MicroEJ Platform Reference Implementation

Developer's Guide

OM13098 2.0.1

Reference:	TLT-0825-DGI-PlatformReferenceImplementation-OM13098
Version:	2.0.1
Revision:	2.0.0

Revision History		
Revision 2.0.0	September 10th 2019	
Changelog updated for release 2.0.0		
Revision 1.2.1	July 24th 2019	
Changelog updated for release 1.2.1		
Revision 1.2.0	June 11th 2019	
Changelog updated for release 1.2.0		
Revision 1.1.1	July 20th 2018	
Update of the changelog.		
Revision 1.1.0	December 28th 2017	
Changelog updated for release 1.1.0		
Revision 1.0.0	December 21st 2017	
First release		

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Intended Audience

The intended audience for this document are developers who wish to develop their first MicroEJ plaform with MicroEJ SDK and deploy a MicroEJ standalone application onto. Notes:

- This document is for the NXP OM13098 board.
- This document is not a user guide for the C development environment used for the final application link. Please consult the supplier of the C development environment for more information.
- Please visit the website https://developer.microej.com for more information about OM13098 products (platforms, videos, examples, application notes, etc.).

1.2. Scope

This document describes, step by step, how to start your development with MicroEJ SDK

- Create a MicroEJ platform for OM13098 board.
- Run a MicroEJ standalone application on the MicroEJ simulator.
- Run a MicroEJ standalone application on the MicroEJ platform and deploy it on the OM13098 board.

1.3. Prerequisites

- PC with Windows 7 or later.
- The MicroEJ SDK environment must be installed.
- OM13098 board.
- The Segger J-Link software.
- IAR Embedded Workbench for ARM 8.32.1. To get IAR Embedded Workbench for ARM, please visit the IAR website.

Note

Higher versions of IAR Embedded Workbench for ARM can be used when building your own platform.

Chapter 2. Create and Use Your First MicroEJ Platform

2.1. Create a MicroEJ Platform

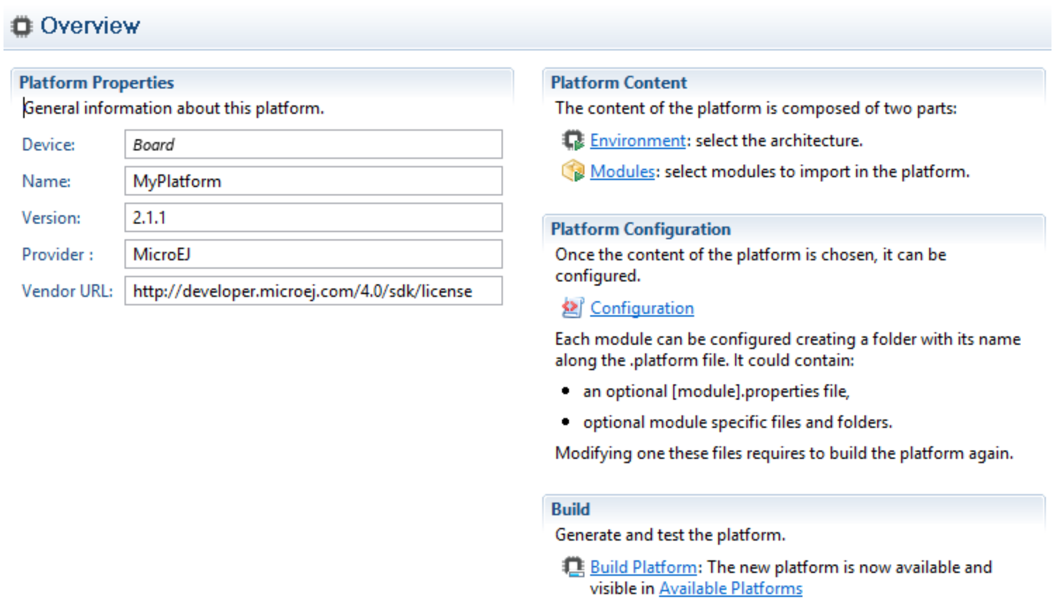
The aim of this chapter is to create a MicroEJ platform from a MicroEJ architecture. The platform will then be used to run a MicroEJ standalone application in subsequent chapters.

Although it is possible to use MicroEJ SDK to create every aspect of a MicroEJ platform in accordance with specific requirements, in this chapter we will use a pre-packaged example of a MicroEJ platform that is already configured for the OM13098.

- Inside MicroEJ SDK, the selected example is imported as several projects prefixed by the given name:
  - {PLATFORM}-configuration: Contains the platform reference implementation configuration description. Some modules are described in a specific sub-folder / with some optional configuration files (`.properties` and / or `.xml`).
  - {PLATFORM}-bsp: Contains a ready-to-use BSP software project for the OM13098 board, including a IAR Embedded Workbench project, an implementation of MicroEJ core engine (and extensions) port on FreeRTOS RTOS and the OM13098 board support package.
  - {PLATFORM}-fp: Contains the board description and images for the MicroEJ simulator. This project is updated once the platform is built.

  Inside the {PLATFORM}-configuration project, open the `.platform` MicroEJ platform configuration file.
- From this MicroEJ platform configuration file, click on the link `Build Platform`

Figure 2.1. MicroEJ Platform Build



Overview

Platform Properties
General information about this platform.

Device:	Board
Name:	MyPlatform
Version:	2.1.1
Provider :	MicroEJ
Vendor URL:	http://developer.microej.com/4.0/sdk/license

Platform Content
The content of the platform is composed of two parts:
Environment: select the architecture.
Modules: select modules to import in the platform.

Platform Configuration
Once the content of the platform is chosen, it can be configured.
Configuration
Each module can be configured creating a folder with its name along the .platform file. It could contain:
- an optional [module].properties file,
- optional module specific files and folders.

Modifying one these files requires to build the platform again.

Build
Generate and test the platform.
Build Platform: The new platform is now available and visible in Available Platforms

The build starts. This step may take several minutes. You can see the progress of the build steps in the MicroEJ console. Please wait for the final message:

```
BUILD SUCCESSFUL
```

At the end of the execution the MicroEJ platform is fully built for the OM13098 board and is ready to be linked into the IAR Embedded Workbench project. Its name is `OM13098-MyPlatform-CM4hardfp_IAR83`.

The MicroEJ platform is now ready for use and available in the MicroEJ platforms list of your MicroEJ repository (`Windows > Preferences > MicroEJ > Platforms in workspace`).

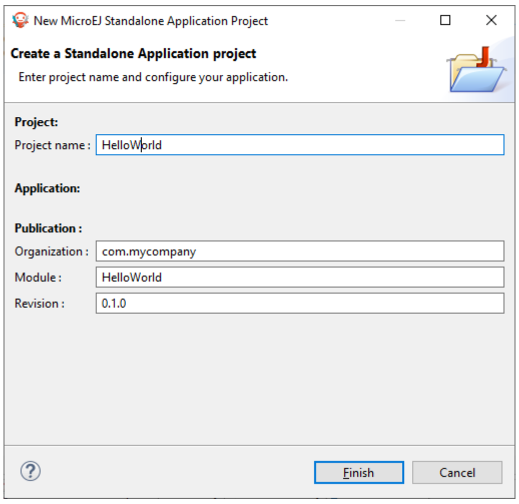
2.2. Run an Example on the MicroEJ Simulator

The aim of this chapter is to create a MicroEJ standalone application from a built-in example. Initially, this example will run on the MicroEJ simulator. Then, in the next section, this application will be compiled and deployed on the OM13098 board using the MicroEJ platform.

2.2.1. Create Example

- Open MicroEJ SDK.
- Open the `File > New > MicroEJ Standalone Application Project` menu.
- Enter your project name (e.g. HelloWorld).

Figure 2.2. MicroEJ Standalone Application Creation



New MicroEJ Standalone Application Project

Create a Standalone Application project

Enter project name and configure your application.

Project:

Project name : HelloWorld

Application:

Publication :

Organization : com.mycompany

Module : HelloWorld

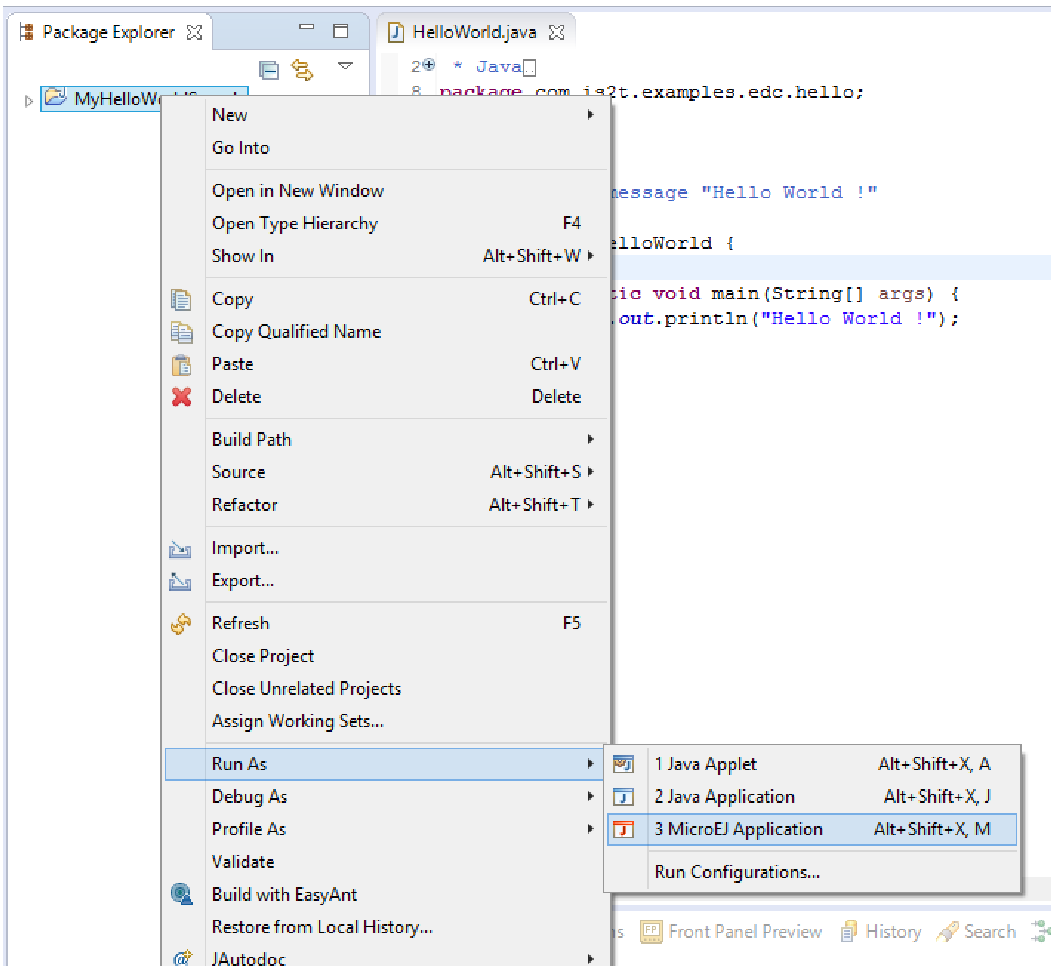
Revision : 0.1.0

- Click on Finish. The example is created into a project with the given name. The main class (the class which contains the `main()` method) is automatically generated.

2.2.2. Run Example

- Select the project in the Package Explorer tree
- Right-click on this project and select `Run As > MicroEJ Application`

Figure 2.3. MicroEJ Standalone Application Running



```
 * Java
package com.is2t.examples.edc.hello;
essage "Hello World !"
lloWorld {
ic void main(String[] args) {
.out.println("Hello World !");
```

The application starts. It is executed on the MicroEJ simulator of the selected MicroEJ platform (OM13098-MyPlatform-CM4hardfp_IAR83). The result of the test is printed in the console:

```
Hello World!
```

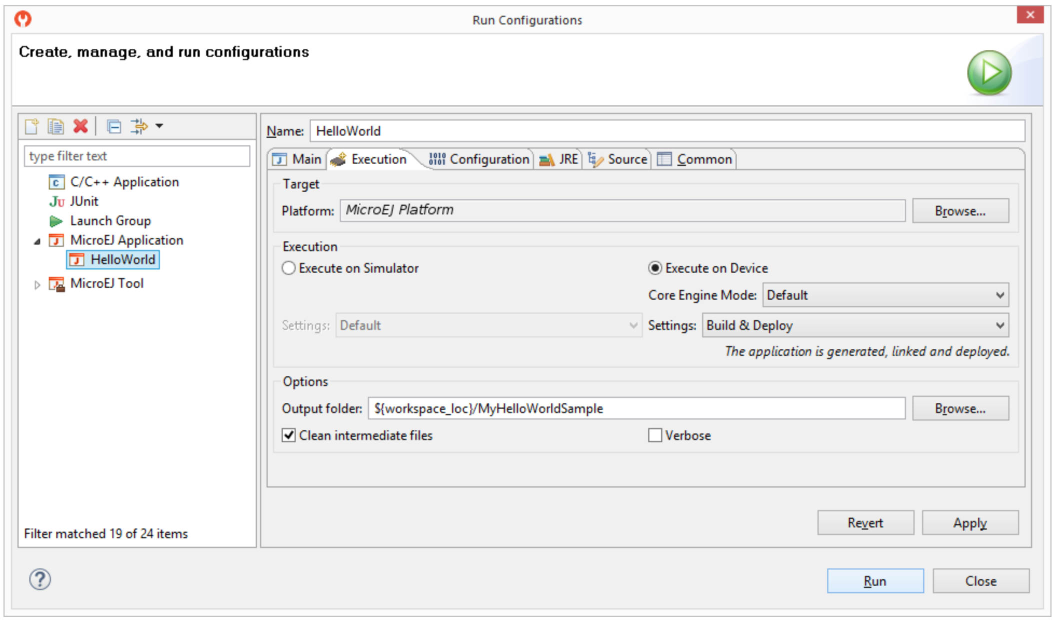
2.3. Run the Example on the OM13098 Board

2.3.1. Compile MicroEJ Standalone Application

- Open the run dialog (`Run > Run configurations...`).

- Select the MicroEJ Application launcher `HelloWorld Main` that is created by the previous chapter.
- Open `Execution` tab.
- Select `Execute on Device`.

Figure 2.4. Execution on Device



Name: HelloWorld

Target
Platform: MicroEJ Platform

Execution
○ Execute on Simulator ● Execute on Device
Core Engine Mode: Default
Settings: Default Settings: Build & Deploy
The application is generated, linked and deployed.

Options
Output folder: ${workspace_loc}/MyHelloWorldSample
☑ Clean intermediate files ☐ Verbose

- Open `Configuration` tab and sub menu `Target > Deploy`. By default, an option is set to deploy the application library at a location known by the third-party IDE. If you want to deploy it elsewhere, unselect this option and enter your output path in the field below.
- Click Run: the application is compiled, and the compilation result (an ELF file) is copied into a well known location in the workspace. The IAR Embedded Workbench BSP project will search for it there when it performs the final link.

2.3.2. Link and deploy the MicroEJ Standalone Application

The aim of the final step is to:

- Compile the BSP project (such as drivers).
- Link the BSP and the others libraries (MicroEJ Core Engine, C stacks, MicroEJ standalone application etc.).
- Deploy a MicroEJ standalone application on the OM13098 board.

Note

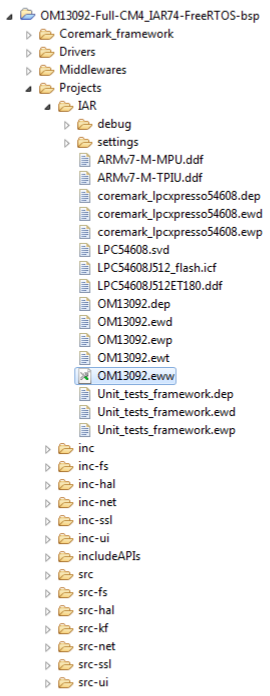
This final step uses IAR Embedded Workbench.

2.3.2.1. Select the BSP project

In MicroEJ SDK:

- In MicroEJ SDK, expand the project OM13098-MyPlatform-CM4hardfp_IAR83-bsp and the folder `Projects/IAR`. A IAR Embedded Workbench project file (`lpcxpresso.eww`) is available.

Figure 2.5. IAR Embedded Workbench Project Selection



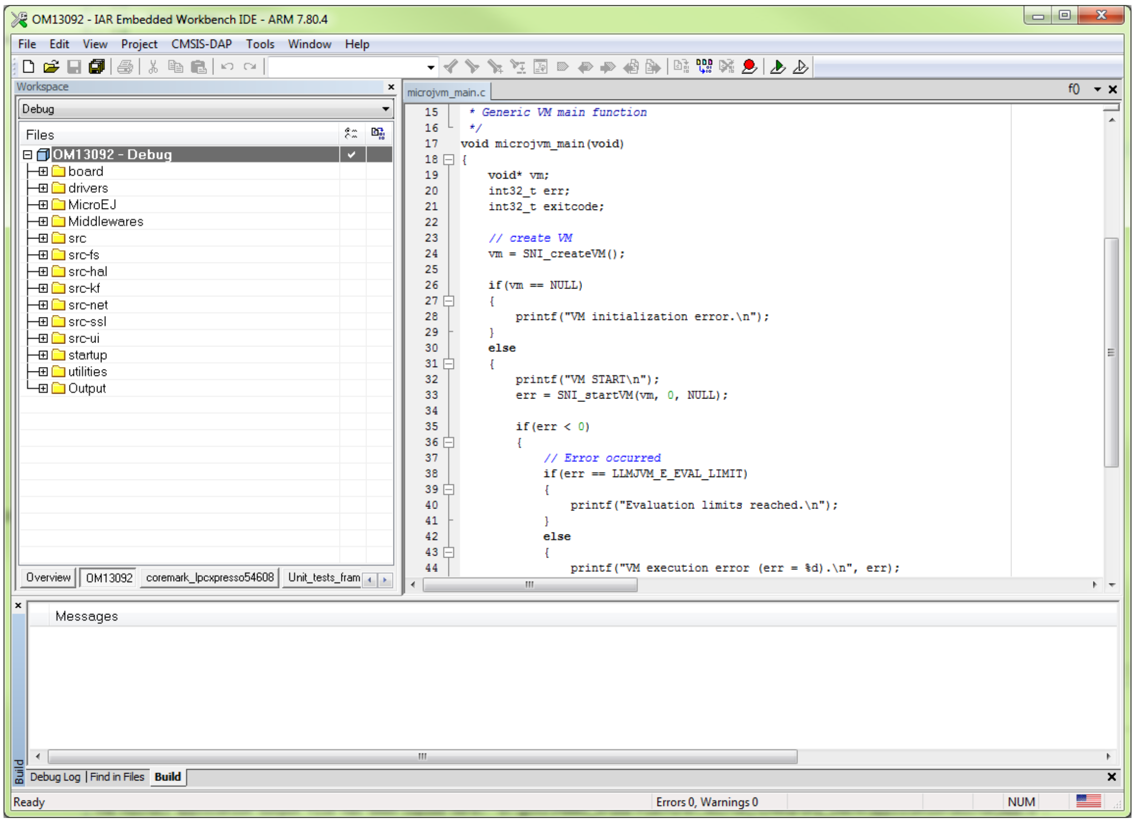
```
OM13092-Full-CM4_IAR74-FreeRTOS-bsp
  Coremark_framework
  Drivers
  Middlewares
  Projects
    IAR
      debug
      settings
      ARMv7-M-MPU.ddf
      ARMv7-M-TPIU.ddf
      coremark_lpcxpresso54608.dep
      coremark_lpcxpresso54608.ewd
      coremark_lpcxpresso54608.ewp
      LPC54608.svd
      LPC54608J512_flash.icf
      LPC54608J512ET180.ddf
      OM13092.dep
      OM13092.ewd
      OM13092.ewp
      OM13092.ewt
      OM13092.eww
      Unit_tests_framework.dep
      Unit_tests_framework.ewd
      Unit_tests_framework.ewp
    inc
    inc-fs
    inc-hal
    inc-net
    inc-ssl
    inc-ui
    includeAPIs
    src
    src-fs
    src-hal
    src-kf
    src-net
    src-ssl
    src-ui
```

Double-click on this file to open IAR Embedded Workbench.

2.3.2.2. Build and link the platform

In IAR Embedded Workbench:

Figure 2.6. IAR Embedded Workbench IDE



```
 * Generic VM main function
 */
void microjvm_main(void)
{
    void* vm;
    int32_t err;
    int32_t exitcode;

    // create VM
    vm = SNI_createVM();

    if(vm == NULL)
    {
        printf("VM initialization error.\n");
    }
    else
    {
        printf("VM START\n");
        err = SNI_startVM(vm, 0, NULL);

        if(err < 0)
        {
            // Error occurred
            if(err == LLMJVM_E_EVAL_LIMIT)
            {
                printf("Evaluation limits reached.\n");
            }
            else
            {
                printf("VM execution error (err = %d).\n", err);
```

Build the IAR Embedded Workbench project by clicking on the menu `Project > Make`. The project is compiled and linked. See “Mandatory Connectors” to use the right connectors.

2.3.2.3. Deploy the application

Note

The target board must be flashed with a J-Link bootloader/downloader before deploying.

- LPCScrypt binaries and flashing tools are available at: `https://www.nxp.com/support/developer-resources/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT`
    - Follow directive from LPCScrypt User guide to install drivers.
    - Set jumper JP5 on the board (it is important to set the jumper while the board power is off).
    - Power the board by connecting the board to the PC using a micro USB to USB cable on connector J8 (USB Debug Link).
    - Launch the Program LPC-Link2 with Segger J-Link script. With a default installation, it's located here: C:\NXP\LPCScrypt\scripts\program_JLINK.cmd.

- On the command invite, press the spacebar to flash the J-Link firmware. Once it is done, unplug power from the board and quit the script.
- Remove the jumper from the JP5 connector.
- Power the board. We can now use J-Link software to flash it.

Deploy the link result on the OM13098 board by clicking on the menu `Project > Download and Debug`.

Finally run the application by clicking on the menu `Debug > Go`.

2.3.2.4. Congratulation: you run the application

The application starts. The result of the execution is output on printf COM port. (See "Mandatory Connectors" to use the right connectors). Congratulations, you have deployed a MicroEJ standalone application on a MicroEJ platform.

Chapter 3. Specification

3.1. Overview

MicroEJ platform on OM13098 is based on board support package provided by NXP: SDK_2.2_LPCXpresso54628. It includes FreeRTOS, a graphical user interface, a TCP/IP network connection, a file system on SD-Card and some custom GPIOs. MicroEJ platform has been built against IAR Embedded Workbench IDE.

3.2. MicroEJ Platform Configuration

MicroEJ platform is based on MicroEJ architecture for ARM Cortex-M4.

Table 3.1. MCU Technical Specifications

MCU architecture	Cortex-M4 (LPC54628)
MCU Clock speed	220 MHz
Internal Flash	512 KB
Internal RAM	256 KB
Internal EEPROM	16 KB
External Flash	16 MB (QSPI)
External RAM	8 MB

MicroEJ platform uses several architecture extensions. The following table illustrates the MicroEJ architecture and extensions versions.

Table 3.2. MicroEJ Configuration

Name	Version
MicroEJ architecture	7.11.0
UI	12.0.1
Network	6.1.5
File System	4.0.2
HAL	1.0.6

3.3. Platform Output stream

MicroEJ platform uses a USB Virtual COM port as output print stream. The virtual COM port is available on the USB Debug-Link/J8 connector.

Implementation Note

The COM port is also used as the output stream for the *printf* calls.

The COM port uses the following parameters:

- Baudrate: 115200
- Data bits bits: 8
- Parity bits: None
- Stop bits: 1
- Flow control: None

Implementation Note

On the OM13098, the following parameters can be adjusted:

- Baudrate
- Parity bits
- Stop bits

3.4. RTOS Configuration

MicroEJ platform uses FreeRTOS 8.2.1. RTOS uses a heap to allocate all its objects: tasks stacks, task monitors, semaphores etc. The heap size is: 45 KB and is allocated in internal RAM. The following table illustrates the available tasks and their stack size.

Table 3.3. FreeRTOS Tasks

Task name	Size	Priority
RTOS idle task	360 B	0
RTOS timer	720 B	16
Core Engine	12 KB	11
CPU load	512 B	15
Touch	512 B	13
Network dispatch	2 KB	12
Network DHCP	720 B	11
Network Ethernet link	720 B	12
Network Ethernet input	1400 B	14
LWIP TCP	4 KB	13
File system	2 KB	12

3.5. Memories

MicroEJ Plaform uses several internal and external memories. The following table illustrates the MCU and board memory layouts and sizes fixed by the MicroEJ platform.

Table 3.4. Internal RAM (256 KB)

Section Name	Size
MicroEJ standalone application stack blocks	512 * *n* bytes [a]
Pre-installed MicroEJ sandboxed application	*n* bytes [b]
MicroEJ platform internal heap	*n* bytes [c]
Any RW	*n* bytes [d]
MicroEJ standalone application heaps	1536 KB [e]

[a] *n* is the number of stack blocks defined in MicroEJ Application launcher options.

[b] *n* depends on the size defined in MicroEJ Application launcher options.

[c] *n* depends on memory configuration set in MicroEJ Application launcher options.

[d] *n* depends on MicroEJ application libraries used.

[e] Maximum size of the addition of MicroEJ heap size and MicroEJ immortal heap size. These sizes are defined in MicroEJ Application launcher options.

Table 3.5. External RAM: SDRAM (8 MB)

Section Name	Size
Display buffers	510 KB
MicroUI working buffer	2586 KB
Multi applications working buffer	3 MB
SSL buffers	130 KB
Ethernet buffers	77 KB

Table 3.6. Internal flash: Program Flash (512 KB)

Section Name	Size
Any RO	*n* bytes [a]

[a] *n* depends on MicroEJ application, MicroEJ libraries, Board support package, RTOS, drivers, etc.

Table 3.7. External flash: QSPI (16 MB)

Section Name	Size
MicroEJ standalone application resources	*n* bytes [a]

[a] *n* is the size of all MicroEJ standalone application resources.

3.6. Multi Applications

This MicroEJ platform includes the Multi applications mode. Multi applications mode allows to build a firmware that can manage MicroEJ sandboxed applications. Multi applications mode requires a specific memory area to load MicroEJ sandboxed applications. This memory area is located in external SDRAM and its default size is 3 MB.

3.7. Graphical User Interface

MicroEJ plaform features a graphical user interface. It includes a display, a touch panel, an user button and a runtime PNG decoder.

3.7.1. Display

The display module drives a 480 x 272 LCD display. The pixel format is 16 bits-per-pixel: 5 bits for red color component, 6 bits for green color component and 5 bits for blue color component. The display device is clocked at 60Hz and the MicroEJ application drawings are synchronized on this display tick.

Implementation Note

The display stack implementation uses the switch double-buffering mode: the current MicroEJ application rendering is performed in a background buffer (called back buffer) and another buffer is used by the TFT display to refresh itself (called frame buffer). When the drawing is done, the BSP performs the `switch`, meaning that the back buffer will become the frame buffer and vice-versa.

Both the back buffer and the frame buffer are located in external SDRAM. The size depends on the display size in pixels and on the number of bits-per-pixel (BPP):

`bufferSize = width * height * bpp / 8;`, where:

- `width` is the display width in pixels: 480
- `height` is the display width in pixels: 272
- `bpp` is the number of bits-per-pixel: 16

The buffers size is 2 `* 262120 = 510 KB`.

MicroUI requires a RAM buffer to store the dynamic images data. A dynamic is an image decoded at runtime (PNG image) or an image created by the MicroEJ application thanks the API `Image.create(width, height)`. This buffer is located in external SDRAM and the reserved size is 2586 Kbytes.

Implementation Note

This buffer is called "working buffer". An image buffer size follows the same rule than the LCD buffer (see before).

3.7.2. Inputs

Touch panel: All touch panel events are sent to the MicroEJ application thanks a `Pointer` event generator.

Implementation Note

The touch events (`press`, `drag`, `release`) are detected by interrupt.

User buttons: The user buttons are reserved to the multi applications feature: they allow to force the kill of a sandboxed application.

Implementation Note

The user buttons events treatments are performed under interrupt.

3.8. Network

MicroEJ plaform features a network interface. A limited number of 10 sockets could be used for TCP connections, 5 for TCP listening (server) connections and 6 for UDP connections. A DHCP client could be activated to retrieve IP address. All DNS requests could be handled by a MicroEJ software implementation or a native one.

Implementation Note

MicroEJ platform uses LwIP v1.4.0 fetched from git repository of the project. This implementation need a 40 KB internal heap to work. The TCP MSS is 1460 bytes.

The network portage use a BSD (Berkley Software Distribution) API with select feature. A mechanism named dispatch event, with a dedicated task, is used to request non blocking operations and wait for completion or timeout.

The DHCP client is handled by LwIP and the DNS features use a MicroEJ software implementation.

3.9. SSL

MicroEJ platform features a network secure interface. Available secured protocols are SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, DTLS 1.0, DTLS 1.2. Keys and certificates supported formats are PEM, DER and PKCS#12.

Implementation Note

MicroEJ platform uses WolfSSL v3.3.2. WolfSSL uses a heap of 130 KB to store certificates.

3.10. File System

MicroEJ plaform features a file system interface. A SD card is used for the storage (previously formated to a FAT32 file system). Up to 2 files could be opened simultaneously.

Implementation Note

MicroEJ platform uses FatFS R0.12b. The FAT FS driver is the SD driver port of SDMMC v2.1.2.

3.11. Serial Communications

3.11.1. UART Connector

MicroEJ platform provide a USB Virtual COM port as output print stream. The virtual COM port is available on the USB Debug-Link/J8 connector. On board revision C and later:

- If JP6 jumper is open, UART use Debug Link (J8) virtual COM port.
- If JP6 jumper is closed, UART use P4 connector.

3.12. HAL

MicroEJ platform includes four expansion connectors (J9, J10, J12 and J13) incorporate an Arduino Uno revision 3 footprint in their inner rows. These connectors provide access to the CAN interfaces, additional digital microphone support signals, I2S, I2C, USART, SPI and GPIO/INT/PWM connections. Note that several of the signals available at these connectors are shared with other connectors or board functions, so may not be usable if those other functions are being used.

MicroEJ platform provides several GPIOs to connect HAL foundation library. All GPIOs that are available on the expansion connectors (J9, J10, J12 and J13), and can be used for external access, are usable through the HAL foundation library. Digital pins are implemented by a GPIO access, and none of the available GPIOs are connected to any ADC channels, so they cannot operate in analog mode.

Each GPIO port / pin value is accessible by several ways:

1. Using the global MCU declaration: all pins of all ports are grouped under only one virtual port (port 0) and have consecutive values: PT0_0 has the ID 0, PT0_1, the ID 1, PT1_0 the ID 32 and so on. For instance pin *PT3_11* is accessible by `(0, 107)`. This declaration is useful to target all MCU pins using only one virtual port.
2. Using the standard MCU declaration: PORT0 has the ID 1, PORT1 the ID 2 etc. Each pin of each port is a value between 0 (Port*N*-0) to 31 (Port*N*-31). For instance pin *PT3_11* is accessible by `(4, 11)`. This declaration is useful to target a specific MCU pin.
3. Using the virtual board connectors: `ARDUINO_DIGITAL` and `ARDUINO_ANALOG`. These connectors make an abstraction between the MicroEJ application and the HAL implementation. For instance pin `PT3_11` is accessible on connector `ARDUINO_DIGITAL`, pin 47 (there are 20 pins on connector J9, 20 pins on connector J10, and `PT3_11` is connected to J12, pin7).
4. 4. Using the physical board connectors. The board has 4 connectors: J9, J10, J12 and J13, with respectively these IDs: 64, 65, 66 and 67. For instance pin `PT3_11` is accessible on connector

J12, pin7: `(66, 7)`. this declaration is useful to target a physical connector pin without knowing which MCU pin it is.

The following table summaries the exhaustive list of GPIOs ports accessible from HAL library, and the ranges of pins IDs:

Table 3.8. HAL GPIOs Ports and Pins

Port name	HAL port ID	Pins range
Global MCU virtual port	0	0 to 170
MCU port 0	1	0 to 31
MCU port 1	2	0 to 31
MCU port 2	3	0 to 31
MCU port 3	4	0 to 31
MCU port 4	5	0 to 31
MCU port 5	6	0 to 10
Board virtual port `DIGITAL`	30	1 to 72
Board virtual port `ANALOG`	31	1 to 72
Board physical port `J9`	64	1 to 20
Board physical port `J10`	65	1 to 20
Board physical port `J12`	66	1 to 12
Board physical port `J13`	67	1 to 20

The following table illustrates the exhaustive list of GPIOs connected to the HAL library, their IDs according the ports IDs and pins IDs (see before). This table indicates too the useful ADC / DAC channels for HAL analog pins:

Table 3.9. HAL GPIOs Declaration (port, pin)

Port / Pin	MCU virtual port (1)	MCU port (2)	Digital virtual connector (3)	Analog virtual connector (3)	Board physical port (4)
P1_17	0, 50	2, 17	30, 4	--	64, 4
P1_18	0, 51	2, 18	30, 2	--	64, 2
P2_17	0, 82	3, 17	30, 72	--	67, 20
P3_11	0, 108	4, 11	30, 47	--	66, 7
P3_12	0, 109	4, 12	30, 49	--	66, 9
P3_16	0, 113	4, 16	30, 43	--	66, 3
P3_17	0, 114	4, 17	30, 70	--	67, 18
P3_28	0, 125	4, 28	30, 68	--	67, 16
P3_29	0, 126	4, 29	30, 66	--	67, 14
P4_2	0, 130	5, 2	30, 10	--	64, 10

Port / Pin	MCU virtual port (1)	MCU port (2)	Digital virtual connector (3)	Analog virtual connector (3)	Board physical port (4)
P4_4	0, 132	5, 4	30, 71	--	67, 19
P4_6	0, 134	5, 6	30, 41	--	66, 1

None of the pins available on the Arduino ports are connected to any ADC channel, therefore they cannot be used in analog configuration.

Chapter 4. Board Configuration

OM13098 provides several connectors, each connector is used by the MicroEJ Core Engine itself or by a foundation library.

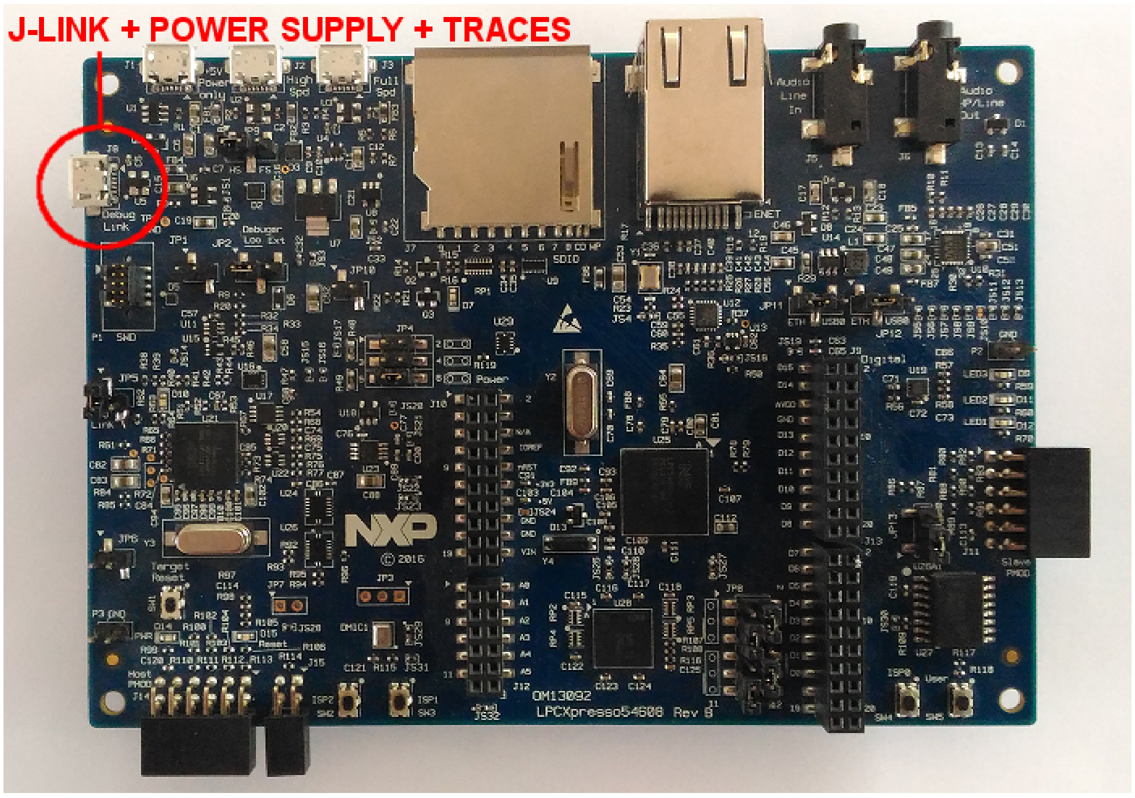
4.1. Mandatory Connectors

OM13098 provides a multi function USB port used as:

- Power supply connector
- Probe connector

Plug a USB type B cable to a computer to power on the board, be able to program an application on it and to see the MicroEJ standalone application `System.out.print` traces.

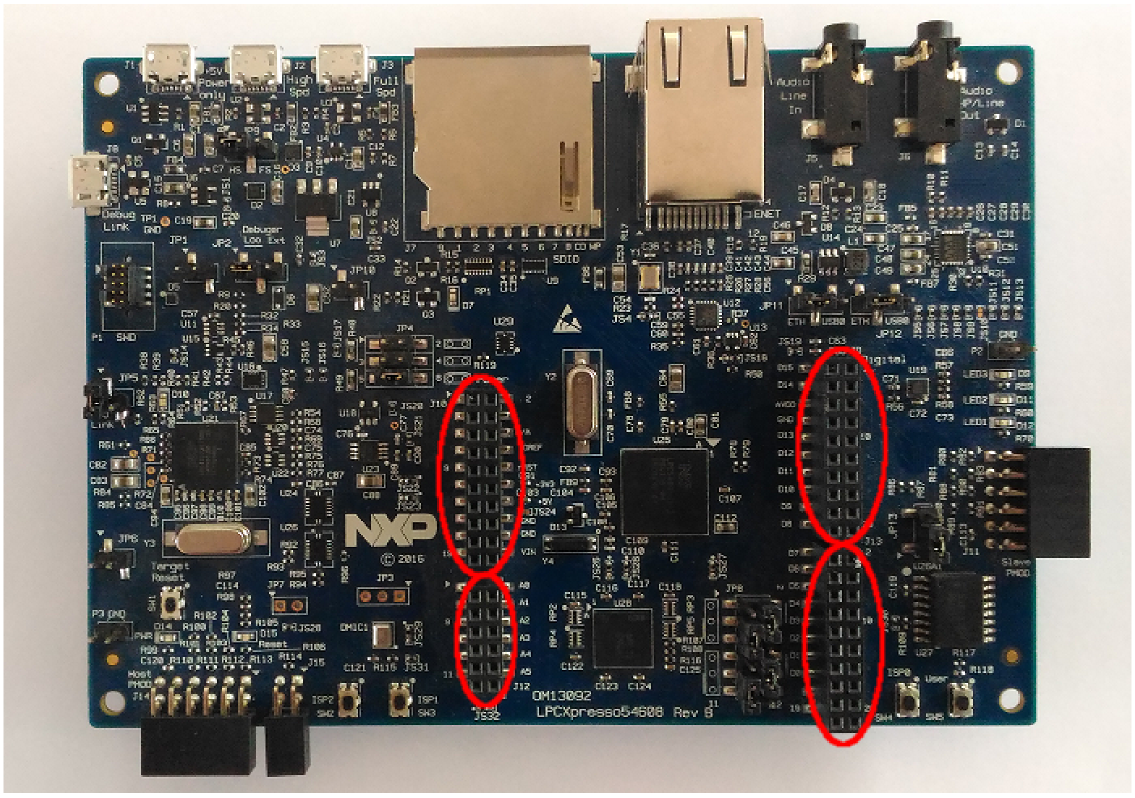
Figure 4.1. Mandatory Connectors



4.2. HAL Connectors

OM13098 provides several HAL GPIOs on Arduino connectors

Figure 4.2. HAL Connectors



Chapter 5. IAR Embedded Workbench Configuration

5.1. Install IAR Embedded Workbench

This section describes how to install IAR Embedded Workbench.

5.1.1. Download IAR Embedded Workbench

- Go to `https://www.iar.com/iar-embedded-workbench/`.
- Press the `Free trials` button in the `Downloads` tab.
- Press the `Download Software` button in the `IAR Embedded Workbench for ARM` element list.
- Run executable file and follow installation steps. Install additional software and drivers if proposed. A new application named `IAR Embedded Workbench xx.yy.z` shall have been created.

5.2. BSP Project Structure

The IAR BSP project folder is included in a MicroEJ standard project. This project is visible from the MicroEJ workspace. This project uses the same tree than the computer file system:

- Drivers: all MCU drivers, board drivers and CMSIS drivers
- Middlewares: all 3rd-party files: FatFs, FreeRTOS, LwIP, etc
- Projects: the MicroEJ platform project itself

The IAR BSP project file is `Projects/IAR/lpcxpresso.eww`. This IAR BSP project has been written for IAR Embedded Workbench for ARM 8.32.1. The project has the following structure:

- board: all board specific setup and configuration files
- drivers: all MCU drivers, board drivers and CMSIS drivers
- MicroEJ: MicroeEJ libraries
- Middlewares/*: all the third-party stacks files
- src: Core Engine implementation over FreeRTOS (always required)
- src-fs: File system implementation over FatFS
- src-hal: HAL implementation
- src-kf: Multi applications implementation

- src-net: Network implementation over LWIP
- src-ssl: SSL implementation over WolfSSL
- src-ui: UI implementation
- startup: initialization routines
- utilities: traces and debug definitions

Chapter 6. Changelog

6.1. [2.0.1] - 2019-11-30

Fixed:

- Documentation is now compliant with MicroEJ distribution 19.05.

6.2. [2.0.0] - 2019-09-16

Added:

- Add support for LPCXresso54628 board(OM13098).
- Add Qualification Tool.
- Add support for Winbond W9812G6JB external RAM chip.

Changed:

- Clock increase to 220 MHz.
- Change number flash access cycle from 9 to 8.
- Update iar_lib_power.a to version from SDK 2.6.
- Merge clock and EMC support from SDK 2.6.
- Improve performance by moving MicroEJ VM Core from flash to internal RAM.

Removed:

- LPCXpresso54618 (OM13094) and LPCXpresso54608 (OM13092) board are not supported anymore.

6.3. [1.2.1] - 2019-08-13

Added:

- UI pack updated from version 12.0.0 to 12.0.1

Changed:

- Fix platform for board using Winbond flash: W25Q128JVFM (Rev E and up plus some rev D).

6.4. [1.2.0] - 2019-07-15

Added:

- Architecture: Updated from 6.18.0 to 7.11.0.
- Standalone pack updated from version 5.4.3 to 6.0.0
- UI pack updated from version 10.0.0 to 12.0.0
- FS pack updated from version 3.0.2 to 4.0.2
- Compiler updated from IAR Embedded Workbench for ARM version 7.4 to 8.3.
- Add SEEGER system view support.

Changed:

- Reduce SD card detection time when there is no SD card.

6.5. [1.1.1] - 2018-07-20

Added:

- Add RSA-SHA256 signature native.

6.6. [1.1.0] - 2018-05-15

Changed:

- Architecture: Updated from 6.9.0 to 6.18.0, bringing performance improvements and bug fixes.
- UI: Updated from 9.0.2 to 10.0.0, bringing performance improvements and bug fixes.
- FS: Updated from 3.0.0 to 3.0.2, bringing bug fixes.
- HAL: Updated from 1.0.4 to 1.0.6, bringing bug fixes.

6.7. [1.0.0]

Initial release of the platform.