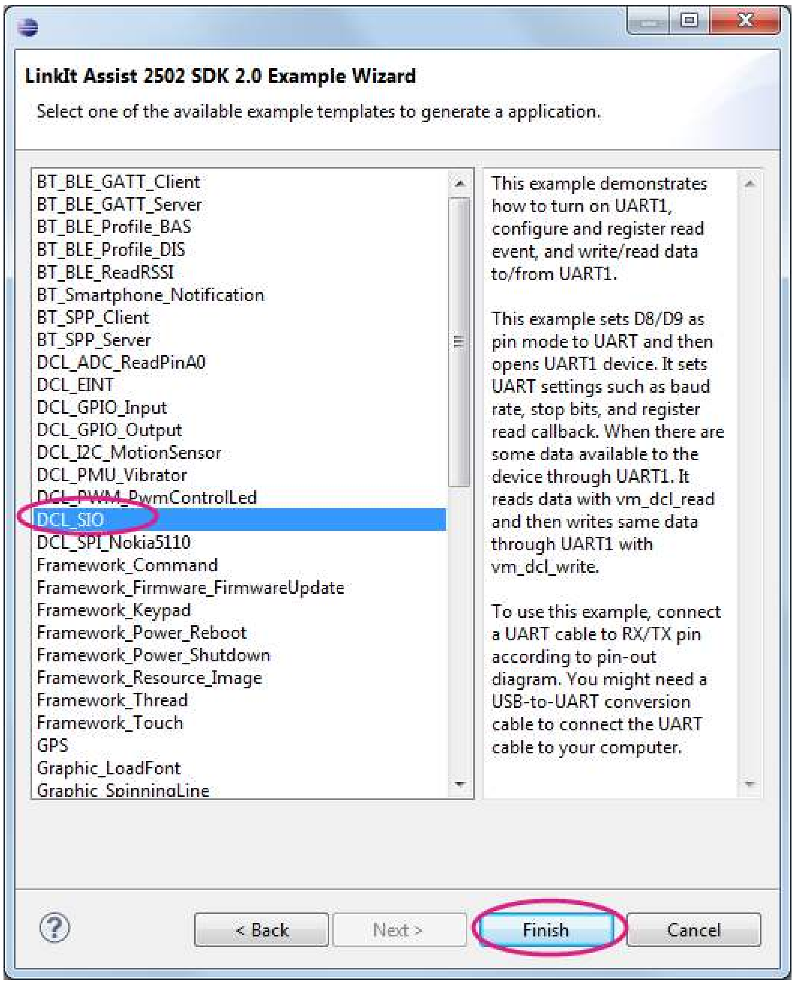
☐ Empty Application

☑ Create Project in Workspace.

D:\workspace

Figure 25 Creating a new LinkIt Assist 2502 application

3) A list of example applications (on the left) and their descriptions (on the right) appear in the **LinkIt Assist 2502 SDK 2.0 Example Wizard** window, as shown in Figure 26.


LinkIt Assist 2502 SDK 2.0 Example Wizard

Select one of the available example templates to generate a application.

BT_BLE_GATT_Client
BT_BLE_GATT_Server
BT_BLE_Profile_BAS
BT_BLE_Profile_DIS
BT_BLE_ReadRSSI
BT_Smartphone_Notification
BT_SPP_Client
BT_SPP_Server
DCL_ADC_ReadPinA0
DCL_EINT
DCL_GPIO_Input
DCL_GPIO_Output
DCL_I2C_MotionSensor
DCL_PMU_Vibrator
DCL_PWM_PwmControlLed
DCL_SIO
DCL_SPI_Nokia5110
Framework_Command
Framework_Firmware_FirmwareUpdate
Framework_Keypad
Framework_Power_Reboot
Framework_Power_Shutdown
Framework_Resource_Image
Framework_Thread
Framework_Touch
GPS
Graphic_LoadFont
Graphic_SpinningLine

This example demonstrates how to turn on UART1, configure and register read event, and write/read data to/from UART1.

This example sets D8/D9 as pin mode to UART and then opens UART1 device. It sets UART settings such as baud rate, stop bits, and register read callback. When there are some data available to the device through UART1. It reads data with vm_dcl_read and then writes same data through UART1 with vm_dcl_write.

To use this example, connect a UART cable to RX/TX pin according to pin-out diagram. You might need a USB-to-UART conversion cable to connect the UART cable to your computer.

Figure 26 LinkIt example code list

4) Select the example you want to load into the application and click **Finish**.

Eclipse will load the selected example into the application. You can now explore the example and update the code to meet your requirements.

2.9. Using the LinkIt Toolbar

The LinkIt Toolbar provides access to the LinkIt Assist 2502 application or project settings. The LinkIt toolbar options are as follow:

- Application settings (⚙)
- Build application (▣)
- Resource editor (✎)
- Monitor tool (◉)
- API reference (🔍)

These features are explained in the following sections.

2.9.1. Application Settings

To define application settings, first select a project in **Project Explorer**, then click the Application Settings toolbutton, as shown in Figure 27.

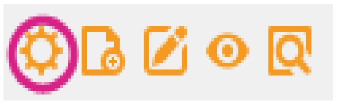
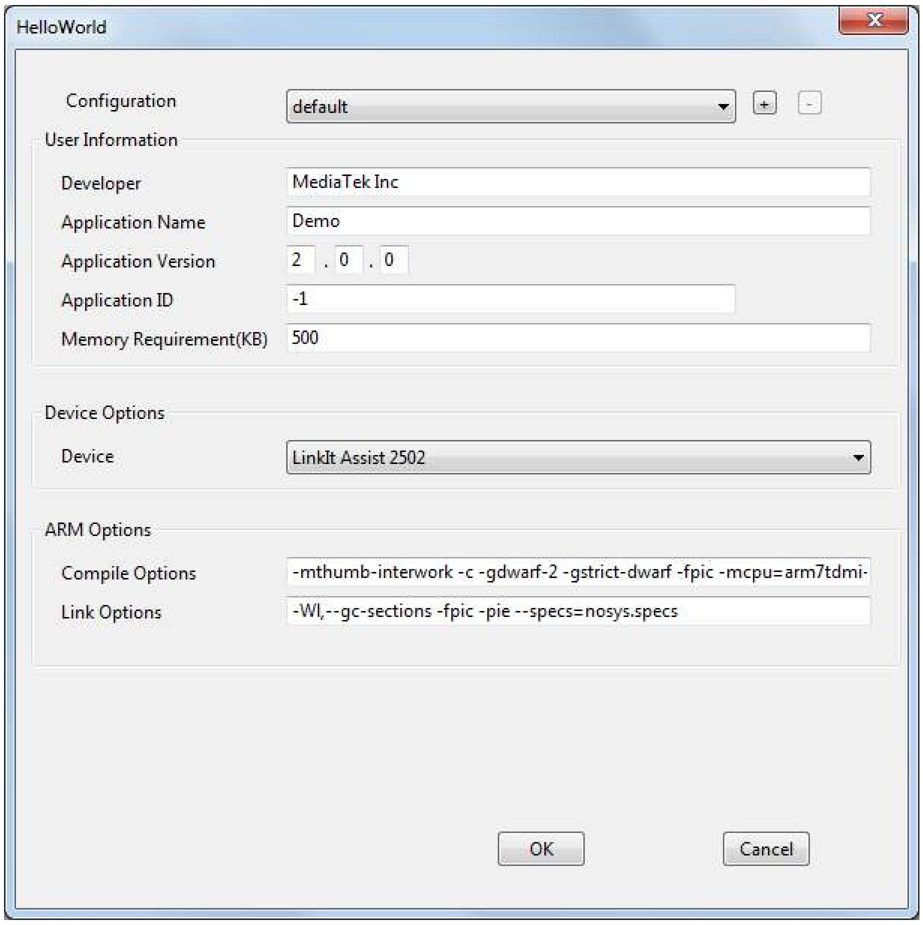


Figure 27 Application Settings toolbutton

A window for the specified project displays, as shown Figure 28.



HelloWorld

Configuration	default
User Information	
Developer	MediaTek Inc
Application Name	Demo
Application Version	2 . 0 . 0
Application ID	-1
Memory Requirement(KB)	500
Device Options	
Device	LinkIt Assist 2502
ARM Options	
Compile Options	-mthumb-interwork -c -gdwarf-2 -gstrict-dwarf -fpic -mcpu=arm7tdmi-
Link Options	-Wl,--gc-sections -fpic -pie --specs=nosys.specs

Figure 28 Project information window

The following settings are available:

- **User Information**

  The fields under **User Information** are properties of your application and they can be queried with an API Tag (please see 5.6, “Tag” for more details). **Memory Requirement** is an important field as it affects your application’s execution. If the number is set too high or too low, the application may not run. Please see 4.5, “Memory Layout of a LinkIt Assist 2502 Application”, for more information on how to set this value.

- **Device Options**

  Select your hardware development platform. Whether certain aspects of the API can operate correctly depends on the features available on the device being used, and also the pin configuration of the hardware.

- **ARM Options**

These are compiler and linker options used during the build process. You may add other options in this field but removing vital options results in build or execution errors.

2.9.2. Build Application

The Build Application toolbutton (see Figure 29) initiates the build and upload to hardware processes. The resulting logs for these processes are shown in the **LinkIt Console** panel.

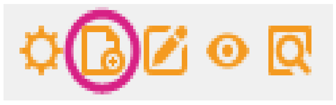


Figure 29 The Build Application toolbutton

2.9.3. Resource Editor

The Resource Editor toolbutton (as shown in Figure 30) launches the Resource Editor tool. This tool enables the embedding of audio, video, binary, image and string content into your application. Please see 4.8, "Application Resource" and 5.7, "Resource" for more details on how to use this tool and how to access it with the Resource API.

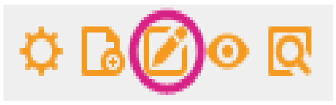


Figure 30 The Resource Editor toolbutton

2.9.4. Monitor

The Monitor toolbutton (as shown in Figure 31) launches the Monitor tool (see Figure 32) which allows you to capture application logs and send commands to the development board.

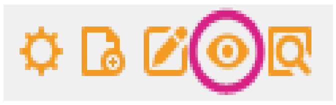
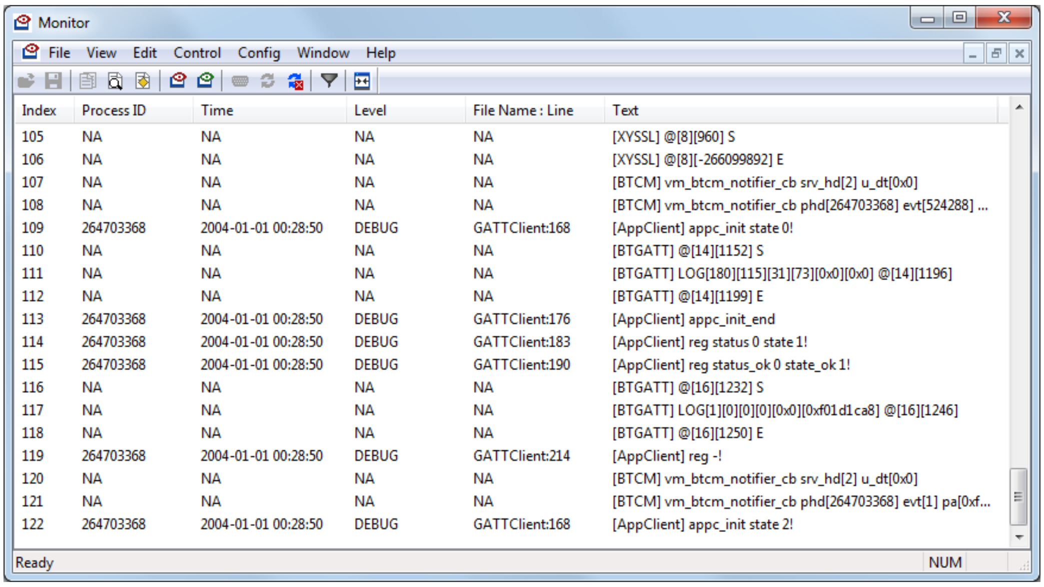


Figure 31 Monitor Icon



Index	Process ID	Time	Level	File Name : Line	Text
105	NA	NA	NA	NA	[XYSSL] @[8][960] S
106	NA	NA	NA	NA	[XYSSL] @[8][-266099892] E
107	NA	NA	NA	NA	[BTCM] vm_btcm_notifier_cb srv_hd[2] u_dt[0x0]
108	NA	NA	NA	NA	[BTCM] vm_btcm_notifier_cb phd[264703368] evt[524288] ...
109	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:168	[AppClient] appc_init state 0!
110	NA	NA	NA	NA	[BTGATT] @[14][1152] S
111	NA	NA	NA	NA	[BTGATT] LOG[180][115][31][73][0x0][0x0] @[14][1196]
112	NA	NA	NA	NA	[BTGATT] @[14][1199] E
113	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:176	[AppClient] appc_init_end
114	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:183	[AppClient] reg status 0 state 1!
115	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:190	[AppClient] reg status_ok 0 state_ok 1!
116	NA	NA	NA	NA	[BTGATT] @[16][1232] S
117	NA	NA	NA	NA	[BTGATT] LOG[1][0][0][0][0x0][0xf01d1ca8] @[16][1246]
118	NA	NA	NA	NA	[BTGATT] @[16][1250] E
119	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:214	[AppClient] reg -!
120	NA	NA	NA	NA	[BTCM] vm_btcm_notifier_cb srv_hd[2] u_dt[0x0]
121	NA	NA	NA	NA	[BTCM] vm_btcm_notifier_cb phd[264703368] evt[1] pa[0xf...
122	264703368	2004-01-01 00:28:50	DEBUG	GATTClient:168	[AppClient] appc_init state 2!

Figure 32 Monitor Tool

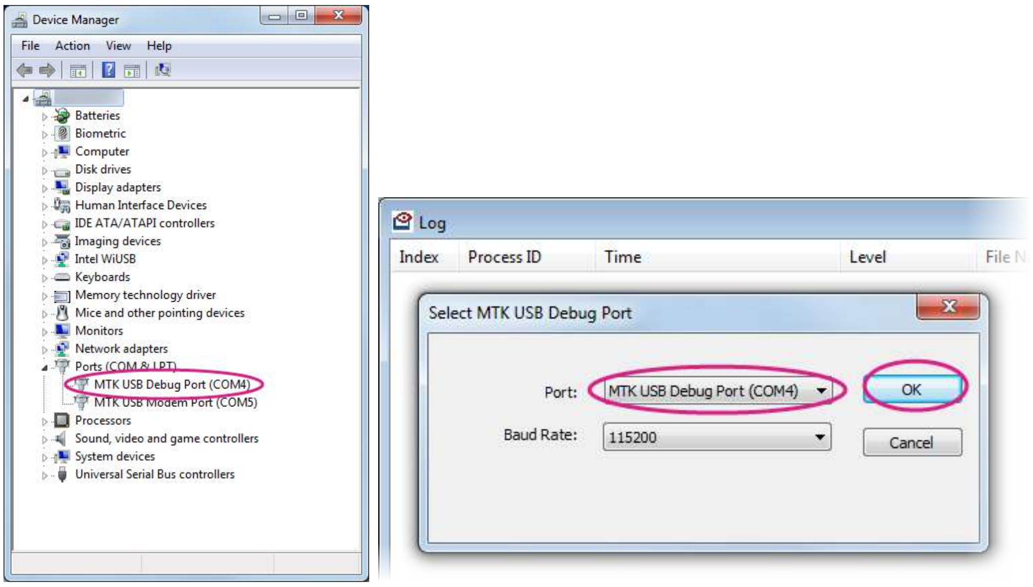
To use the Monitor tool:

1) Choose a database file, which is different for each hardware and firmware version. These databases are included in the SDK and you can find them in this path:

   `{SDK_installation_path}tools\FirmwareUpdater\firmware\LinkIt_Device\{Hardware}\{Firmware_version}\database.db`

   If you choose database file that doesn't match the hardware, a connection error may result.

2) Next, select the **COM** port, it is the MTK USB debug port listed in **Device Manager**, as shown in Figure 33. The port number varies between PCs. Click **OK.**



Port: MTK USB Debug Port (COM4)
Baud Rate: 115200

Figure 33 Device Manager COM port

3) After selecting the **COM** port, click the Connect toolbutton to establish a connection with the development board, as shown in Figure 34.

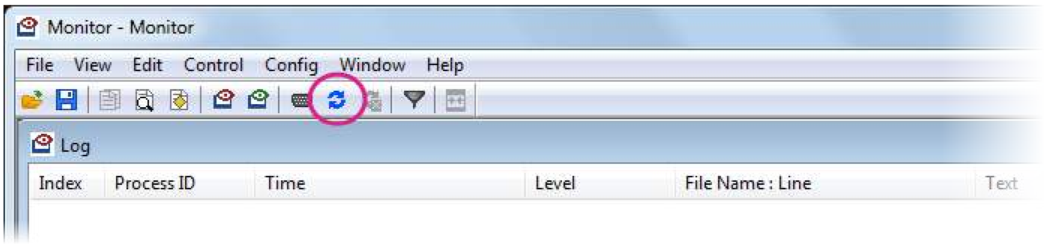
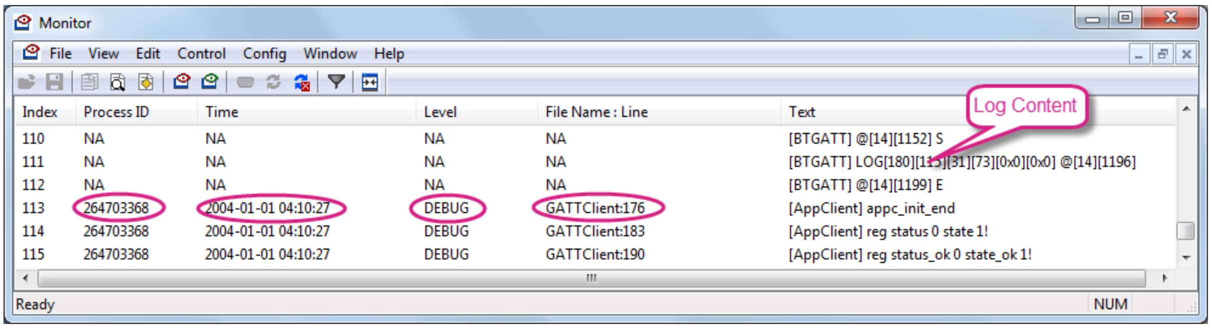


Figure 34 Connecting the COM port

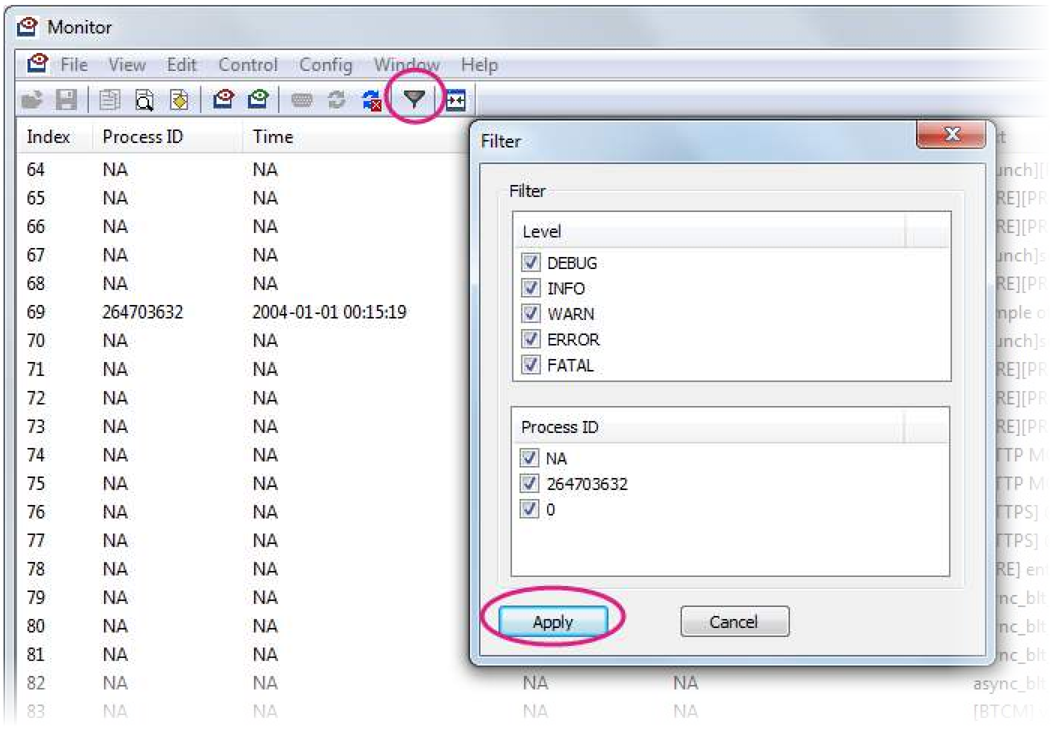
4) The log appears once the connection to the development board is established, you will see the log time, log level, file name and log content, as shown in Figure 35 .



```
Index  Process ID  Time                 Level  File Name : Line  Text
110    NA          NA                   NA     NA                [BTGATT] @[14][1152] S
111    NA          NA                   NA     NA                [BTGATT] LOG[180][115][31][73][0x0][0x0] @[14][1196]
112    NA          NA                   NA     NA                [BTGATT] @[14][1199] E
113    264703368   2004-01-01 04:10:27  DEBUG  GATTClient:176    [AppClient] appc_init_end
114    264703368   2004-01-01 04:10:27  DEBUG  GATTClient:183    [AppClient] reg status 0 state 1!
115    264703368   2004-01-01 04:10:27  DEBUG  GATTClient:190    [AppClient] reg status_ok 0 state_ok 1!
```

Figure 35 Hardware Connection Log

5) You can also create filters for different log levels with the filter toolbutton, as shown in Figure 36. Click **Apply** to invoke the filter settings.



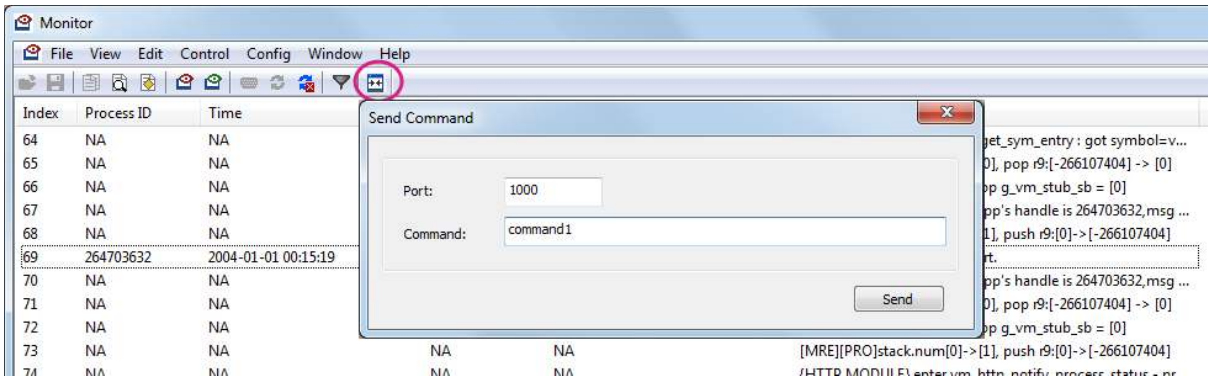
```
Index  Process ID  Time
64     NA          NA
65     NA          NA
66     NA          NA
67     NA          NA
68     NA          NA
69     264703632   2004-01-01 00:15:19
70     NA          NA
71     NA          NA
72     NA          NA
73     NA          NA
74     NA          NA
75     NA          NA
76     NA          NA
77     NA          NA
78     NA          NA
79     NA          NA
80     NA          NA
81     NA          NA
82     NA          NA
83     NA          NA
```

Level: ☑ DEBUG ☑ INFO ☑ WARN ☑ ERROR ☑ FATAL

Process ID: ☑ NA ☑ 264703632 ☑ 0

Figure 36 Log Filter window

6) You can also send commands by clicking the Send Command toolbutton, as shown in Figure 37. Please see 5.4, "Command" for the command string format.



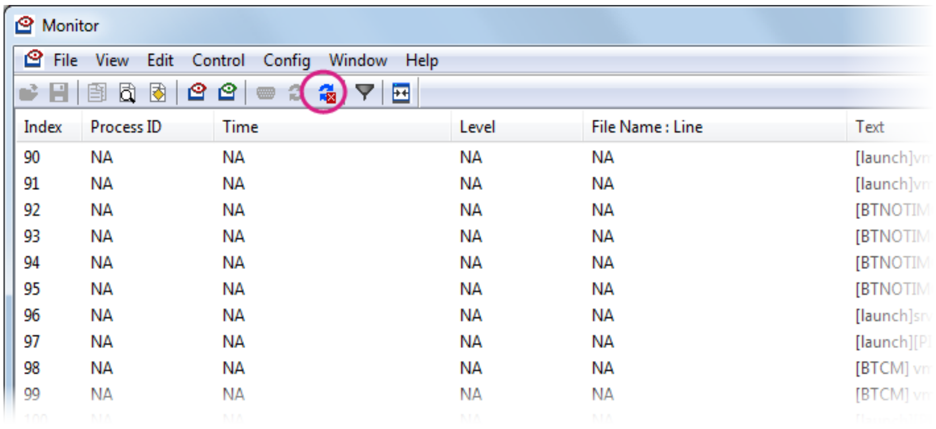
```
Index  Process ID  Time
64     NA          NA                    get_sym_entry : got symbol=v...
65     NA          NA                    0], pop r9:[-266107404] -> [0]
66     NA          NA                    pp g_vm_stub_sb = [0]
67     NA          NA                    pp's handle is 264703632,msg ...
68     NA          NA                    1], push r9:[0]->[-266107404]
69     264703632   2004-01-01 00:15:19   rt.
70     NA          NA                    pp's handle is 264703632,msg ...
71     NA          NA                    0], pop r9:[-266107404] -> [0]
72     NA          NA                    pp g_vm_stub_sb = [0]
73     NA          NA          NA   NA   [MRE][PRO]stack.num[0]->[1], push r9:[0]->[-266107404]
```

Port: 1000

Command: command1

Figure 37 Send Command window

7) To finish, click the Disconnect toolbutton and close the window, as shown in Figure 38. Please note that the COM port remains in use until the Monitor tool is disconnected.



Index	Process ID	Time	Level	File Name : Line	Text
90	NA	NA	NA	NA	[launch]v
91	NA	NA	NA	NA	[launch]v
92	NA	NA	NA	NA	[BTNOTIM
93	NA	NA	NA	NA	[BTNOTIM
94	NA	NA	NA	NA	[BTNOTIM
95	NA	NA	NA	NA	[BTNOTIM
96	NA	NA	NA	NA	[launch]s
97	NA	NA	NA	NA	[launch][P
98	NA	NA	NA	NA	[BTCM] v
99	NA	NA	NA	NA	[BTCM] v

Figure 38 Disconnect toolbutton

2.9.5. API Reference

The API reference toolbutton (see Figure 39) opens your default browser and takes you to the on-line API reference documentation.

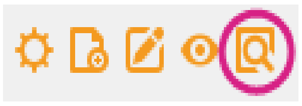


Figure 39 API Reference Icon

3. Troubleshooting

This section provides a guide to commonly encountered issues when working with the LinkIt Assist 2502 development board.

3.1. The LinkIt Assist 2502 development board fails to powerup after attaching the USB cable

If the LinkIt Assist 2502 development board fails to power up, make sure the battery is attached to the development board. (The LinkIt Assist 2502 development board cannot power on without a Lithium-Ion battery attached. A battery is included in the development board package.) Then, use a micro USB cable to connect the board to a computer and check that the power LED of the board lights. If the LED doesn't light, it means the board doesn't have a power supply from the USB cable. In this case, check if the USB cable is connected to a proper power source. Please see Figure 4 on page 5 for PWR LED location on the LinkIt Assist 2502 development board.

If the power LED lights and the board is connected to a computer with USB cable, then you should see a mass storage (removable storage) device displayed in the **Computer** folder of your computer. If it does not show, please contact hardware technical support from Seeed Studio.

If you see the mass storage device in your computer folder, press the PWR key located on the side of the board (see Figure 5 on page 5) for 3 seconds to boot-up the device. Then, the mass storage device will disappear from the Computer folder. At this stage you'll see two COM ports in the **Device Manager** of your computer.

Please note that you need to power-up the development board before the COM ports for uploading VXPs appear. Attach the battery and press the PWR key for about 3 seconds.

3.2. VXP Fails to Upload

If a VXP fails to upload, make sure the development board has been booted-up correctly, see 0, "This section provides a guide to commonly encountered issues when working with the LinkIt Assist 2502 development board.

The LinkIt Assist 2502 development board fails to powerup". Unlike the LinkIt ONE development board, which automatically boots up after attaching a battery or providing a USB power source, the LinkIt Assist 2502 development board boots-up into a normal mode after its PWR key (see Figure 5 on page 5) is pressed for about 3 seconds. If the boot-up is successful, you'll see two COM port devices in **Device Manager**. If the COM port devices don't display, try updating the board's firmware as described in 2.4, "Updating the Firmware".

If the COM port devices displays but the VXP still will not upload, please check if there is only one LinkIt Assist 2502 development board connected to your computer. If there is a second board, please disconnect it and check the correct COM ports are set in the IDE. If the VXP still fails to upload, copy the LinkIt console log message from the Eclipse IDE and post it to the MediaTek Labs forum for technical support.

3.3. Wi-Fi AP Doesn't Display After Scanning

The Wi-Fi band supported by LinkIt Assist 2502 is 2.4 GHz. Some Wi-Fi access points may be configured to provide the 5 GHz band only. In this case, please adjust the configuration of the Wi-Fi access point.

3.4. GSM Function Isn't Working

Please make sure the SIM card and your telecom operator supports a GSM (2G) network. Also, some SIM card requires activation on its first use. In this case, activate your SIM card with a mobile phone before inserting it into the LinkIt Assist 2502 development board.

3.5. GPRS Function Isn't Working

Some operators require specific APN settings to access the GPRS network. In this case, refer to your telecom operator for the correct APN settings, and use `vm_gsm_gprs_set_customized_apn_info()` to defined the correct settings in your VXP software.

4. Programming Guide

This chapter explains the bootup procedure and programming concepts for creating software for LinkIt Connect 2502 using the tools and editors provided by the LinkIt Assist 2502 SDK 2.0.

4.1. Bootup procedure

While the bootup procedure of a MediaTek MT2502 chipset cannot be altered programmatically, it's useful to understanding how it works: The flow of processes and functions that result in either the board firmware being updated or your software loaded and run.

Depending on hardware configurations there are two ways to boot up: by plugging in a USB cable or pressing the power key on the hardware.

Boot loader is initially launched after boot up. If your USB cable is connected to the PC, you'll see the **[MTK USB Port]** in **Device Manager**. The Firmware Updater tool uses this COM port to upload the new firmware (if the COM port is in upload mode), as shown in flow [2-1] in Figure 40. Otherwise, if the bootup is triggered by plugging in the USB cable, the system will enter charging mode, as shown in flow [2-2] in Figure 40.

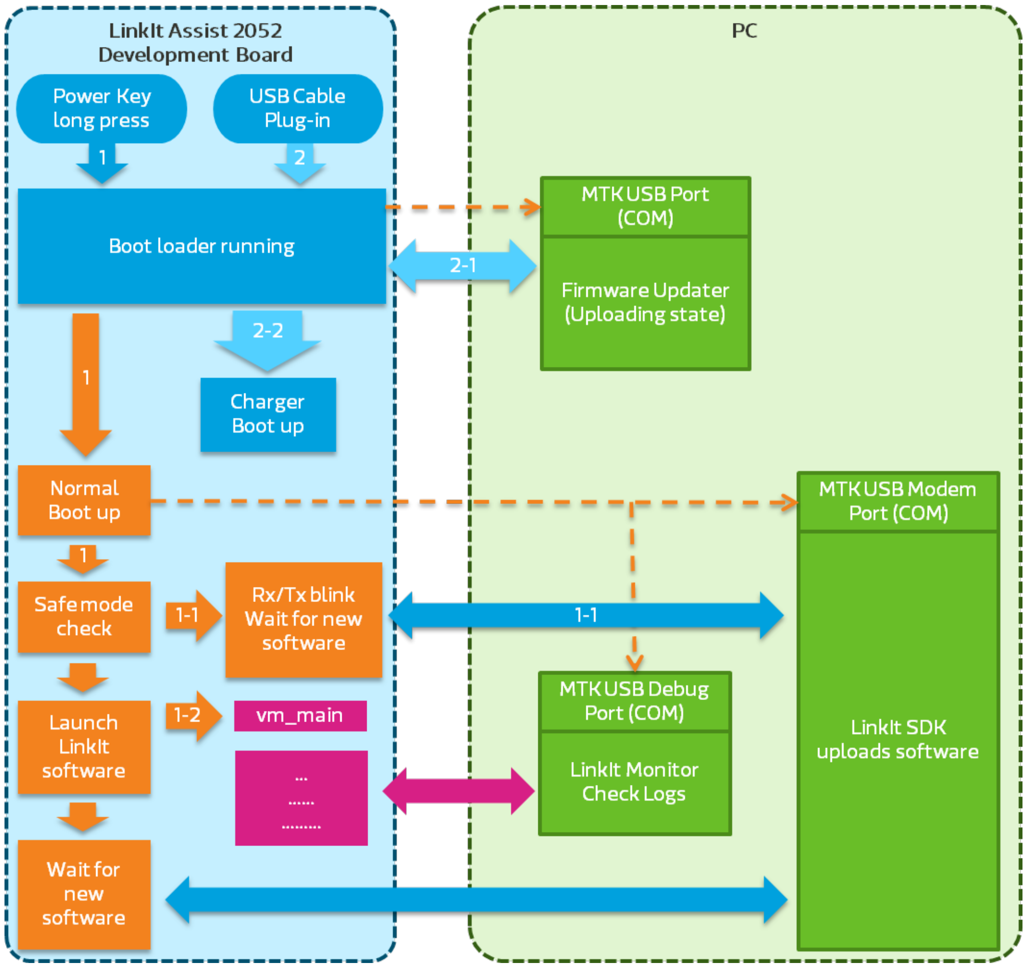


Figure 40 The bootflow sequence

If the bootup is triggered by the power key, the system will enter the normal boot up process, initialize system modules, such as driver, modem protocol, framework and alike. In this case two COM ports will be found in **Device Manager**, one for logs and another for uploading software.

Then the system will perform a safe mode check, to see if the system has crashed previously. If a previous crash is detected, the system enters safemode. The RX/TX LEDs blink and the system waits for new software to upload, as shown in flow [1-1].

If no previous crash is found, the software on the board is started by invoking `vm_main()`, as shown in flow [1-2]. For more information on `vm_main()`, please see 4.2, "LinkIt Assist 2502 Application Entry and Exit Point". If the software includes any serial port output these are sent to the serial port and may be read by the monitor in the LinkIt 2502 SDK.

The SDK can upload a new application at anytime in this state.

4.2. LinkIt Assist 2502 Application Entry and Exit Point

The Runtime launches VXP executable file after the firmware completes the bootup procedure.

After uploading your LinkIt Assist 2502 application using the **Uploader** tool, a new text file will be created in the system partition of the development board which is named `autostart.txt`. The content of this file is the filename of the LinkIt Assist 2502 application (VXP filename). Once the firmware bootup is complete, the LinkIt API run-time environment reads the VXP file and launches the LinkIt Assist 2502 application. The VXP file is then loaded into memory, and resolves the actual RAM address of the LinkIt Assist 2502 API referenced in the VXP file.. Finally, the runtime executes the function `vm_main()` defined in your application. `vm_main()` function contains register event handlers: either a default one specified by the LinkIt Assist 2502 application, or the ones defined by you. The system will call the corresponding processes or events defined by the event handler. In addition, your application can also obtain the program's operating state periodically by setting a timer to handle the different states accordingly. Therefore, the default implementation of `vm_main()` is a call to register a system event handler callback function.

```
void vm_main(void) {
    vm_pmng_register_system_event_callback (app_system_event_handler);
}
```

`vm_main()` is the main entry point for your LinkIt Assist 2502 application. It is implemented by your application, and in general, registers system events and callback functions for input events, if any. Note that unlike other platforms, where a standard C programs terminate after the `vm_main()` function returns, a LinkIt Assist 2502 application keeps running after the `vm_main()` function returns.

`vm_exit_app()` provides an exit function for the LinkIt Assist 2502 application. In this function, your application is unloaded and all occupied resources released. It's usually the last LinkIt Assist 2502 program statement, after which no other LinkIt Assist program can be executed.

During the initial call of `vm_main()` function, LinkIt Assist 2502 sends `VM_EVENT_CREATE` message to the software.

4.3. Event-driven Programming Model

LinkIt Assist 2502 SDK 2.0 adopts an event-driven programming model that allows the system to stay in a power-efficient idle mode when there is no incoming event. Instead of iterating through each software and hardware component to check their state, the developer writes event handlers that are triggered when corresponding system event is called. Due to the nature of an event-driven programming model, event handling routines should not occupy execution context for a long period of time. Failing to do so will result in a congested event processing queue, lead to slow system response, and even system crash. Therefore, it is advised that instead of relying on polling software and hardware status, you should register callback functions for system events. For periodic activities, e.g. blinking a LED with GPIO pins, instead of creating a loop that forces entire system to sleep a certain amount of time before changing the GPIO pin status, a timer API should be used to register a callback function. The timer will be invoked at a specified period of time, and change the GPIO pin status in the callback function.

During the application execution the application message queue stores messages handlers. Only when the code that processes the current message is completed will the next message in the queue be retrieved and processed. Therefore if it takes too long to process a message, the other messages in the queue may not receive responses in a timely manner. This may not only affect user experience but may lead to exceptions as well. It is recommended that operations that require long processing times be broken down into multiple steps, and a timer can be used to control these processes. For example, if it is necessary to load a large amount of resources, a timer can be created to load the resources part by part after each timeout of the timer.

Since LinkIt Assist 2502 applications are message-driven, upon execution, application needs to handle at least two system messages, they are `VM_EVENT_CREATE` and `VM_EVENT_QUIT`.

In the course of launching the application, Runtime framework will send a `VM_EVENT_CREATE` message to the application. At first, in most cases, application will do initialization and allocation of resources in the message handler of `VM_EVENT_CREATE`. If an application is killed, for example after calling `vm_exit_app()`, LinkIt Assist 2502 will trigger the system message `VM_EVENT_QUIT`. In the message handler of `VM_EVENT_QUIT`, it is supposed to release all resources used by the application. If there are still running threads or timers accessing these resources, the program will crash. Therefore, the developer must ensure that no code is executed after `vm_exit_app()` is called.

Application can receive other system messages with no need to handle, such as `VM_EVENT_LOW_BATTERY`, `VM_EVENT_CARD_PLUG_OUT`. See the API reference for detailed description of each system event. Some modules require the developer to process specific events. For example, once the graphics subsystem is ready, the message `VM_EVENT_PAINT` is sent to the application indicating that the graphics module is ready to use.

4.4. Threads

As described in 2.1"Environment", the application is executed in a multi-threaded environment and the application itself runs in the **main thread**, and most APIs are only available to the main thread. The application can create and control a limited number of new threads with the `vm_thread` function. Basic inter-thread communication message mechanisms and synchronization mechanisms are also provided. Refer to 5.5.5, "Threads" for details.

4.5. Memory Layout of a LinkIt Assist 2502 Application

This section explains the memory layout of an application, and how to estimate the Memory Requirement of an application, which should be configured for each application in the Project Settings (see 2.9.1, "Application Settings").

The runtime memory of an application can be broken down into three parts:

1) The memory occupied by the application code itself. This part of the data is loaded into memory at program startup, and will stay in memory until it terminates.
2) The memory dynamically allocated at the runtime using a separate heap for each application.
3) The stack space for function calls and local variables.

The use of fonts requires additional memory for your running software, in the form of the font cache: 100KB of memory when using vector fonts and 50KB for bitmap fonts. To ensure the best performance when using fonts, subtract the font cache size from maximum allowed memory size for your application.

The application must have proper configuration that includes the size of the application code plus its heap size. The stack memory is not configurable and is pre-allocated by the system firmware. Since the stack memory is not configurable and is limited, do not attempt to define local variables that are too big within the body of a function. It is recommended to use `vm_malloc()` to dynamically allocate memory on the heap. Otherwise, when the level call to the function becomes too deep, fatal error may occur due to stack overflow.

To properly configure the memory size for heap and software code, two factors should be taken into consideration: the maximum possible memory available on the system, and the required memory for the software code and heap.

The maximum possible memory available depends on hardware and system firmware. This value can be different for different versions of the SDK. You can also make a query through API `vm_firmware_get_info()`.

The size of the required memory for the application depends on a behavior of the application. After an application is loaded into memory, the memory layout is presented in Figure 41.

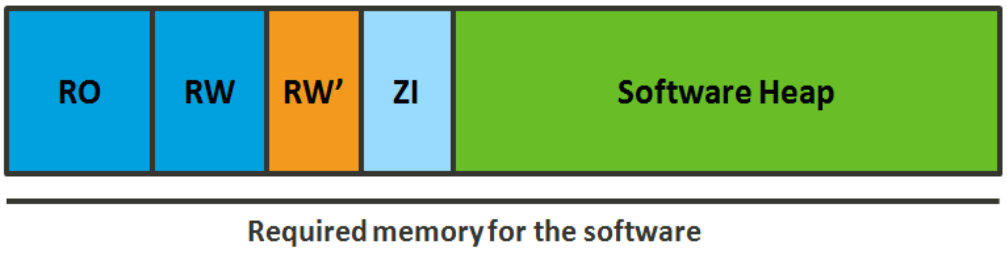


Figure 41 Memory Layout of an application

The Read-Only (RO), Read-Write (RW, RW'), Zero-Initialized (ZI) regions are copied from the executable file. And the *Software Heap* is the memory partition that is used to provide memory space for `vm_malloc()` calls in LinkIt Assist 2502 application.

The size of the memory that a running application can allocate dynamically (on the Software Heap) can be calculated with the following formula:

Size of memory that can be dynamically allocated =
Size of memory requested by the application - Size of memory occupied by the application file.

The total memory space requested by application varies by hardware platform and system firmware.

Suppose the total memory space requested by LinkIt Assist 2502 is 800KB. The Macro window will show the following information when VXP is being generated:

```
Total RO  Size(Code + RO Data)          13616 (  13.30KB) Total
RW  Size(RW Data + ZI Data)               1956 (   1.91KB)
Total ROM Size(Code + RO Data + RW Data)13620 (  13.30KB)
```

Then the size of memory that can be dynamically allocated will be about 800KB-1.91KB-13.30KB=784.79KB

In this case, it's advised to set the Memory Requirement of the application to 800KB. To set the Memory requirement of the application, refer to 2.9.1, "Application Settings".

It's good to keep in mind that ARM processors are very strict about byte alignment requirements. If it's necessary to access the content of 4 bytes in one step, then the starting address of that content must be positioned on the 4-byte boundary. Similarly, if it's necessary to access the content of 2 bytes in a single step, then the starting address of that content must be positioned on the 2-byte boundary and so on. Failure to do so will cause an exception to occur.

Alignment problems occur mainly with incorrect pointer conversion, such as converting a byte pointer into a structure pointer. This type of pointer conversion should be avoided. The pointer returned by `vm_malloc` must be typecast and the pointer returned by `vm_malloc`, regardless of the size of its allocated space, must be aligned properly.

4.6. Log and Command

Once you connect the development board to the computer, two MTK USB COM ports become available; one for debugging purposes, and the other for using the device as a modem. You can see this in **Device Manager**.

The modem port is used to upload the code by the Uploader tool. The Debug port is used by the Monitor tool to display logs output by the application, and also send commands to the application. The application can output logs as described in 5.3, "Log", and process commands as described in 5.4, "Command".

To display the log, launch and connect the Monitor tool as described in 2.9.4, "Monitor". To send a command, click on the Connect toolbutton ( ) first, and then the Send Command toolbutton ( ), as shown in Figure 42.

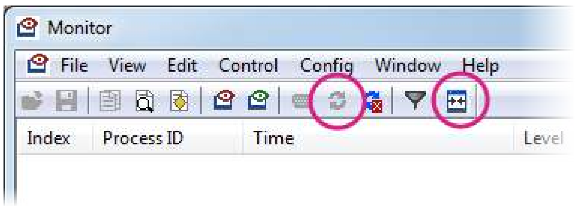
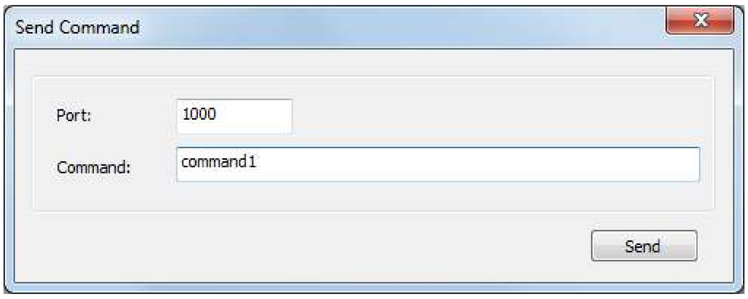


Figure 42 Sending command in Monitor tool

The **Send Command** window displays, type a port number (valid from 500 to 65535) in **Port**, then the command string, which you define (maximum length is 512 bytes) in **Command**, as shown in Figure 43:



Send Command

Port: 1000

Command: command1

Figure 43 Send Command dialog box

The command format is `command_string`. The port numbers are assigned when the software registers the Command module. For details, refer to 5.4, "Command".

4.7. Hardware Peripheral and Driver Functions

LinkIt Assist 2502 HDKs come with a set of breakout pins to connect to various kinds of hardware peripherals. For example GPIO to receive inputs for switches and I2C interface to connect to sensor modules that support I2C interface. To connect to peripherals, use the Driver Common Layer (DCL) module, which is described in 5.10, "Driver API".

The hardware interfaces supported are described in this section.

While the hardware interfaces described in this section are supported by the SDK, they may not be available on all development boards. Refer to the development board specifications for details.

4.7.1. GPIO

GPIO module is one of the most *fundamental* hardware modules. Usually it's used for transmitting digital signals. Digital signals have two statuses, HIGH and LOW, associated with voltage values of the hardware. The voltage values depend on the hardware configuration of the I/O pins. For example, in LinkIt ONE, all pins are in 3.3V to be compatible with Arduino devices, while the LinkIt Assist 2502 has pins in 2.8V to save redundant external voltage conversion circuits. Refer to 5.10.2, "GPIO" for details on the use of GPIO functions in DCL module.

One thing to notice is that in a multi-threaded environment, the threads are prioritized and he exact timing of signals on GPIO pins varies based on the system scheduling behavior. It is suggested instead of simulating signal patterns with GPIO module, use corresponding hardware-supported modules such as I2C, SPI, and PWM modules.

4.7.2. ADC

ADC (Analog-to-digital converter) module reads input voltage value of an assigned analog input pin, and then converts the input voltage value 0 ~ *reference voltage* to discrete integers 0 ~ 1023. Note that the input voltage depends on hardware configurations. For example, the reference voltage is 2.8V in LinkIt Assist 2502, but in LinkIt ONE the input to analog pins is divided into half with external circuits, so it actually turns into 5.6V reference voltage.

4.7.3. I2C

I2C (Inter-Integrated Circuit) protocol provides two pins; SDA and SCL These pins are used to communicate with I2C slave devices connected to the I2C bus. Refer to 5.10.3, "I2C" for I2C commands.

4.7.4. SPI

SPI (Serial Peripheral Interface) is a synchronous serial external port for users to conduct data communication. On LinkIt Assist 2502 platform, SPI needs the following pins to complete communication:

- MISO (Master In Slave Out): Transmits data from slave to master
- MOSI (Master Out Slave In): Transmits data from master to slave
- SCK (Serial Clock): Serial clock output from master, used to sync signal timing between master and slave

It is possible that many slaves connect to one master at the same time. There is a particular pin for the selection of a slave:

- SS (Slave Select): Master selects a slave to communicate to through this pin, when the SS signal of the slave is at low voltage.

The SPI is available only in master mode, and in current version only one set of SPI interface is supported.

4.7.5. Serial IO (UART)

The Serial IO (SIO) module supports the Universal Asynchronous Receiver/Transmitter (UART) protocol. This module is capable of exchanging data with PC or other devices. Set up the COM port at the same baud rate of a remote device to exchange data. In the SDK there are two kinds of SIO devices: hardware UART ports, which are accessible through DCL SIO module, and USB UART ports, which are emulated COM ports that appear when the HDK is connected to a PC. Note that the USB UART port serves a special purpose on LinkIt Assist 2502 SDK 2.0 such as executable uploading and logging, and therefore it is not available for other uses.

4.7.6. PWM

PWM (Pulse Width Modulation) is used to convert discrete digital signals to a continuous analog signal for writing to an assigned pin. The application can use PWM to light a LED at various brightness levels, or control a motor. Different hardware support different number of PWM pins, for example both LinkIt ONE and LinkIt Assist 2502 development boards come with 2 sets of PWM pins, however one of the PWM pin on the LinkIt Assist 2502 development board is used to control the brightness of the display module backlight.

4.7.7. EINT

External Interrupt (EINT) module allows applications to handle events triggered by voltage levels or voltage changes. One thing to notice is that in the SDK, the interrupt handler is handled in the main thread context. When an interrupt is triggered in the hardware, the interrupt service handler in the system creates an event to the main thread, and then the application callbacks are invoked in the context of the main thread. Therefore, the callback handlers of the application won't interrupt the execution of each other, and are always executed in a sequence.

4.7.8. PMU

The power management unit (PMU) allows applications to control the power of certain GPIO pins and certain parts of the system, for example the VIBR LDO output, which is connected to a vibrator in LinkIt Assist 2502. But, the VIBR LDO does not work on LinkIt ONE boards, which does not have a vibrator and therefore doesn't expose this pin in the breakout.

4.8. Application Resource

The SDK allows you to embed resources such as images, strings and files (binary blobs, such as audio) into the application executable. For string resources, the Resource Editor can store strings in different languages, which allows loading applications that require switching between languages.

These external resources are created in the resource editor and included in the application (VXP file) as part of the build process. At runtime functions of the Resource module searches for the address of a given resource id, loads the content of the resource into RAM and report the RAM address to the application.

Refer to 5.7, "Resource" for details on how to retrieve the embedded resource with functions from the Resource module. This section explains how to use the tool. Once the Resource Editor tool is launched, as described in 2.9.3, "Resource Editor", it enables resources to be added to the project file of the application.

4.8.1. Adding File and Image Resources

Resources such as audio, image and arbitrary binary files can be embedded by choosing corresponding items in the resource type panel, as shown in Figure 44.

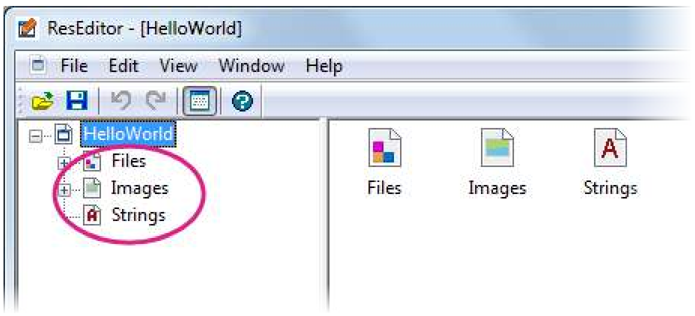


Figure 44 Resource Types

To add an image, click on the **Images** folder first, then **Default** as shown in Figure 45.

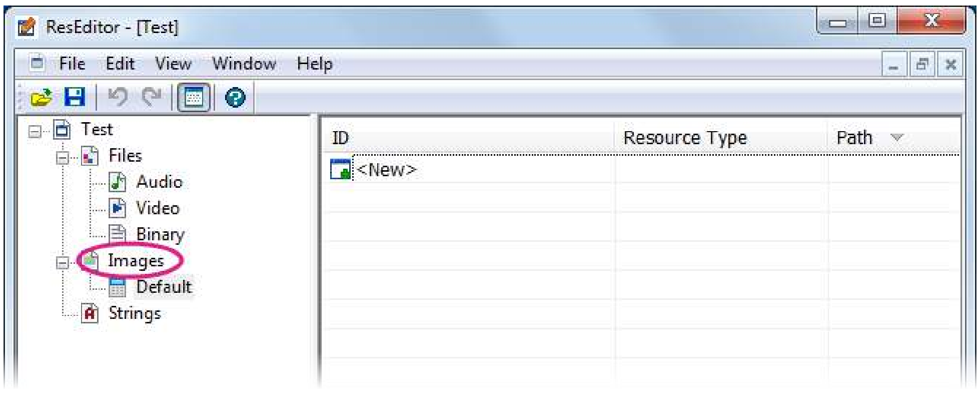


Figure 45 Selecting the Images resource type

Then double-click on the <**New**> ID to assign the identifier to the new resource item, as shown in Figure 46. This identifier will be used by the application to load the resource with Resource module.

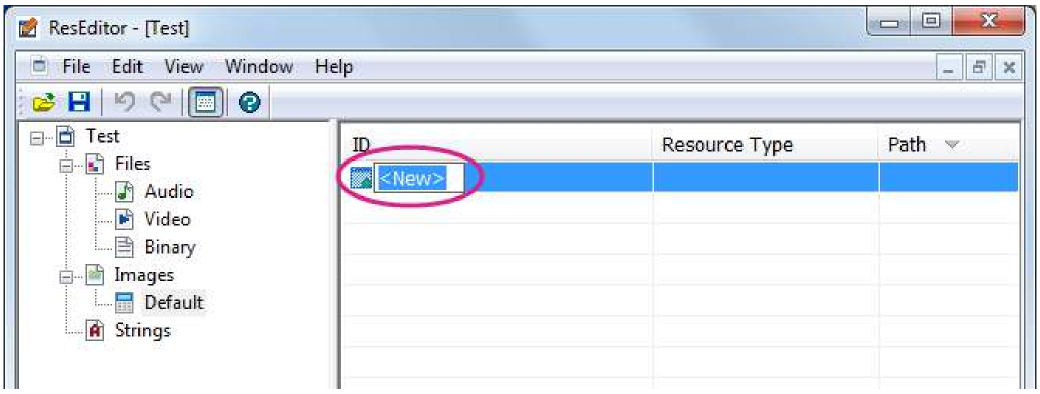


Figure 46Image resource example

Please note the following restrictions on the naming of resource identifiers:

- The ID length cannot be greater than 64.
- The initial character cannot be 0-9.
- All characters must be A - Z, a - z, 0 - 9, or '_'.

In this example, the image resource is named as `APP_IMG_1`. Then click under Resource Type and select **Normal** and click **Browse** to select the image file you would like to embed into the application executable, as shown in Figure 47.

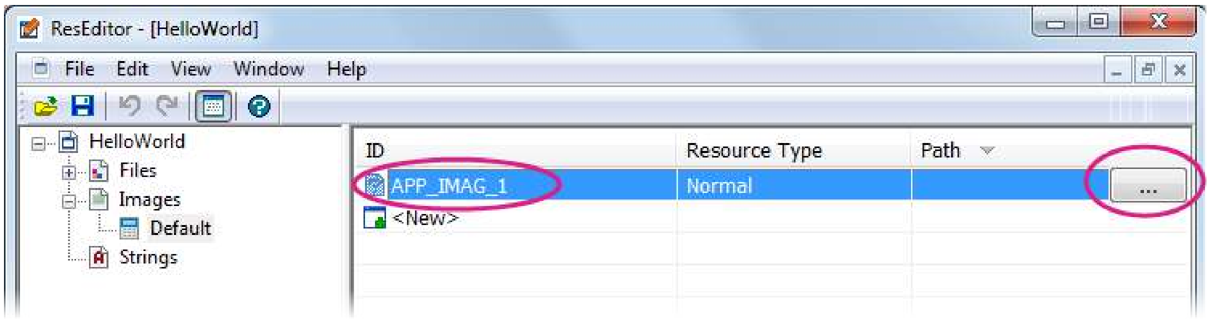


Figure 47 Select image file

Finally, click the Save toolbutton to save the settings into the project as shown in Figure 48. After you've built the application, the file you chose will be embedded into the application and you can access it with resource API as described in 5.7, "Resource".

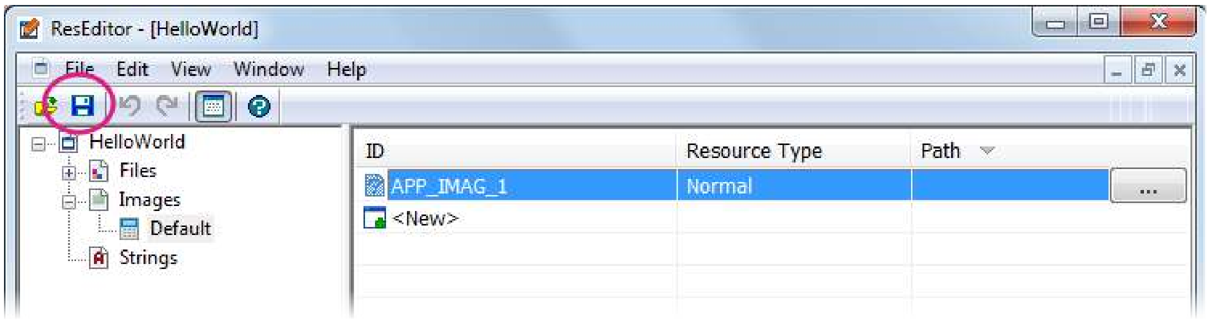


Figure 48 The save toolbutton

Audio and binary files can be embedded in a similar way.

4.8.2. Adding String Resources

Strings can be added the same way as files, but since multiple languages are supported, first define the languages to be supported by selecting the **Strings** folder, as shown in Figure 49.

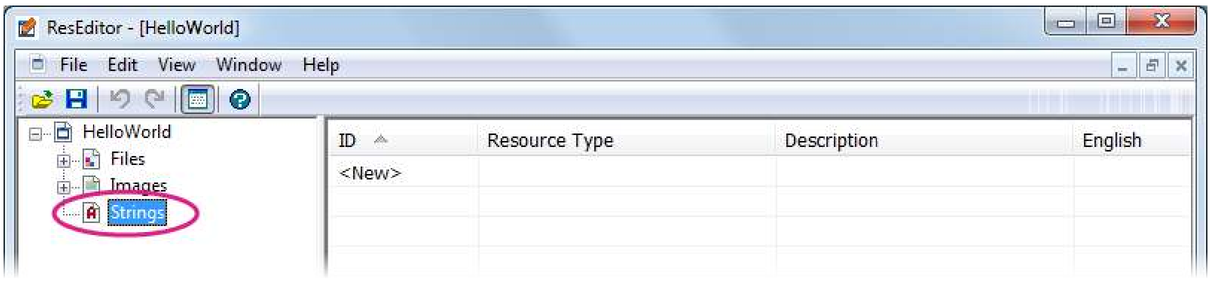


Figure 49 Selecting Strings resource type

On the <**New**> row open the shortcut menu and select **Language**, as shown in Figure 50.

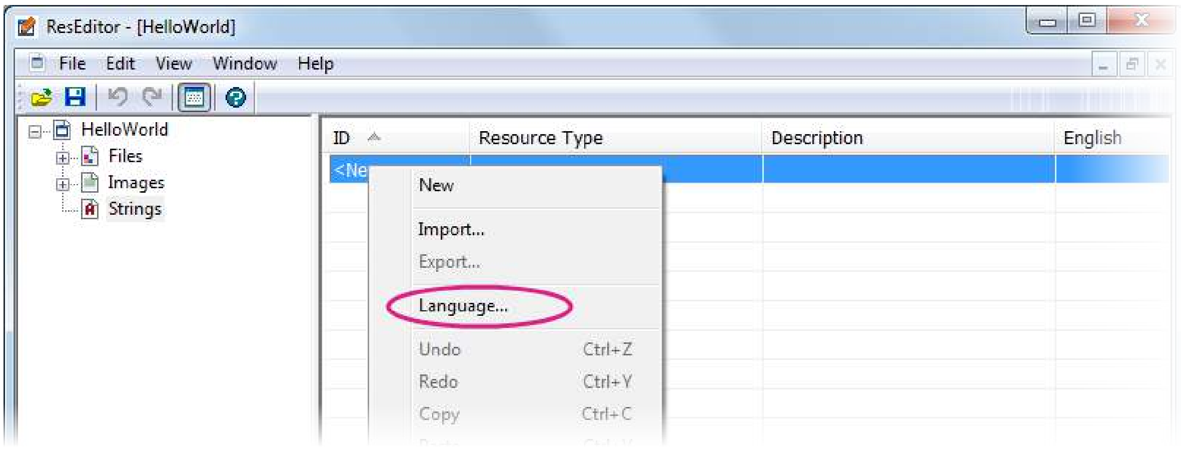
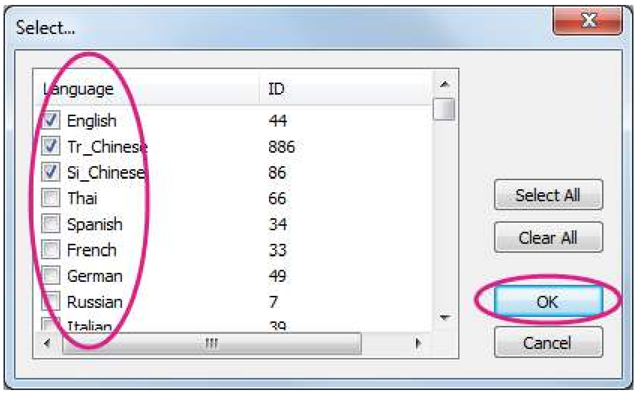


Figure 50 Selecting language for string resources

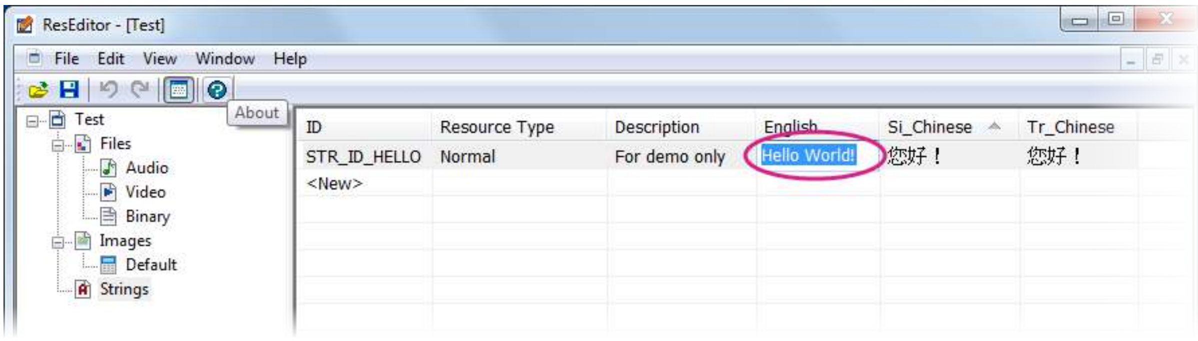
Choose the set of languages you would like to add as columns in the resource editor, as shown in Figure 51. The **Language ID** is used to define different languages in the Resource module. Click **OK**.



Language	ID
✓ English	44
✓ Tr_Chinese	886
✓ Si_Chinese	86
Thai	66
Spanish	34
French	33
German	49
Russian	7
Italian	39

Figure 51 Select the languages to be included in the string resources

Next, edit the text for each language by double clicking below the selected language, such as English as shown in Figure 52.Figure 53 Import/Export String resources



ID	Resource Type	Description	English	Si_Chinese	Tr_Chinese
STR_ID_HELLO	Normal	For demo only	Hello World!	您好！	您好！
<New>					

Figure 52 Edit text for selected language

It is also possible to export and import the string table to Excel files, by choosing **Export** or **Import** from the shortcut menu as shown in Figure 53.

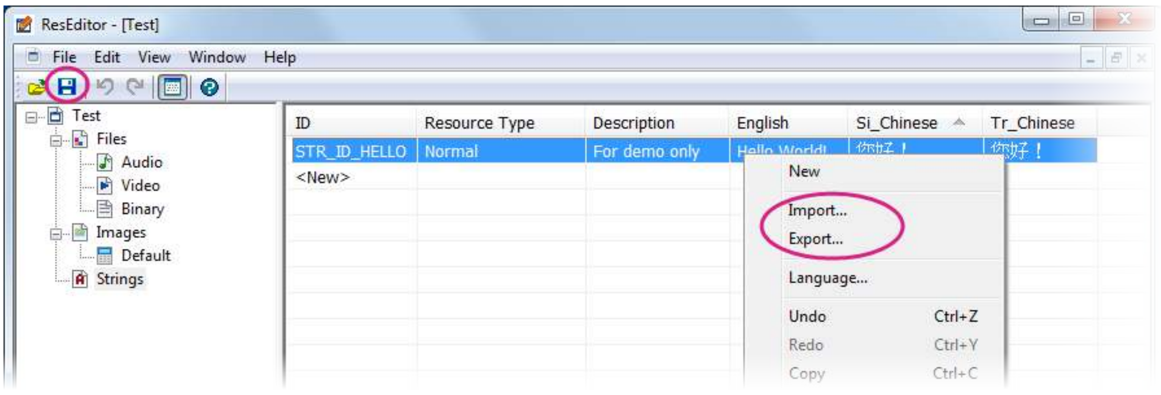


Figure 53 Import/Export String resources

Click the Save toolbutton to save the resulting string table to the project. Refer to 5.7, "Resource" for details on how to load these resources into a software.

4.9. Power Management

SDK does not provide an explicit power-management controller. Instead, since the application is using an event-driven programming approach, the system enters into a sleep mode automatically whenever the event handlers of the application are not running. This allows the system to preserve power when there are fewer events in the system, instead of being managed explicitly by the application.

Some major hardware connectivity modules can be enabled and disabled explicitly to ensure the hardware is completely shutdown to further preserve power. These modules provide separate APIs to enable and disable their hardware components. When the device boots up, most of the hardware modules are disables by default. The application must enable these hardware modules explicitly before accessing their functionalities.

Table 4 shows hardware components that require explicit enable and disable:

MODULE	Power on API	Power off API
Bluetooth	`vm_bt_cm_switch_on`	`vm_bt_cm_switch_off`
WiFi	`vm_wlan_mode_set with VM_WLAN_MODE_STA`	`vm_wlan_mode_set with VM_WLAN_MODE_OFF`
GSM/GPRS	`vm_gsm_switch_mode`	`vm_gsm_switch_mode`
GPS	`vm_gps_open`	`vm_gps_close`
LCM	Future release	Future release

Table 4 The functions available to enable and disable hardware components

4.10. Porting Arduino Sketches and Drivers

It's possible to port Arduino sketches and some peripheral device drivers for use in the LinkIt Assist 2502 development platform. The porting mechanism is similar to that of LinkIt ONE development platform, where a porting layer is provided to execute Arduino sketches and related drivers in the LinkIt ONE run-time environment.

The only difference is that this porting layer is built-in on the LinkIt ONE development platform, whereas in the LinkIt Assist 2502 development platform a subset of the porting layer is provided as an example project. You can import this project into the SDK 2.0 and then integrate Arduino sketches.

The porting example supports core Arduino functionality only, which includes the following:

- Arduino core functions such as pinMode, delay and digitalWrite.
- `Serial1` maps to the hardware transmit and receive ports. However, `Serial` is not available because it maps to USB UART port, which is already occupied by the Monitor tool.
- Wire (I2C) module.
- SPI module.

To port your Sketch from the LinkIt ONE development platform you take the following steps:

1) Import the example project from the LinkIt Assist 2502 SDK's `custom\Arduino` folder as follows:

   a) In Eclipse, on the **File** menu click **Import**. In the **Select** window, open the **General** folder and click **Existing Projects into Workspace** as the import source, as shown in **Error! Reference source not found.**Figure 54.

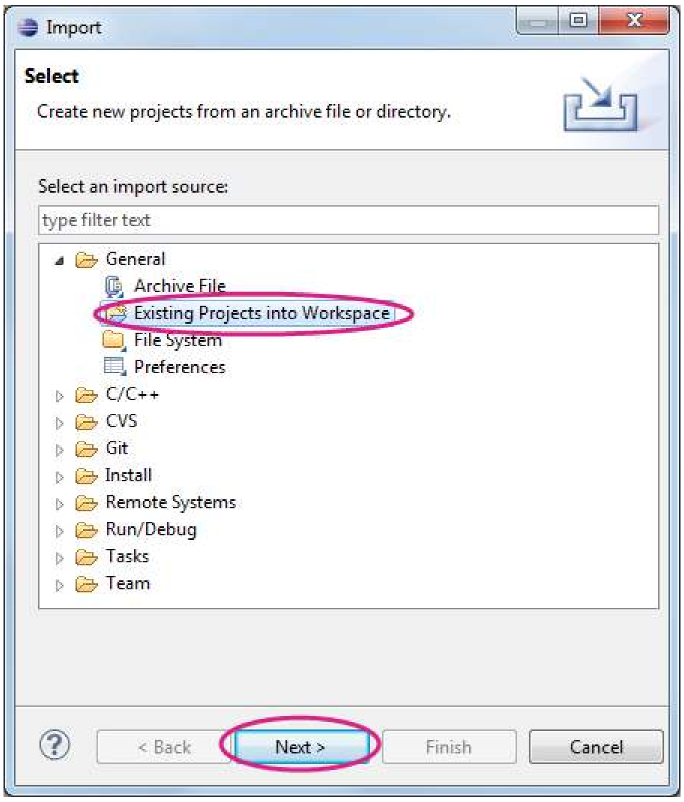
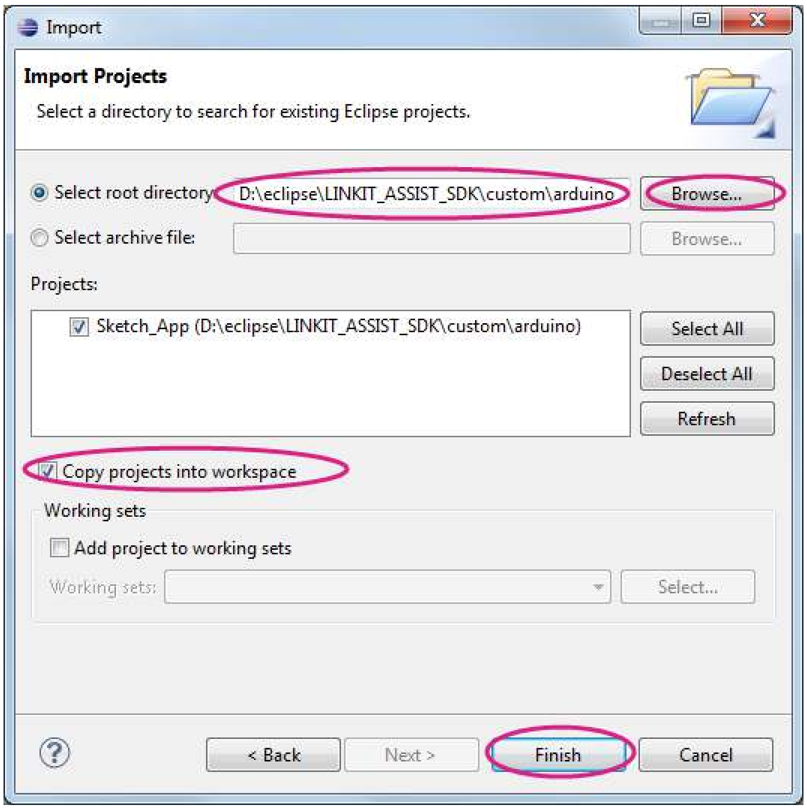


Figure 54 Selecting the import source

b) In the **Import Projects** dialog, next to **Select root directory** click **Browse** and find the `custom\Arduino` folder in LinkIt Assist SDK folder. Make sure the **Copy projects into workspace** is checked, as shown in Figure 55, then click **Finish**.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: D:\eclipse\LINKIT_ASSIST_SDK\custom\arduino

Select archive file:

Projects:

☑ Sketch_App (D:\eclipse\LINKIT_ASSIST_SDK\custom\arduino)

☑ Copy projects into workspace

Working sets

☐ Add project to working sets

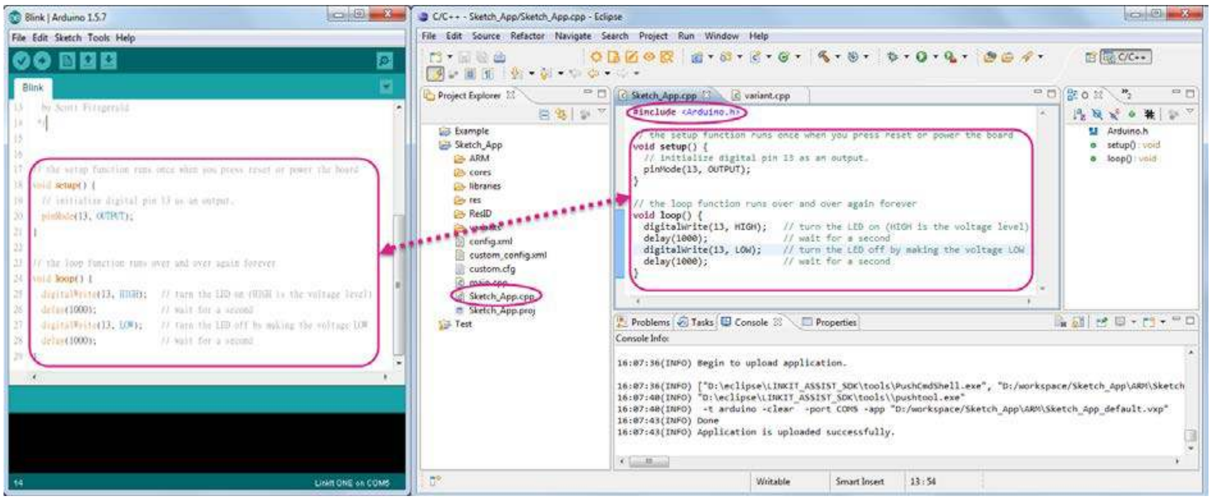
Working sets:

Figure 55 Selecting custom/arduino as the root directory

2) Import your Arduino sketch as follows:

a) In **Project Explorer** click the **Sketch_App** folder and open **Sketch_App.cpp**.

b) Copy and paste the content of your Arduino sketch into **Sketch_App.cpp**. Insert `#include <Arduino.h>` at the start of the `.cpp` file to include Arduino header files, this is because the Arduino IDE has this `#include` path by default but the Eclipse IDE doesn't. As an example, Figure 56 shows the Arduino Blink code being copied into the `.cpp` file. Alternatively, you can delete **Sketch_App.cpp** and replace it with your own sketch file by changing the suffix of your sketch file from `*.ino` to `*.cpp.`


```
#include <Arduino.h>

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

```
16:07:36(INFO) Begin to upload application.

16:07:36(INFO) ["D:\eclipse\LINKIT_ASSIST_SDK\tools\PushCmdShell.exe", "D:/workspace/Sketch_App\ARM\Sketch
16:07:40(INFO) "D:\eclipse\LINKIT_ASSIST_SDK\tools\\pushtool.exe"
16:07:40(INFO)  -t arduino -clear  -port COM5 -app "D:/workspace/Sketch_App\ARM\Sketch_App_default.vxp"
16:07:43(INFO) Done
16:07:43(INFO) Application is uploaded successfully.
```

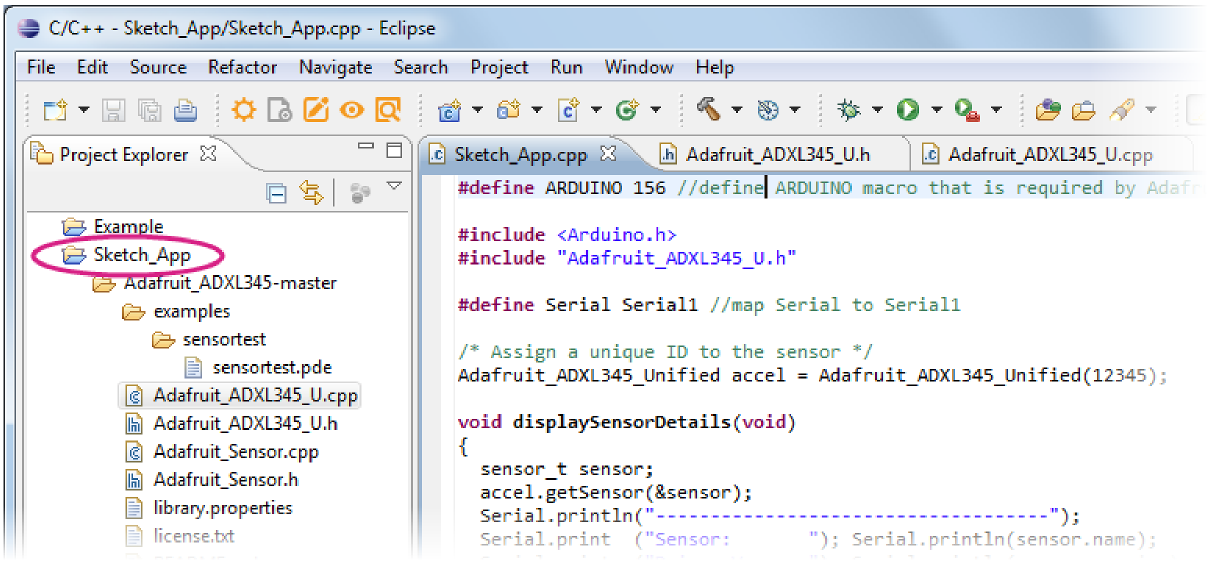
Figure 56 Importing Arduino sketch code by copying it

c) You may need to revise your pin configuration and mapping from that used for the LinkIt ONE development board. You can refer to the file `variants\linkit_one\variant.cpp`, which includes definition of the pin mappings, and update your imported Sketch code accordingly. This mapping is responsible for mapping Arduino pin numbers to the DCL pin numbers used in the LinkIt Assist 2502 SDK 2.0.

d) Click the Build Application toolbutton, and upload your VXP.

3) If your Sketch requires additional Arduino peripheral drivers, import then as follows

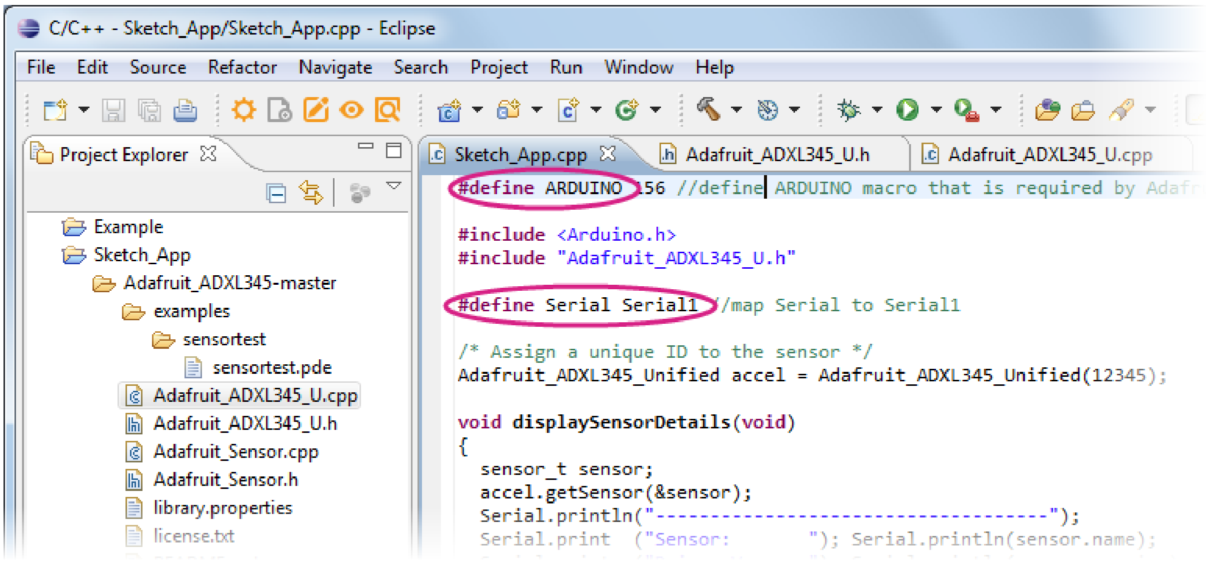
a) Open **Windows Explorer** and copy your Arduino driver library into the **Sketch_App** folder of the imported project's workspace, such as `\workspace\Sketch_App`. In **Project Explorer** open the shortcut menu on **Sketch_App** and click **Refresh** to update the project. Figure 57 shows an example of an ADXL345 driver being imported.



```
#define ARDUINO 156 //define ARDUINO macro that is required by Adafr

#include <Arduino.h>
#include "Adafruit_ADXL345_U.h"

#define Serial Serial1 //map Serial to Serial1

/* Assign a unique ID to the sensor */
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
```

Figure 57 Importing peripheral drivers

b) Depending on the driver, you need to add or modify the `#define` statements required by the library. In the ADXL345 example shown in Figure 58, it requires an Arduino macro that isn't present in the LinkIt Assist 2502 SDK, so you need to add it manually in the Sketch_App.cpp.



```
#define ARDUINO 156 //define ARDUINO macro that is required by Adafr

#include <Arduino.h>
#include "Adafruit_ADXL345_U.h"

#define Serial Serial1 //map Serial to Serial1

/* Assign a unique ID to the sensor */
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
```

Figure 58 Adding the required #define statements

Please note that the Arduino sketch is executed in a thread created in `vm_main`. Please see `main.cpp` in the `Sketch_App` folder for details.

Since the Arduino sketch isn't the main thread, you shouldn't invoke the LinkIt Assist 2502 API directly in the Arduino thread, but instead you should implement synchronization mechanisms between the Arduino thread and main thread according to the software's requirements.

5. API Guide

This chapter introduces the LinkIt Assist 2502 API, module-by-module. It briefly covers the functionality and limitation of each module. For detailed API descriptions, please refer to the API Reference on the MediaTek Labs website.

API functions are exported in C header files and prefixed with `vm`, for example: `vm_malloc`. Similarly, the module names are also prefixed with `vm`, for example: `vm_audio_get_volume`.

5.1. Basic Types

LinkIt Assist 2502 SDK 2.0 runs on ARM-based hardware, therefore the underlying machine word is in little endian and the default integer bit-width is 32 bits.

The majority of general types are defined and prefixed with `VM`, for example: `VMINT`, `VMCHAR`, and `VMSTR`. Others include char, short, int, long and long long. Float and double types are also supported through a software floating pointer calculation library. NULL-terminated strings are defined as `VMSTR` and `VMWSTR`.

Note: as floating point computations are implemented in software, the performance is not as good as a hardware-based implementation would be. Therefore it's recommended that floating-point operations are avoided when possible.

5.2. Return Values

Many LinkIt Assist 2502 functions return `VM_RESULT` as the return type.

For your convenience, LinkIt Assist 2502 SDK 2.0 provides 2 macros: `VM_IS_SUCCEEDED` and `VM_IS_FAILED`, which can be used to determine if a function call succeeded or failed.

When you see a function return type `VM_RESULT`, you can use `VM_IS_SUCCEEDED (x)` or `VM_IS_FAILED(x)`, x being the return value of the function.

Please keep in mind that although generic success values such as `VM_SUCCESS` and `VM_OK` are defined, you should avoid comparing these success values directly. This is because some APIs may return different values that represent different success scenarios. Therefore, instead of writing the code for correct functional logic like this:

```
if(VM_OK == vm_api_function())
   do_success_case();
```

You should do this:

```
if(VM_IS_SUCCEEDED(vm_api_function())
   do_success_case();
```

Note: the above applies only to returning values in `VM_RESULT` type. For other return types such as handles or memory pointers, please refer to their corresponding API reference for error handling.

For return values in `VM_RESULT`, detailed error codes and other success codes are defined as enum values in each module respectively. Please refer to the API documentation if you need to process specific error codes or success scenarios.

5.3. Log

LinkIt Assist 2502 SDK 2.0 provides a log module that sends debugging and error messages generated by the application to the LinkIt Assist 2502 SDK Monitor tool (for more information on how to use the Monitor tool, please see 2.9.4, "Monitor". There are different log functions for different levels of error severity such as FATAL, ERROR, WARN, INFO and DEBUG. The severity levels from highest to lowest are in this order: FATAL > ERROR > WARN > INFO > DEBUG. You can also control the severity filter in the Monitor tool.

To generate a log in the application, you can use the following APIs:

- For debugging information:

  ```
  void vm_log_debug(char* fmt, ...);
  ```

- For information on an APP execution entering a function:

  ```
  void vm_log_info(char* fmt, ...);
  ```

- For warning information of an error which can be resolved by an APP. For example, a program fails to connect to the Internet due to delinquent phone bills, or bad signal coverage. After the bill is paid or moving the phone to a location with a stronger signal, the APP will remove the problem and it will no longer exist.

  ```
  void vm_log_warn(char* fmt, ...);
  ```

- For failed function error messages, but the APP doesn't terminate. For example, the human voice feature fails in an English learning APP, but other features are still working as the APP is still operating.

  ```
  void vm_log_error(char* fmt, ...);
  ```

- For fatal error messages that causes an APP termination. For example, the APP fails to allocate sufficient memory upon program start-up, rending the system unable to generate user interface hence resulting in program termination.

  ```
  void vm_log_fatal(char* fmt, ...);
  ```

Note: the length of each output string may not exceed 255 characters, and since the system has a limited buffer size for logs, it is possible to lose log strings if log functions are called too frequently.

5.4. Command

The Command module receives input commands from the Monitor tool. You can send command to the development board by selecting the **send command** function from Monitor tool (please see 4.6, "Log and Command" for how to send command with the Monitor tool for more information). The command format is `command_string`.

To process inputs from different ports, you can register a specific port by calling `vm_cmd_open_port()`. The command strings are then passed to the callback function provided to `vm_cmd_open_port()`. The max length of each command string is 512 bytes. Many of the example code provided in the SDK use the Command module to trigger example routines, please refer to examples for usage of the Command module.

5.5. Standard Library

The standard library covers core programming elements, including: memory allocation, thread creation, timer callbacks, date time service, string operation, file system and its application data, log service and character set conversion functions.

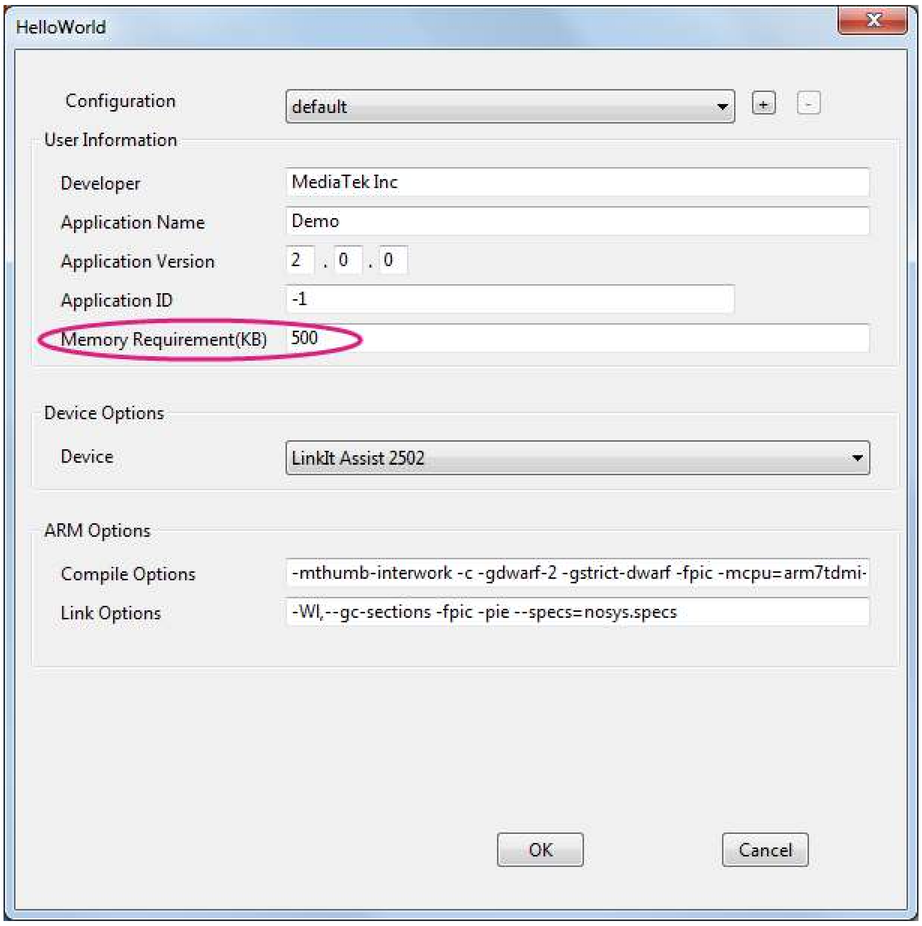
5.5.1. Memory

The Memory module provides memory allocation functions that allow you to allocate chunks of memory from the heap of the application. Instead of allocate and free memory using standard C functions like `malloc` and `realloc`, LinkIt Assist 2502 has its own memory management interfaces. Please use `vm_malloc`/`vm_realloc` to request memory and `vm_free` to free memory.

ARM processors are very strict about byte alignment requirements. If it’s necessary to access the content of 4 bytes in one step, then the starting address of that content must be positioned on the 4-byte boundary. Similarly, if it’s necessary to access the content of 2 bytes in a single step, then the starting address of that content must be positioned on the 2-byte boundary and so on. Failure to do so will cause an exception to occur.

Alignment problems occur mainly with incorrect pointer conversion, such as converting a byte pointer into a structure pointer. This type of pointer conversion should be avoided. The pointer returned by `vm_malloc` must be typecast and the pointer returned by `vm_malloc`, regardless of the size of its allocated space, must be aligned properly.

The maximum possible size that can be allocated is determined by the size of the heap, as shown in Figure 59. The size of the memory heap of a software is configured statically when building the application, as described in 5.5.1, "Memory". You should configure the total memory requirement in Application Settings. See 2.9.1, "Application Settings" for the Application Settings interface. The total memory requirement of a software consists of two parts, the load size and the heap size. If the total memory size is larger than then free memory size of the system, the application fails to load and there will be error logs on the monitor tool.


HelloWorld

Configuration	default
User Information	
Developer	MediaTek Inc
Application Name	Demo
Application Version	2 . 0 . 0
Application ID	-1
Memory Requirement(KB)	500
Device Options	
Device	LinkIt Assist 2502
ARM Options	
Compile Options	-mthumb-interwork -c -gdwarf-2 -gstrict-dwarf -fpic -mcpu=arm7tdmi-
Link Options	-Wl,--gc-sections -fpic -pie --specs=nosys.specs

Figure 59 Application Settings

On LinkIt Assist 2502 SDK 2.0, the system firmware reserves around 1.7 MB of RAM for application usage. However, the exact values will change between different firmware for different hardware, and it will also change when firmware version changes. Therefore, it is suggested that you query the `VM_FIRMWARE_HOST_MAX_MEM` with the firmware API `vm_firmware_get_info` to determine a proper total memory size of the application before configuring it.

If the system cannot provide enough heap memory at the time of allocation, allocation failure will occur. The application must determine and act on the value returned from `vm_malloc`. If the allocation fails, appropriate error handling must be performed, and when `vm_free` is used to release memory, it is also necessary to ensure that the address of the memory to be released is correct. In addition, the memory requested by the application must be freed by the application itself. Otherwise exceptions will occur due to memory leak.

One thing to keep in mind is that if the application calls the `vm_malloc` and `vm_free` functions frequently with very small memory chunks, it is possible that memory fragmentation will occur. When fragmentation happens, it is possible to have allocation failure even though the total remaining free memory is sufficient, it may be too scattered and no adequate contiguous memory block is available for allocation.

For memory that will be accessed by hardware, LinkIt Assist 2502 SDK provides an API, `vm_malloc_dma`, to allocate non-cacheable memory (dma stands for Direct Memory Access). By allocating non-cacheable memory, the hardware and software can access the memory buffer without the need of updating or flushing CPU caches. Therefore, it is important to call `vm_malloc_dma` to allocate memory that will be passed to hardware drivers, for example the buffers used by the Graphics module in 5.17, "Graphics".

5.5.2. String and Character Set

While it is common to process ASCII strings, for applications that has localization/globalization specifications, it is important to process Unicode strings. In the SDK, the `VMSTR` type denotes an ASCII encoded string; while the `VMWSTR` type denotes an UCS2 encoded string. It is very common to see UCS2 encoded string in the SDK, especially in file paths. All the string parameters that denotes folder/file path must be UCS2.

LinkIt Assist 2502 provides basic String Converting functions between UTF8, UTF16, UCS2 and ASCII encoding format functions. The strings being converted must be NULL-terminated. Note that in the case of UCS2 the terminating NULL is 2 bytes long. Because the memory is allocated by caller, there is no memory alloc/free action internally. However, you should keep in mind that for strings that are file paths, it should not exceed 255 characters. In other words, the maximum string length included NULL should not exceed 256 characters.

5.5.3. Timer

Timer functions accept a time parameter and a callback function. The callback function will be called when the designated time period has passed.

The SDK provides three types of timer: precise, non-precise, and HISR timer. Precise timer provides higher precision than non-precise timers, but precise timer stops when the system enters power saving mode; Non-precise timers can wake the system from power saving mode, but it may introduce variance to the timer callback interval. The maximum possible variance is 254 system ticks, where 1 tick equals 4.615 ms.

It's good to keep in mind that a LinkIt ONE firmware is configured to never enter power saving mode, therefore you don't need to keep the system from entering power saving mode in order for precise timers to work.

The function `vm_timer_create_precise` can be used to create precise timer; while the function `vm_timer_create_non_precise` can be used to create non-precise timer. The minimum precision of the timer is 10ms. When a timer is created, the system will invoke the timer handling callback function periodically until the timer is deleted by the application. Since the next round of time-keeping operations will not commence before a timer handling function has completed its execution, and since the timer handling functions for other timers will not be executed before the timer handling function of the current time has been completed, timer callback involves certain time delay. The more timers, the more complicated the timer handling functions, and the longer the accumulated time delay. Also note that when the program exits, the application must delete all timers. Therefore, if you need your application to continue running when the system enters power saving mode, create timers with `vm_timer_create_non_precise` instead.

The HISR timer invoked is very precise and not affected by the power saving mode, however, due to the nature of its execution context, the callback function of HISR timers cannot invoke any LinkIt Assist 2502 API, and it should only be used to perform fast value updates. Failing to do so may result in critical system failure since critical drivers that require real-time update such as GSM modem, will fail.

5.5.4. Date and Time

The Date and Time module provides functionality to query and set the current system time with precision up to seconds, and also provides functions to measure a shorter period of time with precision up to micro seconds.

There are three sets of date and time functions. The first one is the date time API, which gives a structure describing years, months, days and time. You should use . `vm_time_set_date_time` API to set and get the current system time.

The second set is the UNIX time API called `vm_time_get_unix_time`, which returns the number of seconds since UNIX epoch according to current system time.

To measure high-precision duration, such as for graphics animations, use the microsecond timer (US), `vm_time_ust_get_count`, to retrieve the microsecond time from system power on until now. Then you can pass two counter values to `vm_time_ust_get_duration` to calculate the time differences between two counters.

Note: `vm_time_ust_get_count` function can handle counter value overflow and thus is safer than to subtract counter values.

5.5.5. Threads

LinkIt Assist 2502 development platform is a multi-threaded environment. An application in the LinkIt Assist 2502 platform runs on the main thread in the system. An application can call `vm_thread_create()` to create a sub-thread, with a specified priority. The system allows the maximum of 10 sub-threads to be created by the `vm_thread_create()`. The specified priority is just a proposed priority. The system will assign an actual priority (based on the proposed priority) accordingly to the sub-thread based on the current system resource.

If the proposed priority is equal or less than 128, then the actual priority assigned by the system will be lower than the main thread. If the proposed priority is equal or greater than 129, then the actual thread priority assigned by the system will be higher than the main thread. If the proposed priority is 0, the system will assign the default priority to the thread, which is lower than priority of the main thread.

A thread can create a signal by calling `vm_signal_create()`, and then post the signal by calling `vm_signal_post()`, for thread synchronizations. The mutex functions are also provided in the Thread module for sharing data among threads safety.

One thread can send asynchronous messages to another thread through `vm_thread_send_message()`, or receive messages sent from another thread through `vm_thread_get_message()`. To send a message, you need to define a message ID and make sure the message ID is unique among the threads. The valid range of message ID is 1,000 to 60,000. Then the message ID and accompanying user data can be sent to another thread by querying the thread handle with `vm_thread_get_current_handle()`. There is a specialized API for created threads to retrieve the handle to the main thread called `vm_thread_get_main_handle()`.

An important exception is that when sending messages to the main thread, the message is actually redirected to the system event handler registered by `vm_pmng_register_system_event_callback`. In this case, there is no need to call `vm_thread_get_message` in the main thread. Instead, you should process the message ID in the system event handler.

5.5.6. Using C Standard Library Functions

An application in the SDK does not require C standard library to execute. You may however choose to program with and link to the C standard library. Yet, not all functions in C standard library are supported. The application will fail to start when an unsupported function is used. The following contains a list of functions that are not supported. These include file operation functions, assertion functions and memory operation functions such as:

- File operation interfaces: `fopen / fread / fwrite`.
- Abort and exit interfaces: `abort / exit`.
- Character access interfaces: `getchar / getc / putchar / putc`.
- Input display interfaces: `scanf / printf`.
- Assertion and debug interfaces: `ASSERT / TRACE`.

Other platform neutral functions can be compiled, linked and executed, for example `rand()` and `srand()`.

5.5.7. File System

The SDK supports FAT32 file system. The maximum supported length of a path is 256 characters, including the terminating NULL. The encoding format of path name is UCS2.

There are several partition drives defined by the SDK. These drives are accessed by drive letters in the file path. For internal storage, `vm_fs_get_internal_drive_letter` will query what the drive letter is; for external storage such as the SD card, `vm_fs_get_removable_drive_letter` will query what the drive letter is. Note that some HDK does not come with SDK card slot. In this case, the firmware does not support external (removable) drive.

The concept of current file directory doesn't apply in the SDK. When reading and writing a file, an absolute pathname must be used. While the SDK does support opening multiple files at the same time, it is not recommended because of memory resource constraints. A file that is no longer in use should be closed promptly, or it's possible that opening another file may fail due to the lack of available file handle resources. In addition, do not open a file within the message handling logic of `VM_EVENT_QUIT`, or the operation will fail.

When reading or writing a file, proper error handling must be performed for any errors that may occur, or it will result in an exception. For error codes, please refer to `VM_FS_RESULT`.

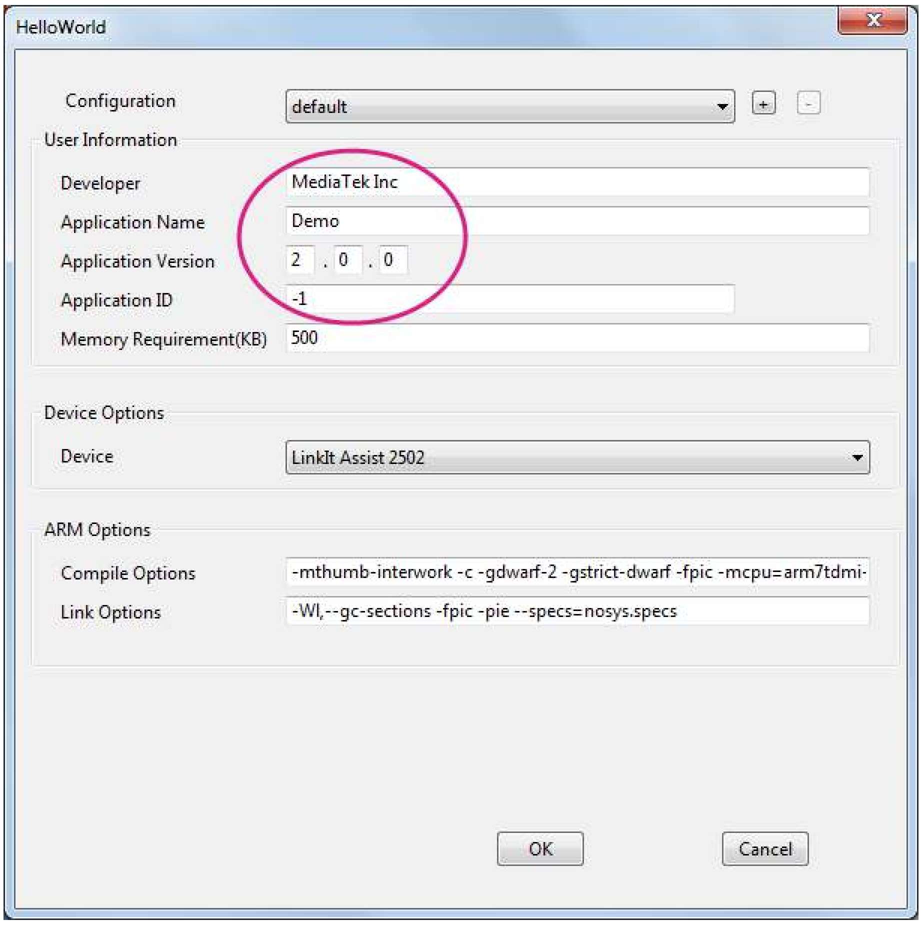
5.5.8. App data

Although it is convenient to use File System to store data for applications, sometimes you need to store data in a place where it's inaccessible from using Mass Storage mode of the HDK. App data is a set of File System APIs that can operate on files in a space located in hidden partition (system drive partition), which cannot be accessed in USB mass storage mode and is invisible to the end user. The data size is limited to 2KB. Note that the partition is not encrypted, just invisible to end users. Generally speaking, the data stored for the App should be kept minimal and when necessary, with extra encryption. The App data relies on the Application ID to index to the current partition; therefore it is important that you set the Application ID in the Application Settings to a specific value to access the same App data. The Application ID can be retrieved by the Tag module, which is described in the next section.

5.6. Tags

The Tag module enables you to embed certain string and integer information into the software, including **Developer, Application Name, Application Version** and **Application ID**. These strings and integers embedded to the vxp executables are called tags.

To define tag information, use the Project Setting in the Eclipse plug-in, as shown in Figure 60.



Configuration: default

User Information

Developer: MediaTek Inc
Application Name: Demo
Application Version: 2 . 0 . 0
Application ID: -1
Memory Requirement(KB): 500

Device Options

Device: LinkIt Assist 2502

ARM Options

Compile Options: -mthumb-interwork -c -gdwarf-2 -gstrict-dwarf -fpic -mcpu=arm7tdmi-
Link Options: -Wl,--gc-sections -fpic -pie --specs=nosys.specs

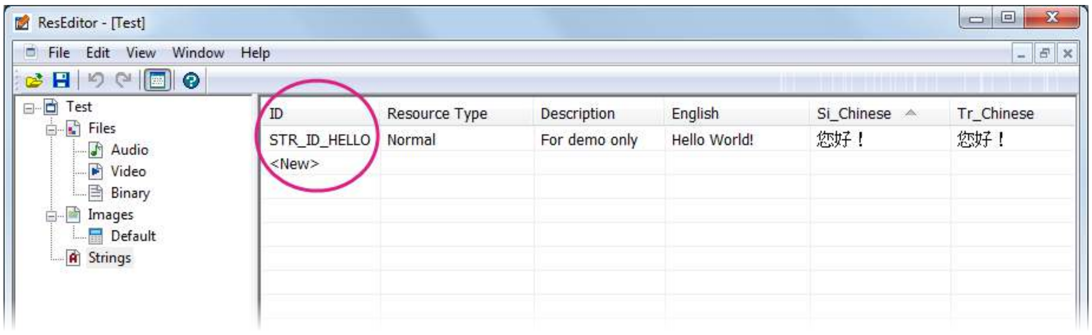
Figure 60 Tag settings in Application Settings

To retrieve the tag information in an application, call API function `vm_tag_get_tag`. Since there is no specific requirement on the length of the tag, the API allows you to pass NULL to the buffer parameter and the function will output required buffer length for given App ID, including the terminating NULL, for example:

```
VMWSTR app_name = NULL;

// passing NULL to retrieve tag size.
vm_tag_get_tag(NULL, VM_TAG_ID_APP_NAME, (void*)NULL, &tag_size);
// allocate buffer according to tag size
app_name = vm_malloc(tag_size);
// copy the tag string into the buffer.
vm_tag_get_tag(NULL, VM_TAG_ID_APP_NAME, app_name, &tag_size);
```

5.7. Resource

An application can call classes in the Resource module to load resources added in the resource editor. Each resource is identified and accessed by an identifier, see Figure 61.



ID	Resource Type	Description	English	Si_Chinese	Tr_Chinese
STR_ID_HELLO	Normal	For demo only	Hello World!	您好！	您好！
<New>					

Figure 61 Examples of resource identifiers

Please note the following restrictions on the naming of resource identifiers:

- The ID length cannot be greater than 64.
- The initial character cannot be 0-9.
- All characters must be A - Z, a - z, 0 - 9, or '_'.

To work with the Resource module, the application must initialize the resource first with `vm_res_init()` with desired language ID. The language ID maps to the language ID in the Language window of Resource Editor, for example:

```
/* load the "Tr_Chinese" language strings */
vm_res_init(886);
```

And then classes such as `vm_res_get_string` and `vm_res_get_image` can be used to retrieve the resources.

The application will need to free the memory in the stored resource data with `vm_res_delete`, with the exception of string resources, which is loaded into the memory during resource initialization and unloaded when the resource module terminates, for example:

```
/* release the resource occupied by the image resource */
vm_res_delete(IMG_ID);
/* unload all resources from memory, including string resources */
vm_res_deinit();
```

There are several reasons that causes resource loading to fail, some of them are listed below:

1) The path specified in the LinkIt Assist 2502 Resource Editor is incorrect, and therefore it failed to pack the resource into the application.
2) Query resources before initializing them. Make sure you call `vm_res_init` before querying resources, and call `vm_res_delete()` when resources are no longer being used.
3) Insufficient memory or insufficient file handle. This can be caused by other parts of the application that are consuming too much memory or opening too many file at the same time.
4) Loading resources within the message handling logic of `VM_EVENT_QUIT`. It is not possible to load resources when the application is being destroyed.

5.8. Process Control

The System module provides core functionalities for LinkIt Assist 2502 application. When an executable is loaded into memory, it becomes a process. The process can then register a system event handler to respond to system events such as entering and exiting the application. There are also APIs to query process information such as the path to the executable file, and the status of the process.

There are also APIs to assist the upgrade of application executables, the framework would terminate the running application, check and update the application executable, and launch the program with a single API call. Please refer to `vm_pmng_exit_and_update_application` for details.

5.9. Firmware

The Firmware module provides firmware-related APIs to query firmware information like version and build time. It also provides API to query if a specific functional module such as Wi-Fi is supported by the current running firmware.

In addition, an API `vm_firmware_trigger_update` is provided to trigger the firmware update process from LinkIt Assist 2502 application. The firmware update package is released along with each SDK release, and the application should implement its own transfer mechanism to transfer the firmware update package to the internal storage of the hardware and trigger the update process. Refer to the API reference and also examples `Framework_Firmware_FirmwareUpdate` in LinkIt Assist 2502 SDK for usage example.

5.10. Driver API

The Driver API provides ability to connect to external peripherals. Peripherals are connected to LinkIt ONE or LinkIt Assist 2502 HDK through hardware pins. Different HDKs have different layouts and functionalities on its breakout pins. For example, on LinkIt ONE, a set of digital pins called D0, D1, D2 and alike are defined, while on LinkIt Assist 2502 board they are called P0, P1, P2 and alike.

Also, the same break out pin may support multiple functionalities, for example the D0 pin of LinkIt ONE is also capable of working as an UART serial port.

To support different hardware and different pin names, a Device Common Layer (DCL) is provided to access peripheral functions in a set of common APIs. All the hardware interfaces like UART, GPIO, ADC, I2C, SPI, and EINT can be accessed through the same set of functions with different parameters, such as: `vm_dcl_open`, `vm_dcl_close`, `vm_dcl_control`, `vm_dcl_read` and `vm_dcl_write`.

The DCL module identifies each pin with a pin number, and then the SDK provides a header file `vmboard.h` to map pin names printed on the board to the actual pin numbers used by the DCL. You can include this header file and simply refer to pin names printed on the board, which is defined to the actual pin identifiers through macro, for example. You can also refer to the pin-out diagram for the mapping between names and numbers, as shown in Figure 62, where for example the "GPIO1" pin maps to pin number 12.

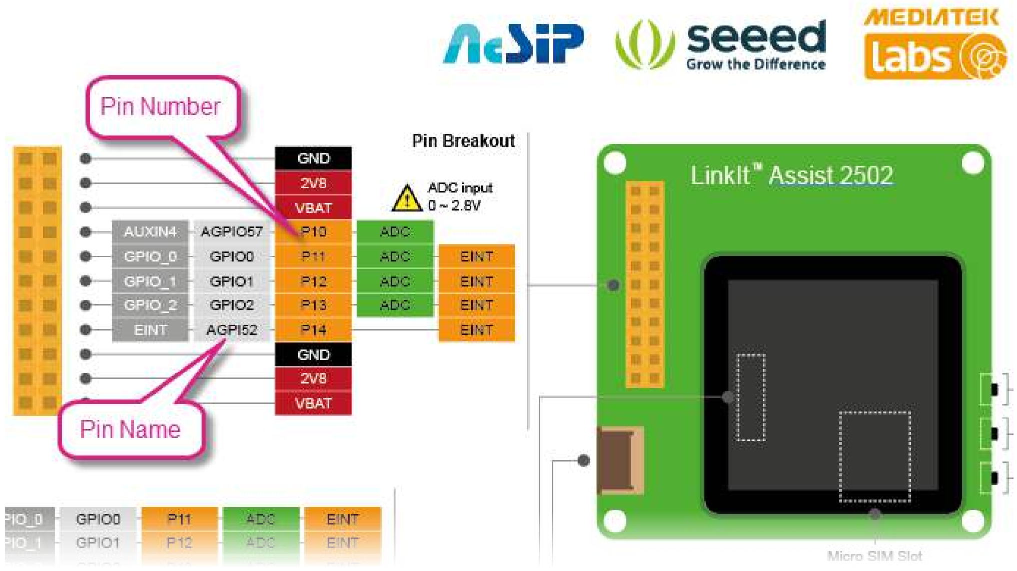


Figure 62 Mapping Pin Name to Pin Number

To access to different functionalities of a pin, the DCL modules provides an API `vm_dcl_config_pin_mode` to configure the functionality of a given pin number. For example, the following code configures the P8 pin on LinkIt Assist 2502 as a GPIO pin. Please see Figure 5 pin out diagram for location of P8.

```
vm_dcl_config_pin_mode(VM_PIN_P8, VM_DCL_PIN_MODE_GPIO);
```

To configure it as UART pin, please use the following codes. Note that UART protocol uses two pins (P8/P9), therefore both P8 and P9 require configuration:

```
vm_dcl_config_pin_mode(VM_PIN_P8, VM_DCL_PIN_MODE_UART);
vm_dcl_config_pin_mode(VM_PIN_P9, VM_DCL_PIN_MODE_UART);
```

Please refer to the HDK documentation for details of the functions supported by each pin. For some functions, the pins are referenced with different indices, for example PWM ID and ADC channel ID. In this case, you can use the following macro to convert pin numbers to function indices:

```
PIN2PWM(VM_PIN_P0)      /* Get PWM ID of pin P0 */
PIN2CHANNEL(VM_PIN_P10) /* Get ADC channel ID of pin P10 */
PIN2EINT(VM_PIN_P12)    /* Get EINT ID of pin P12*/
```

And then you can open a device with `vm_dcl_open` to retrieve the corresponding handle. This handle can then pass to `vm_dcl_read` and `vm_dcl_write` for generic read and write operations, or pass to `vm_dcl_control` which can send device-dependent command structures to the underlying hardware. Refer to each sub-module for available command structures.

Some hardware operations allow you to register callbacks, which are invoked upon specific hardware events. Please note that these callbacks are not executed in the hardware interrupt context, but are executed in the main thread instead. This ensures proper serialization of events but sacrifices realtime constraint. In the following sections you will learn briefly the capability of each sub-module.

5.10.1. ADC

ADC (Analog-to-digital converter) module is used to read the value of an assigned analog input pin, it can convert the input voltage value to discrete integers from 0 to 1023. The reference voltage depends on different HDK circuit design. Refer to HDK documentation for details.

5.10.2. GPIO

GPIO module enables you to set pins in HIGH and LOW voltage, and can act as either INPUT or OUTPUT. Before using a certain GPIO pin, you have to set up the pin mode of that pin first. The default mode is INPUT, in most cases, when you use GPIO pins, you have to combine many pins together, constructing a port to communicate with peripheral devices.

5.10.3. I2C

The I2C module supports I2C protocols to connect to peripherals. The SDK works as the I2C master. After setting up pin mode and opening the I2C device handle, call `vm_dcl_control()` with the following commands:

- Send `VM_DCL_I2C_CMD_CONFIG` command to set slave address, transaction mode and mode speed.
- Send `VM_DCL_I2C_CMD_SINGLE_WRITE/READ` or `VM_DCL_I2C_CMD_CONT_WRITE/READ`, or `VM_DCL_I2C_CMD_WRITE_AND_READ` to communicate with the slave devices.

5.10.4. SPI

The LinkIt Assist 2502 HDK work as SPI master and can connect to SPI slave devices. Pay extra attention to the slave configuration table because only when relevant parameter settings comply with the demand from the slave can the normal data communication between master and slave start. Before using a new slave, please make sure you:

- Access the SPI port with `VM_DCL_SPI_PORT1`.
- Check the settings in `VM_DCL_SPI_CONTROL_SET_CONFIG_PARAMETER` is “MSB first” or "LSB first" in `rx_msbf` and `tx_msbf` field.
- The polarity and phase settings of the clock in the fields `clock_polarity` and `clock_phase` is correct.
- Meet the required transmission speed of slave device.
- The SPI module works in DMA mode, the buffer for exchange data must be non-cacheable. Therefore, the API `vm_malloc_dma` should be used to allocate the buffer passed to SPI structure `vm_dcl_spi_control_read_write_t`.

5.10.5. Serial IO (UART)

The Serial Input and Output (SIO) module supports UART (Universal Asynchronous Receiver/Transmitter) protocol. The SDK supports both the UART port emulated by the USB and also the hardware serial device in the HDK. However, since the USB UART ports are used by the SDK itself for Monitor tool and Uploader tool, you should use the hardware serial device only. To access the hardware serial device, first step is to configure the pins to UART mode, for example:

```
vm_dcl_config_pin_mode(D0, VM_DCL_PIN_MODE_UART)
vm_dcl_config_pin_mode(D1, VM_DCL_PIN_MODE_UART)
```

Next, open the hardware serial device, which is `VM_DCL_SIO_UART_PORT1`, as in:

```
handle = vm_dcl_open(VM_DCL_SIO_UART_PORT1, vm_dcl_get_ownerid());
```

Then, configure the UART port by calling `vm_dcl_control` with `VM_DCL_SIO_COMMAND_SET_DCB_CONFIG` as command and `vm_dcl_sio_config_t` as parameter. Refer to the API reference for details.

Finally, you should register callbacks that are invoked when the read or the write buffer are available by calling `vm_dcl_register_callback` with events `VM_DCL_SIO_UART_READY_TO_WRITE` and `VM_DCL_SIO_UART_READY_TO_READ`, and call `vm_dcl_write / vm_dcl_read`.

5.10.6. PWM

PWM (Pulse Width Modulation) is used to convert discrete digital signals to a continuous analog signal for writing to an assigned pin. It can be used to light a LED at varying brightness. Both LinkIt ONE and LinkIt Assist 2502 support two PWMs, and they're identified by `VM_DCL_PWM_1` and `VM_DCL_PWM_4`.

Steps to control PWM are described below. In this example, two LinkIt Assist 2502 pins that support PWM are used – pins P0 and P1. First please make sure you configure the pin mode as PWM:

```
vm_dcl_config_pin_mode(VM_PIN_P0, VM_DCL_PIN_MODE_PWM);
```

Next, open PWM device handle:

```
pwm_handle = vm_dcl_open(PIN2PWM(VM_PIN_P0), 0);
```

Then, send the following commands by calling `vm_dcl_control` to configure the PWM module:

- `VM_PWM_CMD_START`
- `VM_PWM_CMD_SET_CLOCK`
- `VM_PWM_CMD_SET_COUNTER_AND_THRESHOLD`

And finally, use `vm_dcl_close` to close the handle if it's not in use.

An application of the PWM module worth mentioning is the backlight control of the display module on LinkIt Assist 2502 HDK. In this HDK the backlight is controlled by pin P1, which maps to `VM_DCL_PWM_1`. Therefore, if you open the device and set command `VM_PWM_CMD_SET_COUNTER_AND_THRESHOLD` with a value, the brightness of its backlight changes. If you set threshold to 0, then the backlight is turned off, if you set threshold to a number larger than 0, then the backlight will turn on, the larger the threshold, the brighter the backlight.

Note: the threshold number should be smaller than the counter number.

5.10.7. EINT

The EINT module allows applications to register callbacks to respond to external interruptions. The handler callbacks are executed in main thread instead of interrupt context, so there are no re-entrance to take into consideration. The external interruptions are triggered by voltage changes, including RAISING, FALLING, or EDGE trigger. Since the handler callbacks are executed in the main thread, it is possible that the speed of event generation is faster than the speed of the handler function. In this case, the excessive event may congest the main thread event queue, and lead to fatal error in the system. It is suggested that you enable the hardware debounce setting to prevent excessive EINT events. The shortest period of debounce setting is 1 ms. Please check LinkIt Assist 2502 SDK example code `DCL_EINT` for example usage flow.

5.10.8. PMU

The power management unit (PMU) module can be used to control the power of the vibrator pin. To control the vibrator pin, first open the PMU module as follows:

```
pmu_handle = vm_dcl_open(VM_DCL_PMU, 0);
```

and then, send `VM_DCL_PMU_CONTROL_LDO_BUCK_SET_ENABLE` command with `vm_dcl_control`, as in:

```
vm_dcl_pmu_ld0_buck_enable_t val;
val.enable = TRUE;
val.module = VM_DCL_PMU_VIBR;
vm_dcl_control(pmu_handle, VM_DCL_PMU_CONTROL_LDO_BUCK_SET_ENABLE, (void
*)&val);
```

You can also refer to example `DCL_PMU_Vibrator` for example of use.

5.11. Network

The SDK supports internet connection through Wi-Fi and GPRS. Instead of separated APIs, a set of common APIs are provided to allow connecting to the internet with Wi-Fi and GPRS networks. You should use Wi-Fi and GSM-GPRS module to configure the connectivity hardware, then use the Network APIs to create connections to remote servers. The following sub-modules are provided:

- **BSD sockets**: provides BSD socket compatible interfaces
- **TCP and UDP** module: provides both synchronous and asynchronous socket connections
- **SSL** module: provides SSL socket connection
- **HTTPS** and HTTP module: provides HTTPS/HTTP connection

5.11.1. BSD Sockets

BSD sockets (or Berkeley sockets) is a computing library with an API for internet sockets and UNIX domain sockets, used for inter-process communication (IPC). LinkIt Assist 2502 implements major parts of the BSD sockets APIs for internet connection. You can create sockets as server or client.

Before creating a socket, you have to configure the Wi-Fi or GPRS connectivity first. Refer to Wi-Fi module and GSM-GPRS module for details. To create a server socket with the BSD socket API, follow these steps:

1) Open Bearer

   The GPRS or Wi-Fi bearer must be opened before creating a socket connection, so that the BSD sockets knows which underlying connectivity hardware should be used. To open the bearer, call `vm _bearer_open` with the parameter *APN* indicating which kind of bearer will be used, it can be one of the following values:

   a) `VM_BEARER_DATA_ACCOUNT_TYPE_GPRS_NONE_PROXY_APN:` use the GPRS bearer without a proxy .

   b) `VM_BEARER_DATA_ACCOUNT_TYPE_GPRS_PROXY_APN:` use the GPRS bearer that has a proxy.

   c) `VM_BEARER_DATA_ACCOUNT_TYPE__WLAN:` use the Wi-Fi bearer.

   d) `VM_BEARER_DATA_ACCOUNT_TYPE_GPRS_CUSTOMIZED_APN:` use the customized APN.

   If the bearer is opened successfully `VM_BEARER_OK` will be returned, after which the application can create the socket connection. If `VM_BEARER_WOULDBLOCK` is returned, it means platform is opening the bearer and application shouldn't create socket connection until it receives the `VM_BEARER_ACTIVATED` event in the callback function.

2) Hold bearer (optional)

   For the GPRS bearer, platform will release the bearer automatically when the socket connection is closed. If application wants to keep the bearer for creating another socket connection, this can be achieved by a call of `vm_gsm_gprs_hold_bearer()`. By calling this API, the bearer will be held and reused. Application needs to call `vm_gsm_gprs_release_bearer()` when the bearer is not in use anymore.

3) Create socket connection

   The create socket connection API is same as the BSD sockets API where `socket()` is called to create a socket connection. Compared with BSD sockets, LinkIt Assist 2502 only supports PF_INET / PF_INET6 protocol family and SOCK_STREAM / SOCK_DGRAM socket type.

4) Bind socket

   `bind()` assigns a socket to an address. When a socket is created using `socket()`, it is only given a protocol family, but not assigned an address. This association with an address must be performed with the `bind()` system call before the socket can accept connections to other hosts.

5) Listen to socket

   After a socket has been associated with an address, `listen()` prepares it for incoming connections. However, this is only necessary for the stream-oriented (connection-oriented) data modes, i.e., for socket types (`SOCK_STREAM`).

6) Send or receive data through connection

   When a connection request from a client is coming, you can use `send() / sendto() / recv() / recvfrom()` functions to send or receive data through connection.

7) Close socket connection

   When the connection is not in use anymore, a call to `closesocket()` will close the connection and release the socket resource.

8) Release bearer

   If `vm_gsm_grps_hold_bearer` is not called, the bearer will be released automatically by platform, or else `vm_gsm_gprs_release_bearer` have to be called to release the underlying connectivity hardware.

To create a client socket, the open bearer and hold bearer process remains the same, as described below:

1) Open bearer: same with the server case.
2) Hold bearer (optional): same with the server case.
3) Create socket connection: same with the server case.
4) Setup socket connection

   After a socket has been created, application can set the socket options by a call of `setsockopt()`, a socket could be configured as working at asynchronous mode, which means the APIs will be returned immediately even if there are no data available. Application needs to use polling if it doesn't get the data completely.

   Application can use `connect()` to initiate a connection to a remote host specified by that host's address in the argument list.

5) Send or receive data through connection: same as the server case.
6) Close socket connection: same with the server case.
7) Release bearer: same with the server case.

All the applications share the socket resource, the maximum number of sockets can be created is 12, and may be further limited by other resource constraints. If there aren't any free sockets, an error code `VM_SOC_LIMIT_RESOURCE` will be returned by the API `vm_soc_socket()`.

The socket APIs are thread safe, and therefore can be called in threads created by thread APIs. Since some of the sockets API are synchronous, it is advised that you to call these blocking APIs in a separate thread other than the main thread, to prevent the event queue of the main thread from being congested.

5.11.2. TCP

The TCP module, as defined in header file `vmtcp.h`, is a set of APIs for creating TCP client and server with synchronous or asynchronous function calls. The maximum number of concurrent connections for TCP is 3.

This API allows you to assign underlying connectivity hardware by using the data account parameter during connection, for example:

```
vm_tcp_connect_sync(URL, Port, VM_BEARER_DATA_ACCOUNT_TYPE_WLAN)
```

So you don't not have to call separate bearer APIs as in BSD socket module.

Using the synchronous APIs, application will be blocked until the result is received even if the operation takes a long time. This set of API is thread-safe and can be called in created threads. Refer to `Network_TCP_Sync` example.

Using asynchronous APIs, the application will register a callback function and the user will not be blocked until the result is obtained. When the result is received, the callback function will be called with a specified event. Refer to the example `Network_TCP_Async` for example usage.

Note: the asynchronous APIs is NOT thread safe. It can only be invoked in main thread. Do not call Asynchronous APIs in created threads.

5.11.3. UDP

The UDP module comes with asynchronous APIs only. These APIs can only be called in the main thread. The Application can send and receive data through these APIs. Using UDP to send or receive data involves the following steps:

- Create a UDP connection with `vm_udp_create()`. A callback function should be registered.
- Handle the UDP events in the callback function.
- Send data by calling `vm_udp_send()` after receiving `VM_UDP_EVENT_WRITE`.
- Send data by calling `vm_udp_receive()` after receiving `VM_UDP_EVENT_READ`.
- When connection is broken, `VM_UDP_EVENT_PIPE_BROKEN` is received.
- When the connection is closed, `VM_UDP_EVENT_PIPE_CLOSED` is received.
- Close the connection once it is no longer needed, using `vm_udp_close()`.

5.11.4. SSL

The SDK comes with a set of SSL APIs based on XySSL SSL library interface and implementation. An example usage flow of these SSL APIs are as follows:

- Prepare the certificate. The SDK does not have a certificate pre-installed. You should provide a certificate file in X.509 format (.cer files) which can be exported from most web browsers.
- Use `vm_ssl_connect()` to connect to the host. This is an asynchronous API and the resulting event will be passed to the registered callback.
- Load the certificate file with `vm_ssl_load_ca_chain_certificate()`.
- Use `vm_ssl_read()` or `vm_ssl_write()` to read and write data after the security connection is established
- Close the connection by using `vm_ssl_close()`.

Note: the SSL APIs can only be executed in the main thread.

5.11.5. HTTPS

The SDK comes with HTTPS protocol support, defined in `vmhttps.h`. Ordinary HTTP methods like `GET/HEAD/POST/PUT/DELETE` are also supported – it is simply a degenerated case of the HTTPS protocol with `VM_HTTPS_PROTOCOL_HTTP` protocol setting.

The main flow of using HTTPS is as follows:

1) Register the callback function by calling `vm_https_register_context_and_callback()`.
2) Configure a dedicated channel for HTTPS requests by calling `vm_https_set_channel()`.
3) Send https request by using `vm_https_send_request()`.
4) Read the content by using `vm_https_read_content()`.

There are callback functions for each API, detailed in Table 5.

Callback function	API	Description
vm_https_set_channel_response_callback	vm_https_set_channel	This callback will be called when channel is opened
vm_https_unset_channel_response_callback	vm_https_unset_channel	This callback will be called when channel is closed
vm_https_release_all_request_response_callback	NA	
vm_https_wps_termination_callback	vm_https_set_channel	This callback will be called if there is abnormal when opening the channel.
vm_https_send_response_callback	vm_https_send_request	This callback will be called when there is data can be read for the first time after sending request.
vm_https_read_content_response_callback	vm_https_read_content	This callback will be called when there is data can be read.
vm_https_cancel_response_callback	vm_https_cancel	This callback will be called when the request is cancelled.
vm_https_status_query_response_callback	NA	

Table 5 HTTPS callback functions

5.12. Wi-Fi

LinkIt Assist 2502 provides a series of APIs to support Wi-Fi features, with these APIs, an application can perform following Wi-Fi operations:

- Wi-Fi control: scan and connect to a Wi-Fi access point.
- Smart Connection: listen to Smart Connection configuration broadcasts. These configuration broadcasts can be broadcasted from Smart Phones with the accompanying Smart Connection library on iOS and Android. The Smart Connection is located in the **Custom** folder of the SDK folder.

5.12.1. Wi-Fi control

Call `vm_wlan_mode_set()` in order for your application be set to work in station(STA) to connect to access points, or `VM_WLAN_MODE_OFF` to disable the Wi-Fi radio. The application can perform following operations when in station mode.

1) Scan AP list: up to 16 APs can be scanned. The exact maximum number may be different in different hardware platforms.
2) Connect to an AP: application can connect to an AP with the SSID / auth mode / password.
3) Get information, such as MAC address / IP address / netmask settings.
4) Enable WLAN roaming: The WLAN roaming enables the station roaming between APs in an AP group.

5.12.2. Smart Connection

Smart connection is a feature that enables end user to set AP connection dynamically from a smart phone application. This application installed in smart phone will broadcast the AP information using the Smart Connection protocol. A set of library and example applications are provided in the SDK in the **Custom** folder.

To listen to Smart Connection configurations, the application should set the Wi-Fi to STA mode and enter the *sniffer* mode by calling `vm_wlan_sniffer_on`. The application will then receive the configuration information in the callback function with `cb_type` as `VM_WLAN_SNIFFER_ON_AP_INFO`. Subsequently, the application can use this information to connect the corresponding AP.

5.13. GSM

The GSM module allows you accessing 2G connectivity features, including:

- SIM card information
- Cellular information
- High-level telephony operations
- GPRS data connectivity

These APIs are defined in header files `vmgsm_*.h`. For example, the SIM card information module is defined in `vmgsm_sim.h`. The developer has to prepare a valid SIM card which is valid for corresponding operations like GPRS data plan and SMS plan. Insert the SIM card into the SIM card slot on the HDK before booting up the device.

Note: the default status of the GSM modem is on. To switch the GSM modem off or on, call the `vm_gsm_switch_mode` function. Please refer to `vmgsm.h` for details.

5.13.1. SIM Card

The SDK provides a number of SIM card-related API calls to obtain relevant SIM card status and information, for example, IMEI can be retrieved by calling `vm_gsm_sim_get_imei`. One thing to notice is that the module is designed to be compatible with hardware with multiple SIM cards, and the SIM card identifiers starts from `VM_GSM_SIM_SIM1` instead of 0.

5.13.2. Telephony

The SDK provides high-level APIs for applications to make or receive phone calls, defined in `vmgsm_tel.h`. Note that the Audio hardware is occupied when make or receive phone calls, and therefore the Audio API will be interrupted by the Telephony APIs.

5.13.3. Cell Information

The GSM Cellular information module allows you to retrieve information about base stations, including:

- ARFCN - Absolute Radio Frequency Channel Number.
- BSIC - Base Station Identity Code.
- MCC - Mobile Country Code.
- MNC - Mobile Network Code.
- LAC - Location Area Code.
- CI - Cell ID.
- Received signal levels. Refer to header file `vmgsm_cell.h` for detailed `structure` information. To retrieve information, call `vm_gsm_cell_open` to allocate required resources for the module. Then
- `vm_gsm_cell_get_current_cell_info` can be used to retrieve info about current cell.
- `vm_gsm_cell_get_neighbor_cell_info` can be used to retrieve info about neighboring cells.

Since the cell information may be updated, a system event `VM_EVENT_CELL_INFO_CHANGE` will be dispatched to the system event handler when there is a change in the cell info.

To stop querying cell information, call `vm_gsm_cell_close`.

5.13.4. SMS

The SDK provides several high-level functions for SMS, including: read/send/cancel send/create and obtain SMS center number. The SMS module is not available immediately after the system boots up. Use the API `vm_gsm_sms_is_sms_ready` to retrieve the availability of the service. After the module is ready the developer can send messages or save messages into storage - SIM card or T-card. It can delete the specified one message or the specified box type messages through delete API. These messages exist after reboot.

Note: the cancel send API `vm_gsm_sms_cancel_send()` can be used only after send SMS API `vm_gsm_sms_send` is called and before send callback is invoked.

5.13.5. GPRS

The GPRS is a sub-module of the GSM module and it provides several APIs to configure the GPRS function:

- Hold / Release bearer

  As mentioned in 5.11.1"BSD Sockets", by holding the bearer, the bearer can be reused for different socket connections. If the application has to issue multiple connections subsequently in a short period of time, it is more efficient to hold the bearer.

- Set customized APN

  LinkIt Assist 2502 platform preloads many default APNs for different operators, but the firmware can't cover all the operators, so it allow application set its own APN by providing the API `vm_gsm_gprs_set_customized_apn_info()`, a parameter `data_account` should be passed to this API, it can be generated by calling `vm_gsm_gprs_get_encoded_data_account_id()`.

5.14. Bluetooth

The SDK supports both Bluetooth 2.1 and 4.0. In this version, the following sub-modules are supported:

- Connection Manager: This module controls the Bluetooth radio on/off, and device discovery and pairing of Bluetooth 2.1 devices.
- Profile SPP: This module support Serial Port Profile of Bluetooth 2.1.
- Profile GATT: This module supports GATT profile of Bluetooth 4.0.

5.14.1. Connection Manager

The Connection Manager defined in `vmbt_cm.h` is the first API you need for Bluetooth services. The module must be initialized by calling:

- `vm_bt_cm_init`

To power on / off the Bluetooth module, use following APIs:

- `vm_bt_cm_switch_on`
- `vm_bt_cm_switch_off`

Once the device is ready, application can set the host name and visibility with:

- `vm_bt_cm_set_host_name`
- `vm_bt_cm_set_visibility`

This module also manages the discovery and pairing of Bluetooth 2.1 devices. Note that Bluetooth 4.0 devices are not discovered and paired through this module – use the GATT profile module instead.

5.14.2. Profile SPP

Bluetooth Serial Port Profile (SPP) supports one-to-one connection over the Bluetooth SPP profile. The APIs are defined in `vmbt_spp.h`. This module allows the application connect to other Bluetooth 2.1 devices that supports SPP and exchange of data. The application can configure the hardware as a server or a client. Once the Bluetooth device is setup properly with Connection Manager, the application can then

- Call `vm_bt_spp_open` to access SPP module, and allocate required memory for the module.

Then, the application can either do one of the following:

- Call `vm_bt_spp_bind` to setup a server.
- Use Connection Manager API `vm_bt_cm_search` to discover nearby SPP servers and connect to a server with `vm_bt_spp_connect`.

Please refer to examples `BT_SPP_Server` and `BT_SPP_Client` for example use.

5.14.3. Profile GATT

For Bluetooth 4.0, the SDK provides Profile GATT (Generic Attribute Profile). Unlike SPP which is a defined protocol with specific usage in mind, the GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a Bluetooth 4.0 link. The application should then define or implement other Bluetooth 4.0 profiles on top of the GATT profile service.

Here are the roles and responsibilities that apply when a LinkIt Assist 2502 device interacts with a Bluetooth device:

- Central vs. peripheral. This applies to the Bluetooth connection itself. The device in the central role scans, looking for advertisement signal, and the device in the peripheral role makes the advertisement.
- GATT server vs. GATT client. This determines how two devices talk to each other once they've established the connection.

All the APIs in the GATT module is asynchronous. This means all the functions are requests passed to underlying framework for execution. The execution results are passed back from callback functions. Therefore, you initialize the GATT server and client module by registering a set of callback functions, each representing an event from the GATT server and client model.

To start a GATT peripheral – which is a server – the application will call `vm_bt_gatt_server_register` with a structure `vm_bt_gatt_server_callback_t`. In the structure the application needs to define required callbacks for different GATT server events, for example service initialization, characteristic and descriptor definition, and incoming connections. Please see Figure 63 for an example of calling sequence of APIs and callbacks.

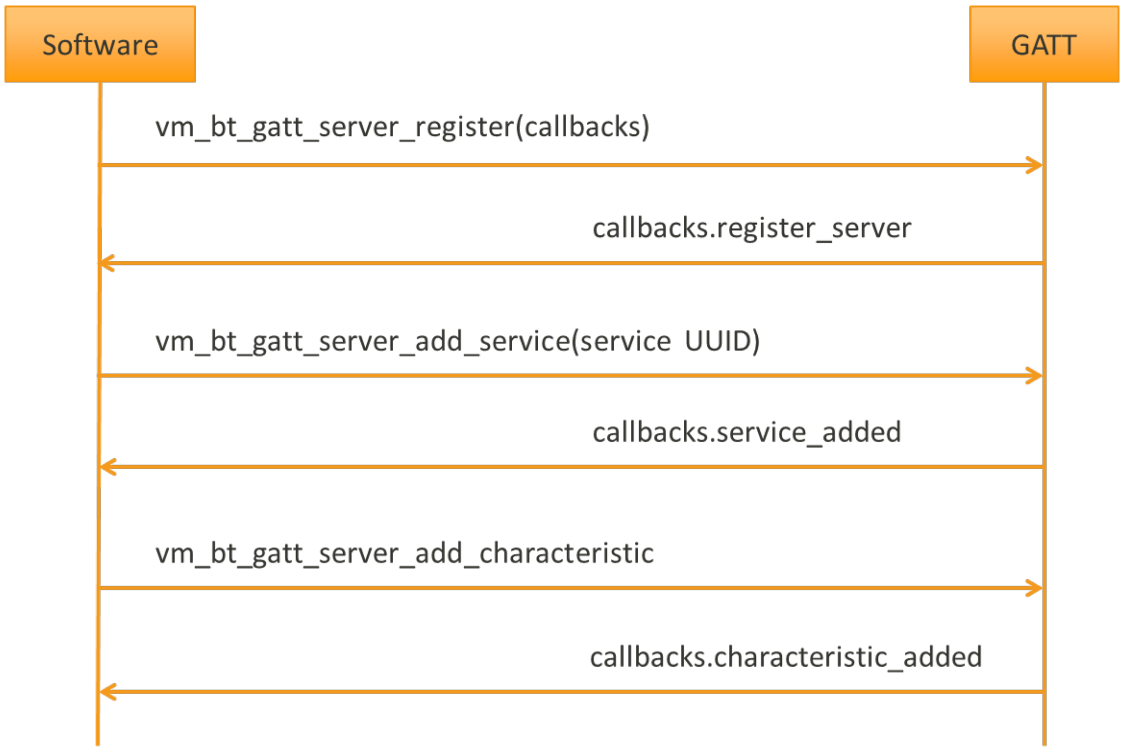

Software
GATT
vm_bt_gatt_server_register(callbacks)
callbacks.register_server
vm_bt_gatt_server_add_service(service UUID)
callbacks.service_added
vm_bt_gatt_server_add_characteristic
callbacks.characteristic_added

Figure 63 Sequence diagram for GATT callbacks

Similarly the application should start a GATT central/client by calling `vm_bt_gatt_client_register` with a callback table for GATT client events, including registration completion, scan, connection, read characteristics/descriptor for the corresponding service, and read/write characteristics.

Please refer to the following SDK examples on GATT API call flow:

- `BT_BLE_GATT_Client`: Shows the GATT client example flow.
- `BT_BLE_GATT_Server`: Shows the GATT server example flow.
- `BT_BLE_Profile_BAS`: Shows an example of creating Bluetooth 4.0 Battery Service (`https://developer.bluetooth.org/TechnologyOverview/Pages/BAS.aspx`) with GATT server API.
- `BT_BLE_Profile_DIS`: Shows an example of creating a Bluetooth Device Information Service – which is very common for most Bluetooth 4.0 profiles – with GATT server API.
- `BT_BLE_ReadRSSI`: Shows scan results of nearby Bluetooth 4.0 devices and report their RSSI strength.

5.15. Audio

The audio module provides following sub-modules:

- Audio: general audio functions, e.g. controlling the volume and registers event callback functions. Defined in `vmaudio.h`.
- Audio play: plays MP3, AAC, WAV, AMR, and MIDI format files. Defined in `vmaudio_play.h`.
- Audio record: records microphone input to WAV and AMR format files. Defined in `vmaudio_record.h`.
- Audio stream play: allows applications to play in-memory datastreams in PCM, ARM, AAC, MP3 format. Defined in `vmaudio_stream_play.h`.

The audio play module is fairly straightforward to use, refer to `vmaudio_play.h` for interface definitions. Only one file can be played at time, and the audio resource is shared between audio module and telephony module, therefore audio playback will be interrupted by incoming telephony calls.

The audio record module records microphone audio inputs into a single file. WAV and AMR file formats are supported. Note that there is no interface to access raw audio data during recording.

The audio stream play module supports only playback. Applications provide streaming data in memory buffer to the module in PCM/AMC/AAC/MP3 format. The module then consumes the data and sends `VM_AUDIO_STREAM_PLAY_EVENT_DATA_REQUEST` to the callback function registered by `vm_audio_register_interrupt_callback`. The application should then call `vm_audio_stream_play_put_data` to fill subsequent streaming data. If the application fails to feed data, audio playback fails.

5.16. GNSS

Where the MediaTek MT3332 chipset is included in hardware, such as LinkIt Assist 2502 development board, you can use the GPS module to access GPS NMEA data. If the underlying hardware build supports multiple satellite systems and the use of multiple satellite systems is defined, the underlying driver will merge the results from different satellite systems into the same output in NMEA format with following rules:

- GGA information across different satellite systems will be merged into GPGGA strings.
- GSA and GSV information are kept separately, for example:
  - GPGSV for GPS GSV data.
  - GLGSV for GLONASS GSV data.
  - BDGSV for BeiDou GSV data.

To use the GPS-related functions, call `vm_gps_open()` to enable the underlying hardware and assign satellite systems you want to use and a callback function that will receive events such as NMEA string and parameter updates. See `VM_GPS_TYPE` for a list of supported satellite systems, and `VM_GPS_MESSAGE` for a list of types of messages passed by callback function.

After optionally setting parameters with `vm_gps_set_parameters()`, once the GPS system has located the current position information, the callback function registered in `vm_gps_open()` will start to receive `VM_GPS_MESSAGE` messages of type `VM_GPS_SENTENCE_DATA`, you should then parse the NMEA sentences to retrieve information needed. Refer to the example *GPS* for an example of parsing the NMEA string.

To un-initialize and turn off GPS modules, call `vm_gps_close()`.

5.16.1. Extended Prediction Orbit (EPO)

The LinkIt Assist 2502 development platform also supports MediaTek Extended Prediction Orbit (EPO) for the MediaTek MT3332 chipset. This feature enhances the GPS user experience by enabling a faster GPS positioning fix, reducing the time to first fix (TTFF). MediaTek EPO does this by downloading satellite calendar data based on the device's location using Wi-Fi or GPRS. The file downloaded is small (about 10 KB) and you have the option to disable this feature by editing the `VM_GPS_SET_EPO_AUTO_DOWNLOAD` parameter to `0` in the `vm_gps_set_parameters()` function of the GPS module. For example:

```
vm_gps_set_parameters(VM_GPS_SET_EPO_AUTO_DOWNLOAD, 0, user_data)
```

Please note the default `VM_GPS_SET_EPO_AUTO_DOWNLOAD` parameter is 1, which is ON.

5.17. Graphics

The graphics module provides following functions:

- Perform pixel operations.
- Draw points, lines, rectangles and ellipses.
- Decode image files.
- Support MediaTek Serial Interface LCD device.
- Draw text with vector fonts.

All the APIs in the graphics module must be called after `VM_EVENT_PAINT` arrive as the system event handler, and can only be called in the main thread.

5.17.1. Images and Display

LinkIt Assist 2502 SDK 2.0 supports these four image formats: JPEG, GIF, PNG and BMP. The input can be from file or from memory. Note that the image can be decoded from a file or a resource embedded to the application. To embed images to application, refer to 4.8.1, "Adding File and Image Resources".

LinkIt Assist 2502 draws the shapes, images or text described above to a *frame*. A frame contains following elements:

- Width: the pixel width of the frame.
- Height: the pixel height of the frame.
- Color format: the color format of the frame, it indicates how many bits that one pixel needs and whether there is an alpha channel of the pixel.
- Buffer: the memory that contains the graphic data. This must be allocated with `vm_malloc_dma()`.
- Buffer length: the size of the buffer, it dependents on the width, height and `color_format` of the frame.

The application should setup the frame properly before using the graphics module. If the size of the buffer is larger than the display device, it will be cropped. To display frame on the display device, make sure that its size and color format matches the display device, and call `vm_graphic_blt_frame()` which takes an array of frame pointers up to 4 frames, composites them and then transfer the result to the display device. Note that only display devices that supports MediaTek Serial LCD interface are supported. For display devices with SPI interface, use the DCL APIs to transfer the buffer according to the protocol defined by the device itself.

5.17.2. Vector Font

To draw vector font onto a frame, call `vm_graphic_draw_text`. There are also variations of the text drawing API that allows application to draw text according to different constraints, e.g. baseline and bounding width. Please refer to API reference for details.

The texts are drawn with the system default font, which supports Latin characters only. The SDK comes with a set of font files in the `custom\VectorFont` folder in the SDK, supporting different languages.

To change the font used to draw the texts, the application should allocate the memory resource for the font files to be loaded with `vm_graphic_get_font_pool_size` and then `vm_graphic_init_font_pool. vm_graphic_get_font_pool_size` takes an estimation value of memory consumption for each font, which can be using the tools WHAT TOOL? Provided in the `custom\VectorFont` directory.

Then the application should call `vm_graphic_set_font`, which takes an array of font paths as input and loads them one by one. The font in the front of the array has the highest priority, which means system will try to find the font data for a given character from this font file first, if it doesn't find it, it will try the second priority font file. This allows application to fallback to different font for different languages.

The font loading operation takes some time to complete, and is only required when changing fonts. If the application want to clean all the installed font and revert to the built-in font, it should call `vm_graphic_reset_font()`.

The use of fonts require additional memory for your running software, in the form of a 100KB vector font cache. To ensure the best performance when using vector fonts, subtract the font cache size from maximum allowed memory size for your application (see 4.5, "Memory Layout of a LinkIt Assist 2502 Application").

5.17.3. Calculating Text Width

It is recommended that `vm_graphic_get_text_width` is used to obtain the width of the string in one step and `vm_graphic_draw_text` be used to draw the entire string. The value returned from `vm_graphic_measure_character` or `vm_graphic_get_character_width` contains the width of the individual character without padding. Therefore if a vector font is used, calculating the length of a string by the aggregate lengths of the individual characters will produce unpredictable and inconsistent results under different handset models and versions.

You can use the `vm_graphic_measure_character` interface to access the height and width of the character simultaneously; the input character parameters can be in ASCII or Chinese character format. Furthermore, use the `vm_graphic_get_character_width` interface to access the width of ASCII and Chinese characters and `vm_graphic_get_character_height` to obtain the height of ASCII characters.

5.18. VXP and Firmware Update

LinkIt Assist 2502 provides vxp update and firmware update methods, vxp update replaces the older vxp with the latest one and firmware update will install a new firmware on your device. Only full package update is supported.

For VXP update, please see `vm_pmng_exit_and_update_application` in `vmsystem.h.`

Filename is vxp's full path, currently only DES encryption is supported, please don't apply padding mode for DES and make sure the encrypted-file's size is the same as the original file's size, otherwise, an exception error will occur or the vxp cannot run. If the vxp file is not encrypted, please set three parameters: `key to NULL,key_length to 0, encryption to VM_PMNG_ENCRYPTION_NONE`.

For Firmware update, please see `vm_firmware_trigger_update()` in `vmfirmware.h`, the binary must be written to the following path in file system `C:\\image.bin`, and this path cannot be modified. If power outage occurs during update period, then update will be triggered next time automatically. In most cases, if firmware update fails, it is possible that the `image.bin` is inaccurate. The error code is in the file `C:\update_status`, please refer to firmware example on how to read the error code. There are many possible errors and five most common are summarized below for your reference.

1) The image.bin file is too large and cannot be updated. You should regenerate an image with a size that is suitable to use. The error code maybe -1107,-1106,-1108,-321.
2) The image.bin's signature is inaccurate, please place the right image.bin file. The error code may be -303,-304.
3) It cannot find the image.bin, please check the image.bin and make sure it is in `C:\ root dir`. The error code maybe -701,-702.
4) The image.bin 's MBA does not match the target, please check the MBA of the image.bin. The error code maybe-412 to 416, 324 or 325.
5) The image.bin's UBIN does not match the target, please check the UBIN of the image.bin, The error code may be -417 to -420.

5.19. Smartphone Notification

Smartphone notification module can receive notification from Android and iOS devices in a high level fashion - without the need to implement the underlying protocols.

For iOS, ANCS protocol is supported.

For Android, an accompanying Smartphone Notification library is included in the custom folder in the SDK. There is also an Android app called "SmartDevice" for a quick test. The library connects to LinkIt Assist 2502 devices using Bluetooth SPP or GATT, depending on different use cases.

Before using Smartphone Notification APIs, the application should enable Bluetooth with Connection Manager and enable visibility to allow smartphones to scan and connect to the LinkIt Assist 2502 device. Then the LinkIt Assist 2502 device can receive notifications, such as:

- Message: if a smart device receives new SMS, it will push SMS information into LinkIt Assist 2502 device, the information includes SMS sender number and SMS content.
- Missed call: if a smart device misses a call, it will push missed call count information to LinkIt Assist 2502 device.
- Low battery warning: if there's a low battery status, the smart device will push a warning information into the LinkIt Assist 2502 device.
- Other notifications: they include content and sender (application name) information.

Note: this module relies on the iOS or Android devices to send the notification but some notification types may not be available on certain platforms.

For more information please see MediaTek LinkIt Smartphone Notification Developer's Guide.