MICROEJ SDK 5 Basics

T

For STM32F7508-DK

© MICROEJ 2024



DISCLAIMER

All rights reserved. Information, technical data and tutorials contained in this document are proprietary under copyright law of MicroEJ S.A. Without written permission from MicroEJ S.A., copying or sending parts of the document or the entire document by any means to third parties is not permitted. Granted authorizations for using parts of the document or the entire document do not mean MicroEJ S.A. gives public full access rights.

The information contained herein is not warranted to be error-free.

MicroEJ® and all relative logos are trademarks or registered trademarks of MicroEJ S.A. in France and other Countries.

Other trademarks are proprietary of their respective owners.

Java[™] is Sun Microsystems' trademark for a technology for developing application software and deploying it in cross-platform, networked environments. When it is used in this site without adding the """ symbol, it includes implementations of the technology by companies other than Sun. Java[™], all Java-based marks and all related logos are trademarks or registered trademarks of Sun Microsystems Inc, in the United States and other Countries.



AGENDA



WHAT YOU WILL LEARN

By the end of this training, you will be able to use MICROEJ SDK to:

- Build a MICROEJ VEE Port.
- Build and Run a Java Application.
- Edit a Front Panel File.
- Create your own Foundation Libraries.
- Call a C function from Java.



REQUIREMENTS

- STM32F7508-DK Board and a mini-USB cable.
 - Clone the <u>STM32F7508 2.1.2 VEE Port</u> repository using the Git --recursive option to get the submodules.
- Download the flopi7G26-8.0.0-eval.xpf MICROEJ Architecture for Cortex M7 GCC (<u>https://repository.microej.com/modules/com/microej/architecture/CM7/CM7hardfp_GCC48/flopi7</u> <u>G26/8.0.0/</u>).
- Windows 10 or 11 64-bit:
 - Install JDK 11 64-bit (<u>https://adoptopenjdk.net/?variant=openjdk8&jvmVariant=hotspot</u>).
 - Note: select the "JavaSoft (Oracle) registry keys" feature in the installer
 - Install MICROEJ 23.07 SDK (<u>https://repository.microej.com/packages/SDK/23.07/MicroEJ-SDK-Installer-Win64-23.07.exe</u>).
 - Install a serial terminal (<u>https://www.compuphase.com/software_termite.htm</u>).
 - Install **STM32CubeIDE 1.9.0** (<u>https://www.st.com/en/development-tools/stm32cubeide.html</u>).
 - Access to internet and the MICROEJ Central Repository (<u>https://repository.microej.com/</u>).



INSTALLING MICROEJ ARCHITECTURE

IMPORTING ARM CORTEX-M7 GCC ARCHITECTURE

- Download & Install MICROEJ SDK (see download link in slide 5).
- Once installed, launch the MICROEJ SDK and select the default workspace.
- In MICROEJ SDK, click on Window > Preferences > MICROEJ > Architectures > Import.
- Select the MICROEJ Architecture previously downloaded flopi7G26-{version}-eval.xpf (see download link slide 5).
- Accept the license terms and click on Finish.
- The architecture is now imported.
- Click on **Apply and Close** button.

Name	Version	Select All
ARM Cortex-M7 GCC EVAL	8.0.0	D <u>e</u> select All
MicroEJ SDK EULA		
THIS SOFTWARE LICENSE AGREEMENT (TH On purchase of any Licensed Product from THE LICENSEE, AS A USER OF THE LICENSE 1 DEFINITIONS	E « AGREEMENT ») APPLIES TO n MICROEJ or an MICROEJ Partr D PRODUCTS REFERRED TO AB	PRODUCTS LICENSED BY IND ner or an MICROEJ Distributor, OVE AND ON THE RELEVANT I
THIS SOFTWARE LICENSE AGREEMENT (TH On purchase of any Licensed Product from THE LICENSEE, AS A USER OF THE LICENSE 1 DEFINITIONS	E « AGREEMENT ») APPLIES TO n MICROEJ or an MICROEJ Partr D PRODUCTS REFERRED TO AB	PRODUCTS LICENSED BY IND ther or an MICROEJ Distributor, OVE AND ON THE RELEVANT I
THIS SOFTWARE LICENSE AGREEMENT (TH On purchase of any Licensed Product from THE LICENSEE, AS A USER OF THE LICENSE 1 DEFINITIONS	HE « AGREEMENT ») APPLIES TO In MICROEJ or an MICROEJ Partr ED PRODUCTS REFERRED TO AB conditions and I want to install	PRODUCTS LICENSED BY IND her or an MICROEJ Distributor, OVE AND ON THE RELEVANT I > the copyrighted Software



ACTIVATING MICROEJ ARCHITECTURE LICENSE

GETTING THE UID

- In MICROEJ SDK, go to Window > Preferences > MICROEJ > Architectures.
- Select the **ARM Cortex-M7 GCC EVAL** Architecture.
- Click on Get UID.
- Copy the UID. It will be needed when requesting a license.

Petrice core	Architectures			↓ → ↓
General Ant C/C++	Add or remove Architectures. Target:			
Checkstyle	Name	Version	Lic	Select All
Help	ARM Cortex-M7 GCC EVAL	8.0.0	×	Decelect All
Install/Opdate				Deselect All
MicroEJ				Import
Architectures				Uninstall
Module Manager				
Naming Convention				Get UID
Platforms				2
Settings	UID successfully generated			
Updates	Your UID was successfully generated.			
Virtual Devices				
Mylyn				
Plug-in Development	Your UID is:			
				OK
PMD				1 10
PMD Run/Debug				OK
PMD Run/Debug SonarLint				
PMD Run/Debug SonarLint Team Terminal				



ACTIVATING MICROEJ ARCHITECTURE LICENSE

GENERATING THE ACTIVATION KEY

• Go to <u>license.microej.com</u>.

- Click on Create a new account link.
- Create your account with a valid email address. You will receive a confirmation email a few minutes after. Click on the confirmation link in the email and login with your new account.
- Click on Activate a License.
- Set Product P/N: to 9PEVNLDBU6IJ.
- Set UID: to the UID you generated before.
- Click on Activate.
- The license is being activated. It can be downloaded from the home page of <u>license.microej.com</u>.
- Once generated, download the attached zip file that contains your activation key.

Activate a MicroEJ License

Product P/N

Activate

UID: *

Once you downloaded and installed MicroEJ SDK, you have to activate your license to start developing, even in case of a free trial license. To activate a license, please enter your Part Number (P/N) and UID:

Part Number is a 12-digit number that you can find on the <u>MicroEJ SDK Getting Started page</u>

• UID is a 16-digit number available from your MicroEJ SDK, or a 8-digit number attached to your USB dongle

9PEVNLDBU6IJ	



ACTIVATING MICROEJ ARCHITECTURE LICENSE

ACTIVATING MICROEJ SDK

- In MICROEJ SDK, go to Window > Preferences > MICROEJ.
- Press Add..
- Browse the previously downloaded activation key archive file.
- Press **OK**. A new license is successfully installed.





VEE Port Concept

Computing platform for embedded system development



VEE PORT

- MICROEJ SDK brings the concept of **computing platform** to embedded system development
- Goals of this presentation:
 - Why computing platforms help to develop applications
 - How to make a platform with MicroEJ SDK?
- Computing platform = software platform = platform = VEE Port



STATE OF PLAY

- Programs made for workstations and servers are portable to Linux / OS X / Windows
- iOS or Android let you run the **same application on several hardware** targets
- Developers use **high level languages** and tools
- Low level actions are delegated to the operating system (OS)
 - Why should not we do the same for embedded devices?



VEE PORT AND ABSTRACTION

APPLICATION FEATURES ARE SPLIT IN 2 CATEGORIES

- 1. Hardware dependent features (ex: screen): into the VEE Port, hiding details of what **might change**
- 2. Hardware-independent features:
 - Mathematical algorithms
 - Software using the VEE Port functionalities
 - UI
 - Connectivity protocols
 - Business logic

AICROEJ

PURPOSE OF ABSTRACTION

- Hardware abstracted software is the key point for **portability**
- Portability is needed when
 - You want to **reuse** the same code for several projects
 - Your hardware platform becomes **obsolete**
 - You target several hardware platforms with the same application
- When switching to a new hardware platform
 - You only change the hardware specific parts
 - You re-create an **iso-functional** computing VEE Port
 - Your software runs identically on this new VEE Port



VIRTUAL EXECUTION ENVIRONMENT

Overview



MICROEJ VEE

MICROEJ VEE is a scalable Virtual Execution Environment for **resource-constrained** embedded and IoT devices running on 32-bit microcontrollers or microprocessors.

MICROEJ VEE allows devices to **run multiple and mixed Java and C** software applications.

Key Figures:

- Boots in 2 ms on a Cortex-M4 @180MHz.
- Optimized for low-power.
- Compact (< 30 KB footprint).
- Runs from Cortex-M0 with 128 KB flash and 32 KB RAM, to Cortex-A7.

MICROEJ VEE

SERVICES

MICROEJ VEE provides a fully configurable set of services that can be expanded, including:

- A secure **multi-application** framework.
- A **network connection with security** (SSL/TLS, HTTPS, REST, MQTT, CoAP,...).
- A GUI framework (includes widgets).
- A basic analog and digital IO framework.
- A sensor framework.
- A storage framework (file system).

As it runs Java, MICROEJ supports all security, networking and IoT communication protocols and frameworks such as MQTT, CoAP, etc.

MICROEJ VEE – DETAILED VIEW





Se MICROEJ

GUI EXAMPLES FOR \$1 TO \$5 MCU







Iter your credentials
Username
Remember me on this phone
Valider













© MICROEJ 2024



BUILD FLOW

Build flow explained



OVERVIEW

WORKBENCH

MICROEJ SDK includes:

- MICROEJ IDE, based on Eclipse.
- Tools to **build VEE Ports.**
- Tools to **build applications.**
- Add-on libraries to code with Java as high-level language.
- Native libraries and mechanism to allow developers to use C and to create **interactions between C and Java** features.
- Support for Eclipse plugins.



MICROEJ ARCHITECTURE

- A **MICROEJ Architecture** is a software package that includes the MEJ32 port to a target instruction set and a C compiler, MICROEJ Foundation Libraries and the MEJ32 Simulator.
- MICROEJ Architectures are provided by MICROEJ and distributed within MICROEJ SDK.
- Menu Window > Preferences > MicroEJ > Architectures.
- Example of MICROEJ Architectures:
 - ARM Cortex-M4 Keil ARM Compiler 5.
 - Renesas RXv2 IAR 8.0.
 - ARM Cortex-A7 GCC 5.3 Linaro Linux HardFP.
- List of the architectures:
 - https://developer.microej.com/mej32-embedded-runtime-architectures/

pe filter text	Architectures		← → ⇒ →
General Ant C/C++ Checkstyle Easyant4Eclipse Help Install/Update Ivy Java MicroEJ Architectures Naming Conventior Platforms Platforms in worksp Settings Updates Mylyn Plug-in Development PMD Bun/Debug	Architectures Add or remove architectures. Platforms, Virtual Devices and Architectures: Name	Version License 7.4.0 ✓ 7.7.0 ✓ 6.9.0 ✓ 6.9.0 ✓ 6.9.0 ✓ 6.9.0 ✓ 6.9.0 ✓ 7.4.0 ✓	Select All Deselect A Import Uninstall Get UID
Run/Debug			
Terminal			
/alidation v			
		R	estore <u>D</u> efaults <u>A</u> pply



MICROEJ VEE PORT

- A **MICROEJ VEE Port** is a port of a MICROEJ Architecture for a specific hardware, RTOS and BSP.
- MICROEJ VEE Ports are built using MICROEJ SDK.
- They are distributed as source (including C sources) or binary (pre-built C BSP).
- Menu Window > Preferences > MicroEJ > Platforms.
- Example of MICROEJ VEE Port:
 - Renesas S7G2-DK ThreadX SSP 1.3.
 - NXP OM13092 FreeRTOS KSDK.
 - Atmel SAMA5-Xplained Linux.
- List of the platforms:
 - <u>https://developer.microej.com/supported-hardware/</u>

Streferences						×
type filter text		Platforms			⇔ -⇒	
 > General > Ant > C/C++ Checkstyle Easyant4Eclipse > Help > Install/Update > lvy > Java > MicroEJ Architectures Naming Convention Platforms Platforms in workspace Updates 	^	Add or remove platforms. Platforms, Virtual Devices and Architectures: Name GR-peach MultiApp Production [EWCPC] OM13092 SingleApp Production [ZRLE0] STM32F746G-DISCO SingleApp Production [K1AU3]	Version 1.5.2 0.10.0 3.1.5	License V V	Select A Deselect Import. Uninsta	All All all D
> Mylyn > Plug-in Development	~		Resto	re <u>D</u> efaults	Appl	у
?				ОК	Cance	I



BUILD FLOW / VEE PORT





BUILD FLOW / FIRMWARE



V5.1 Aug. 2024 25



Build VEE Port

For STM32F7508-DK



IMPORT VEE PORT SOURCES

- File > Import... > General > Existing Projects into Workspace.
- Click the directory where
 VEEPort-STMicroelectronics-STM32F7508-DK has been cloned.
- The projects appears in the Projects list. Select the following ones:
 - **stm32f7508_freertos-configuration**: the configuration project used to configure the VEE Port
 - **stm32f7508_freertos-fp**: the front panel project. It describes the UI of the simulator
 - **stm32f7508_freertos-bsp**: contains the Board Support Package (BSP) source code
 - STM32F7508-Platform-CM7hardfp_GCC48-{version}: the VEE Port project (empty)
- Click on **Finish**.

💡 Import	-	
mport Projects Select a directory to sear	ch for existing Eclipse projects.	
Select root directory:	C:\workspaces\ \V ~	Browse
Select archive file:	Browse	
Projects:		_
stm32f7508_freert	tos-bsp (C:\workspaces\ tos-configuration (C:\workspaces\	Select All
stm32f7508_freert	tos-fp (C:\workspaces	Deselect All
stm32f7508_freert	tos-validation (C:\workspaces\ Port-CM7hardfp_GCC48-2.1.2 (C:\workspaces\	Refresh
Search for nested pro Copy projects into w	ojects vorkspace	
Close newly importe	d projects upon completion ready exist in the workspace	
Working sets		
Add project to work	king sets	New
Working sets:	~	Select

BUILD STM32F7508 VEE PORT

- Right click on stm32f7508_freertos-configuration project
- Click on **Build Module** to build the VEE Port.
- The VEE Port project **STM32F7508-Platform-CM7hardfp_GCC48** is now filled.
- You can see the VEE Port in **Platforms in workspace** menu:
 - Window > Preferences > MicroEJ > Platforms in workspace





Application

Build & Run

APPLICATION CREATION

JAVA PROJECT CREATION

- Go to File -> New -> MicroEJ Standalone
 Application Project.
- Fill the input fields.

😜 New Microf	EJ Standalone Application Project — 🛛 🗙	
Create a Stand	dalone Application project	
Enter project n	ame and configure your application.	7
Project:		
Project name :	HelloWorld]
Application:		
Publication :		
Organization :	com.microej.training	
Module :	HelloWorld	
Revision :	0.1.0]
?	< Back Next > Finish Cancel	
		_



BUILD FLOW





APPLICATION CREATION

JAVA PROJECT LAUNCHER

- Right-Click on the Project.
- Run As -> Run Configuration.
- Double click on MICROEJ Application.
- Go to **Execution** tab.
- Select **Execute on Device.**

Que Run Configurations			— 🗆 X
Create, manage, and run configurat	ions		
Image: Second system Image: Second system Image: Secon	Name: HelloWorld Main	A JRE 15 Source Common GCC48 (2.1.2)	Browse
	Options Output folder: S{project_loc:HelloWorld} Clean intermediate files	Uerbose	Browse
	Options Files		Add Remove Up Down
Filter matched 6 of 13 items		Re	evert Apply
?			Run Close



BUILD FLOW

© MICROEJ 2024





MICROEJ AND 3RD PARTY IDE

- Java application must be **linked with BSP:**
 - BSP = drivers + (optional: operating system) + abstraction layer.
 - Done by a 3rd party IDE.
- MicroEJ provides:
 - Java application as an **object file (microejapp.o).**
 - Java runtime environment as a **library file (microejruntime.a).**
 - **Header files** with types and functions provided by this library (.h).
 - Abstraction layer interface (.h).
 - Abstraction layer implementation (.c, .cpp).
- 3rd party IDE is responsible for **compiling BSP, linking, and generating an executable file.**



RUN THE JAVA APPLICATION ON DEVICE

IMPORT THE BSP PROJECT

- Open STM32CubeIDE in an empty workspace.
- Select File > Import...
- Select General > Existing Projects into Workspace.
- Press **Next**.
- Next to the **Select root directory** field, press **Browse...**
- Navigate to the **stm32f7508_freertos-bsp/projects/microej/SW4STM32** folder.
- Select the **application** project.
- Press **Finish**.



RUN THE JAVA APPLICATION ON DEVICE

BUILD AND FLASH THE BSP

- In STM32CubeIDE, right-Click on the **application** project.
- Press Build Project.
- Wait for the end of the build.
- Plug the STM32F7508-DK board to the PC thanks to a mini-USB cable (CN14 USB ST-Link connector).
- In STM32CubeIDE, select **Run > Run Configurations...**
- Under STM32 Cortex-M C/C++ Application, select the application_debug run configuration.
- Press Run.
- The firmware will be downloaded on the STM32F7508-DK.


RUN THE JAVA APPLICATION ON DEVICE

GET THE APPLICATION TRACES

- Open the Termite serial terminal.
- Click the **Settings** button.
- Select the STM32F7508-DK board COM port.
- Reset the STM32F7508-DK board pressing the **black** button.
- The application starts and the **Hello World** message is printed in the console!

🚯 Termite 3.4 (by CompuPhase)			-		×
COM7 115200 bps, 8N1, no handshake	Settings	Clear	Abo	ut	Close
Start MCU revision identifier: 0x1001 MCU device identifier: 0x449 watchdog started MicroEJ START Hello World! MicroEJ END (exit code = 0)					
					•



38

MICROEJ CORE ENGINE STARTUP

- MicroEJ header files are in: projects/microej/SW4STM32/platform/inc
 - 🗸 🖳 platform
 - ✓ (⇒ inc > (⇒ intern)
 - > h BESTFIT_ALLOCATOR_impl.h
 - > h BESTFIT_ALLOCATOR.h
 - > h LLAT_impl.h
 - > 🔥 LLBSP_impl.h
 - > h LLCOMM_BUFFERED_CONNECTION_impl.h
 - > h LLCOMM_BUFFERED_CONNECTION.h
 - > h LLCOMM_CONNECTION.h
- MicroEJ libraries and Java application object file are used during link edition:
 - ✓ A platform
 > ⇒ inc
 > ⇒ lib
 ⇒ microejapp.o
 ⇒ microejruntime.a
- MicroEJ Core Engine is invoked in: projects/microej/core/src/microej_main.c with SNI_createVM():

// create VM	Note: in the STM32F7508 VEE Port, microej_main() is called	
<pre>vm = SNI_createVM();</pre>	from a FreeRTOS task in main.c.	
—	It is also possible to run MicroEJ Core Engine on a baremetal	
© MICROEJ 2024	device (no RTOS).	V5.1 Aug. 2024



APPLICATION

Configuration

LIBRARY DEPENDENCY FILE

Contains a description of all the libraries required by the application.



flow module.ivy

Loaded by the MicroEJ Module Manager (MMM) to fetch automatically the dependencies using Ivy.

Available MICROEJ libraries can be found here:

- <u>https://repository.microej.com/</u>
- <u>https://forge.microej.com/artifactory/microej-developer-repository-release/</u>

From the MICROEJ Javadoc you can search for a Class and get the MMM dependency that provides it by visiting <u>https://repository.microej.com/javadoc/microej_5.x/apis/index.html</u>

Se MICROEJ

Library dependency file



LIBRARY DEPENDENCY FILE

Example:

https://repository.microej.com/javadoc/microej_5.x/apis/index.html?ej/restserver/RestServer.html This button let you copy the MMM dependency directly into the clipboard.

All Classes	OVERVIEW PACKAGE CLASS TREE INDEX HELP	Use this class in your project	
Packages	PREV CLASS NEXT CLASS FRAMES NO FRAMES	In order to use this class in your project, add this dependency in your project build file:	
android.net	SUMMARY: NESTED FIELD CONSTR METHOD DETAIL: FIELD CONSTR METHOD		
com.microej.http.util	ej.restserver	SDK 6 (build.gradie.kts)	
com.microej.kf.util com.microej.kf.util.message	Class RestServer	implementation("ej.library.iot:res 🖽	
com.microej.kf.util.observable com microei kf.util.security	java.lang.Object	SDK 5 (module.ivy)	
com.microej.nls	HTTPServer	<pre><dependency n<="" org="ej.library.iot" pre=""></dependency></pre>	
com.microej.soar			
All Classes	whis class DestConver		
AbstractAllTestClassesWrapper	extends HTTPServer		
AbstractCollection			
AbstractContent AbstractDelegateHttpsURLConnection	tractContent A simple implementation of HTTP server capable of easily exposing endpoints.		
AbstractJSONExpr	REST servers can always handle requests that target endpoints. Other kind of request are handles by custom request handlers.		
AbstractKFList AbstractList	See Also:		
AbstractMap AbstractMap.SimpleEntry	EndpointHandler, RequestHandler		
AbstractMap.SimpleImmutableEntry			
AbstractMessageLite AbstractMessageLite.Builder	Constructor Summary		
AbstractMethodError AbstractPackedMap			
AbstractParser	Constructors		
AbstractRemoteService Constructor and Description			
AbstractResource AbstractSequentialList	<pre>RestServer(int port, int maxSimultaneousConnection, int jobCountBySession)</pre>		
AbstractSet AbstractTestWrapper	Creates a new server bounded to given port.		
AbstractWeakPackedMap	RestServer(int port, int maxSimultaneousConnection, int jobCountBySession, ServerSocketFactory	serverSocketFactory)	
AccessPoint	Creates a REST server on top of the given HTTP server.		



RUN CONFIGURATIONS

- Run Configurations:
 - Eclipse provides the concept of "run configurations"
 - A run configuration tells what is executed, what is the runtime environment, what are the execution options
 - Available through the Run menu
- A Run Configuration can be executed as:
 - A Run Configuration to simply run an application •
 - A Debug Configuration to debug this application
- External Tool Configuration to run an external program



RUN CONFIGURATIONS

MICROEJ provides two specific run configuration types:

- MICROEJ Application
- MICROEJ Tool







MICROEJ APPLICATION

KIND OF EXECUTION (SIMULATOR OR DEVICE)

Run Configurations			— D >
reate, manage, and run configuration	ons		
] 🖻 🎨 🗎 🗙 🖻 🏹 🗸	Name: My Configuration		
type filter text	🗊 Main 📣 Execution 👫 Configuration 🛋 J	RE 🥪 Source 🔲 Common	
C/C++ Application	Target		^
Ju JUnit	Platform: STM32F7508-Platform-CM7hardfp_G	CC48 (1.5.0)	Browse
My Configuration MicroEl Tool	Execution		
	Execute on Simulator	O Execute on Device	
		Core Engine Mode: Default	~
	Settings: Default	✓ Settings: Build & Deploy	\checkmark
	The Application is simulated		
	Options		
	Output folder: \${project_loc:MyProject}		Browse
	Clean intermediate files	Verbose	
	Options Files		
			Add
			Remove
			Up
			Down
ter matched 6 of 12 items			Revert Apply
2		Γ	Run Close



MICROEJ APPLICATION

CONFIGURE LIBRARIES AND MEMORY USAGE

😜 Run Configurations			- 🗆 X
Create, manage, and run configurations			
Image: Second system Image: Second system Image: Second	Name: My Configuration	Java heap size (in bytes) Java heap size (in bytes) Immortal heap size (in bytes) Threads Number of threads Number of blocks in pool Block size (in bytes) Maximum size of thread stack (in blocks)	65536 4096 5 15 512 4
Filter matched 6 of 12 items			Revert Apply
?			Run Close



RUN CONFIGURATION

SHARE RUN CONFIGURATIONS

- 1. Go to **Run -> Run Configurations**
- 2. Select a run configuration
- 3. In **Common** tab, select **Save as Shared file** and choose the directory where it is saved
- 4. You can now commit the **.launch** file in your Version Control System

- ⊈ ⊒ 🗶	Name: Main	
type filter text	🗊 Main 📣 Execution 🔐 Configuration 🛋 JRE 🧤 Source 🔲 Common	
C/C++ Application	Save as	_ ^
J u JUnit	🔿 Local file	
🚭 Launch Group	Shared file:	1
Launch Group (Deprecated)	Theiloworld Browse	1
✓ J MicroEJ Application	Display in favorites menu	
J Main	Default - inherited (UTF-8)	



Customization

V5.1 Aug. 2024 47

PRINCIPLE

- MICROEJ environment allows applications to be developed and tested in a Simulator rather than on the target device, which might not yet be built.
- To make this possible for devices operated by the user, the Simulator must connect to a "mock" of the control panel (the "Front Panel") of the device.
- The Front Panel generates a graphical representation of the device, and is displayed in a window on the user's development machine when the application is executed in the Simulator.
- The Front Panel implements MicroUI. However it can be use to show a hardware device, blink an LED, interact with user without using MicroUI library.

See <u>https://docs.microej.com/en/latest/PlatformDeveloperGuide/frontpanel.html</u>



PROJECT CONTENT

A Front Panel project has the following structure and contents:

- **src/main/java (optional):** contains custom widgets and button event listeners.
- **src/main/resources:** holds files that define the contents and layout of the Front Panel (**.fp** file and images).
- JRE System Library: required to compile the custom widgets and listeners.
- Modules Dependencies: contains front panel framework and default widgets.
- **lib/:** contains a local copy of Modules Dependencies.



🔑 MICROEJ,

FRONT PANEL

FRONT PANEL FILE

- Description written in XML (.fp file): <device ...> element contains the elements that define the widgets that make up the Front Panel.
- Loaded by the Front Panel Engine to build the graphical representation of the real device.
- Declare the widgets that simulate the drivers, sensors, and actuators of the real device.

```
1 ∨ <frontpanel
2
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3
        xmLns="https://developer.microej.com"
        xsi:schemaLocation="https://developer.microej.com .widget.xsd">
4
5
6 ~
         <device name="STM32F7508DK" skin="Board stm32f7508.jpg">
             <ej.fp.widget.Display x="71" y="41" width="480" height="272"/>
7
             <ej.fp.widget.Pointer x="71" y="41" width="480" height="272" touch="true"/>
8
             <ej.fp.widget.Button label="0" x="19" y="196" skin="But0.png" pushedSkin="But1.png" listenerClass="com.is2t.microej.fp.ButtonListener"/>
9
             <ej.fp.widget.LED label="0" x="30" y="258" ledOff="LedOff.png" ledOn="LedGreen.png"/>
10
         </device>
11
     </frontpanel>
12
```

- Widgets:
 - The name of the widget element references the Java class of the widget (see widget-x.y.z.jar in Module Dependencies).
 - A widget can be identified by a label, which must be unique for the widgets of the same type.
- ^{© MICROEJ 2024} Position specified with x and y attributes.



EDITING THE FRONT PANEL

- To edit a .fp file, open it using the Eclipse XML editor:
- Right-Click on the .fp file, select Open With > XML
 Editor and select the Source tab.
- Within the XML editor, content-assist is obtained by pressing **CTRL + SPACE** keys.
- To obtain a preview of the Front Panel, go to Window > Show View > Other... > MICROEJ > Front Panel Preview.
- The preview is updated each time the **.fp** file is saved.
- The VEE Port needs to be rebuilt to get the Front Panel updates.

x *stm	32f7508_freetos-fp.fp 🔀	- 6
10		^
110 <	frontpanel	
12	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"	
13	xmlns="https://developer.microej.com"	
14	xsi:schemaLocation="https://developer.microej.com .widget.xsd">	
15		1
160	<device name="STM32F7508DK" skin="Board stm32f7508.jpg"></device>	
17	<ej.fp.widget.display height="272" width="480" x="71" y="41"></ej.fp.widget.display>	
18	<ej.fp.widget.pointer height="272" touch="true" width="480" x="71" y="41"></ej.fp.widget.pointer>	
19	<pre><ei.fp.widget.button <="" label="0" listenerclass="com.is2t.microei.fp.ButtonListener</pre></td><td>" pushedskin="But1.png" skin="But0.png" td="" x="19" y="196"></ei.fp.widget.button></pre>	
20	<ei.fp.widget.led label="0" ledoff="LedOff.png" ledon="LedGreen.png" v="258" x="30"></ei.fp.widget.led>	
21		
22 <	/frontpanel>	
23		~
		>
lecian	Source	

🔄 Console 📳 Problems 🖷 Progress 🔚 Git Repositories 💷 stm 32f7508_freertos-fp.fp 🔀





MICROEJ SDK

Tools



EXCEPTION GENERATION

- By default, on error, the stack trace of the exception thrown is printed on the **serial console**.
- Let's generate an error. Add the following code in your HelloWorld main method:

```
byte[] array = new byte[5];
array[5] = 42; // Invalid access to the array
```

- Compile the application in MICROEJ SDK:
 - 1. Right click on the HelloWorld MICROEJ project.
 - 2. Run as -> MicroEJ Application.
- Build the BSP Project.
- Flash the board.





EXCEPTION OUTPUT

• In the console, we can see the stack trace:



- Name of the faulty **method is not printed** directly:
 - Only the address of the method is printed
 - MICROEJ does not embed the names of the methods to limit the footprint
- To help reading the stack trace, a tool is available: **the stack trace reader**



CONFIGURATION

In MICROEJ SDK, create the Run configuration

- 1. Go to Run -> Run Configurations...
- 2. Double-click on MicroEJ Tool.
- 3. Enter a name for the launcher.
- 4. Select your VEE Port.

© MICROEJ 2

- 5. Use settings: **Stack Trace Reader.**
- 6. Go to **Configuration** tab.
- 7. Use the ELF file generated by the 3rd party linker:

	Name: Stack Trace Reader
type filter text	💰 Execution 🔣 Configuration 🛋 JRE 🔲 Common
C/C++ Application	Target
Ju JUnit	Platform: \$TM32F7508-Platform-CM7hardfp_GCC48 (2.1.2) Browse
✓ ☑ MicroEJ Application	
(EMB) HelloWorld	Execution
V III (SIM) HelloWorld	Settings: Stack Trace Reader 🗸 🗸 🗸 🗸 🗸 🗸 V
The Stack Trace Reader	Reads stack trace generated by MicroEJ core engine.
	Options
	Output folder: \${project_loc:HelloWorld} Browse
	Clean intermediate files

Na	ame: Stack Trace Reader		
	Execution	🚮 JRE 🔲 Common	
	Stack Trace Reader	Application	
4		Executable file: 2f7508_freertos-bsp/projects/microej/SW4STM32/Debug/application.elf Browse	

.1 Aug. 2024 55



USAGE

- 1. Click **Run**
- 2. **Copy/Paste** the trace in your console

You can also configure it to read data directly from the com port of your device.

🗐 Console 🔀 🔝 Problems 🖷 Progress 🛷 Search 🍖 SonarLint Rule Description 🔲 Properties Ju JUnit
Stack Trace Reader_ [MicroEJ Tool] C:\Program Files\Java\jdk1.8.0_202\bin\javaw.exe (Mar 17, 2021 10:26:47 AM)
======================================
[INFO] Paste the MicroEJ core engine stack trace here.
Exception in thread "main" @C:0x00001470@
at @A:0x00002A84:0x00002B41@
at @A:0x00002E50:0x00002E63@
at @A:0x0000311C:0x00003124@
at @A:0x00003210:0x00003217@Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
at com.microej.example.Main.main(<u>Main.java:84</u>)
at java.lang.MainThread.run(<u>Thread.java:557</u>)
at java.lang.Thread.runWrapper(Thread.java:283)

Online documentation: <u>https://docs.microej.com/en/latest/ApplicationDeveloperGuide/stackTraceReader.html</u>

Debugger

- JDWP (Java Debug Wire Protocol) to use Eclipse debugger.
- Classical debugger features:
 - Breakpoints.
 - Step-by-step execution.
 - Variables and fields value monitoring.
 - Thread execution stacks list.
- Run your Launch Configuration as a Debug Configuration:
 - Debug perspective.



HEAP DUMPER

Irred module.ivy README.md

A heap file, describing the heap content, is created each time garbage collector is executed:

• **System.gc()** to force heap dumping:

— 🗆 X	😵 Debug Configurations		
Explorer 🛚 🕆 Type Hierarchy 🕒 🕏 🗢 🗖	Create, manage, and run configu	rations	
roject			
src/main/java			
/ 朣 com.mycompany			
> 🛽 Main.java		Name: My Configuration	
src/main/resources	type filter text	Main 🚅 Execution 🎟 Configuration	🔺 JRE 🦻 Source 🗉 Common
■ Ivy module.lvy [*] > build	Ju JUnit	> Libraries	Heap Inspection
com.mycompany.Main	✓ ☑ MicroEJ Application	> Runtime	
≥ bon	My Configuration	✓ Simulator	Activate neap dumper
🗁 externalResources	🖳 Remote Java Application	Code Coverage	
😂 heapDump		Com Port	
0 heap-0.heap		Debug	
heap-1.heap		Device	
heap-2.heap		FS	
🛿 heap-3.heap		HAI	
l heap-4.heap		Heap Dumper	
l heap-5.heap		Logs	
l heap-6.heap			
l heap-7.heap			
l heap 0 heap			
• resourceBuffer			
CHANGELOG.md			
LICENSE.txt			



HEAP DUMPER

- Open .heap files with the Heap Analyzer plugin.
- Inspect objects graph.
- Detect memory leaks.
- This is an advanced feature: a good knowledge of Java and the program is required.

HEAP DUMPER

🔄 Console 💦 Problems 🚦 Outline 🤮	g- Instance Browser 🕱	
References - heap file name :	Demo-Widget\ej.demo.ui.widget\ej.demo.ui.widget.WidgetsDem	o\heapDump\heap-20.heap
Field ✓ ③ this	Type Gej.demo.ui.widget.page.AbstractDemoPage\$3	Value #6103
✓ [] [8]	🔄 java.lang.Object[]	#12494 (40 items)
✓ () elementData	🕒 java.util.ArrayList	#966
🗊 myLeak	G ej.demo.ui.widget.page.AbstractDemoPage	Type ej.demo.ui.widget.page.Abstr



MEMORY MAP INSPECTOR

A *SOAR.map* file is generated when a build for device is done. The map file maps Java and MICROEJ memory usage (no BSP).

] Main.java Mt SOAR.map ⊠				
Name	Image Size	Runtime Size	Image Sizes	
> O All	20.6 KB	50.7 KB		
> ApplicationCode	352 B	0 B	IMAGE: 72 B / 20.6 KB	
> ApplicationImmutables	44 B	0 B	0.34%	
> ApplicationResources	20 B	0 B		
> ApplicationStrings	1.8 KB	0 B	Runtime Sizes	
> 🔴 BSP	848 B	2.9 KB		
> 🔵 ClassesNames	1.2 KB	0 B		
> O CoreEngine	904 B	7.5 KB		
> O CoreEngineAllocator	0 B	36.0 KB		
> 🕒 LibFoundationBON	72 B	0 B		
> O LibFoundationEDC	12.1 KB	73 B		
> O LibFoundationMicroUI	1.5 KB	4.1 KB		
 LibFoundationNET 	72 B	0 B		
SECTIONrodata.com.is2t.libraries.netember	12 B	0 B		
SECTIONrodata.com.is2t.libraries.netember	12 B	0 B		
SECTIONrodata.com.is2t.libraries.netember	12 B	0 B		
SECTIONrodata.com.is2t.libraries.netember	24 B	0 B		
SECTIONrodata.com.is2t.libraries.netember	: 12 B	0 B		
> 🕘 LibFoundationSP	112 B	0 B		
> ONATIVESTACKECOM_COMM	1.4 KB	32 B		
			Run additional Memory Map Script	
			Browse Run	
			Select a Memory Map Script to run	





CODE COVERAGE

Code coverage reports:

- List used and unused source code.
- Find untested or dead code.
- HTML report generation.

Image: Second s	Name: My Configuration			
Ju JUnit MicroEJ Application My Configuration MicroEJ Tool	 Libraries BasicImmutablesNLS ECOM EDC FS MicroUI Net SSL Shielded Plug Runtime Memory Simulator 	Code Coverage Activate code coverage analysis Saving coverage information period (in		

😂 Run Configurations



CODE COVERAGE

Image: Run Configurations X		\$
Create, manage, and run configurations		Package Explorer 😂
	Name: Code Coverage Report	✓
type filter text Jr JUnit ~ 12 MicroEJ Application 12 MicroEJ Tool 13 Code Coverage Report 24 Code Coverage Report 25 Code Coverage Report 26 Code Coverage Report 27 Clean interr	Execution ## Configuration Target Platform: STM32F7508-Platform-CM7hardfp_GCC48 (1.3.2) Browse Execution Settings: Code coverage report ~ Generate code coverage HTML report. Options Output folder: \$[project_loc:MyProject] Browse Verbose	 > src/main/java # src/main/resources > Ivy module.ivy [*] > build > com.mycompany.Main > bon > bon > cc > htmlReport > bytecode

- > 🗁 CSS
- icons
- > > > sources
- bytecode.html
- index.html
- methods_covered.html
- methods_uncovered.html
- methods.html
- output.html
- source.html
- com.mycompany.Main_1578414465155.cc



Dump the States of the Core Engine

LLMJVM DUMP

- What?
 - Prints the state of the MicroEJ Core Engine to the standard output stream.
 - For each Java thread, the Java stack trace, the name, the state and the priority are printed.
- How-To?
 - Call the C function **LLMJVM_dump()**.
 - It is provided by **LLMJVM.h**.
- When?
 - Call the **LLMJVM_dump** as a last resort in a fault handler to get a snapshot of the Core Engine, to check if the issue comes from a <u>LLAPI</u> or the underlying C code.
 - Call the **LLMJVM_dump** in the Core Engine task at runtime to diagnose unexpected behavior (ex: UI freeze).
- Requirements:
 - A way to read stdout (usually UART).

Se MICROEJ

LLMJVM DUMP EXAMPLE

```
void HardFault_Handler(void)
```

```
uint32_t hfsr;
print_stacked_registers();
puts(__func__);
```

```
hfsr = SCB->HFSR;
printf("Hard Fault Status Register =\t%lX\n", hfsr);
```

```
if(hfsr & SCB_HFSR_FORCED_Msk)
{
    printf("FORCED");
}
LLMJVM_dump();
```

```
INFINITE_LOOP();
```

• Note: the **Stack Trace Reader** can be used to decode the trace of the **LLMJVM_dump()**.

```
Java threads count: 3
Peak java threads count: 3
Total created java threads: 3
Last executed native function: 0x90035E3D
Last executed external hook function: 0x0000000
State: running
Java Thread[1026]
name="main" prio=5 state=RUNNING max java stack=456 current java stack=184
java.lang.MainThread@0xC0083C7C:
   at (native) [0x90003F65]
   at com.microej.demo.widget.main.MainPage.getContentWidget(MainPage.java:95)
      Object References:
Java Thread[1536]
name="Thread1" prio=5 state=READY max java stack=60 current java stack=57
java.lang.Thread@0xC0082194:
   at java.lang.Thread.runWrapper(Unknown Source)
     Object References:
         - java.lang.Thread@0xC0082194
   at java.lang.Thread.callWrapper(Thread.java:449)
State: Stopped
Last analyzed object: null
Total memory: 15500
Current allocated memory: 7068
Current free memory: 8432
Allocated memory after last GC: 0
Free memory after last GC: 15500
Τd
        CloseFunc Owner
                             Description
```

😂 MICROEJ.



SNI

SNI (Simple Native Interface) Call C code from Java



PRINCIPLE (1/2)

SNI Resolves native calls by executing them in another language (most of the time in C language).



Online documentation: <u>https://docs.microej.com/en/latest/PlatformDeveloperGuide/sni.html</u>



PRINCIPLE (2/2)

SNI provides a simple mechanism for implementing native Java methods in the C language.

SNI allows you to:

- Call a C function from a Java method.
- Access a Java array from a native method written in C.
- Access a Java Immortal array from another RTOS task, an interrupt handler, or a DMA (see the BON specification to learn about immortal objects).

SNI does not allow you to:

- Access or create a Java object in a C function.
- Access Java static variables in a C function.
- Call Java methods from a C function.

SNI provides some Java APIs to manipulate some data arrays between Java and the native (C) world.



NAMING CONVENTION

```
package com.corp.examples;
public class Hello {
    public static void main(String[] args){
        int i = printHelloNbTimes(3);
        }
        public static native int printHelloNbTimes(int times);
    }
}
```

```
#include <sni.h>
#include <stdio.h>

jint Java_com_corp_examples_Hello_printHelloNbTimes(jint times){
    while (--times){
        printf("Hello world!\n");
        }
    return 0;
}
```



DATA TYPES

- Primitive data type can be manipulated through SNI (return value and parameter):
 - byte, short, int, long, float, double, boolean, char.
- Arrays of primitive data type are managed by SNI with some limitations:
 - C globals, C Heap, DMA, RTOS tasks can reference only Immortal arrays.
 - Non-immortal arrays can be referenced only from a native function local.



Se MICROEJ

Implement a Java Native Method with SNI


ADD THE JAVA NATIVE METHOD

• Modify the code of the HelloWorld main method:

```
public static void main(String[] args) {
    System.out.println("Hello World!");
    printHelloNative();
}
```

```
public static native void printHelloNative();
```

- Compile the application in MICROEJ SDK:
 - Right click on the HelloWorld MICROEJ project.
 - Run as -> MicroEJ Application.
 - Run the launcher configured to **Execute on Device.**



GET THE LINKER ERRORS

- In STM32CubeIDE, click on your project once to select it.
- Go to **Project** > **Build Project**.
- Wait for the end of the build. The following error appears:

C:\workspaces\HelloWorld\com.microej.training.Main\SOAR.o:(.text.soar+0x1f78): undefined reference to `Java_com_microej_training_Main_printHelloNative'

• The printHelloNative() method is a native method. It must be implemented in the BSP.



IMPLEMENT THE NATIVE METHOD IN THE BSP

- In STM32CubeIDE, open microjvm_main.c
- Implement the **printHelloNative()** method, use the method signature provided by the linker error:

```
#include <stdio.h>
#include "microej_main.h"
#include "LLMJVM.h"
#include "sni.h"
void Java_com_microej_training_Main_printHelloNative() {
    printf("Hello from BSP!\n");
}
```

- Go to **Project** > **Build Project**.
- The build is successful.
- Flash the firmware:
 - Run > Run Configurations > STM32 C/C++ Application > application_debug > Run.



RUN THE EXAMPLE ON DEVICE

- Open the Termite serial terminal.
- Click the **Settings** button.
- Select the STM32F7508-DISCO board COM port.
- Reset the STM32F7508-DISCO board by pressing the **black** button near to the screen.
- The application starts and the **Hello World** and **Hello from BSP** messages are printed in the console!



Se MICROEJ



Foundation Library



DEFINITION

- A Foundation library is a Java library that depends on C code.
- Composed of:
 - A main project with the Java library source.
 - Abstraction Layer Interface or Low Level API (LLAPI) specified in C header files.
 - A **mockup** of the Java library for the simulator.





FOUNDATION LIBRARY EXAMPLE

- Import the GPIO Foundation Library Example:
 - Open menu File > Import... > General > Existing Projects into Workspace.
 - Select the archive file [training-package]/gpio_foundation_library_example-{version}.zip.
 - Select all the projects.
 - Click on Finish.
- If some projects don't compile click on Project > Clean... menu, select Clean all projects and click on Clean.





GPIO FOUNDATION LIBRARY

The **GPIO** class in the **gpio** project defines 2 native methods:

```
/**
  * GPIO management class.
  */
 public class Gpio {
 /**
  * Sets a value on the digital pin.
  *
  *
    @param pin
               the pin identifier
  *
    @param value
  *
  *
               digital pin value: true for high, false for low.
  */
 native public static void set(int pin, boolean value);
 /**
  * Gets the value of the digital pin.
  *
  *
    @param pin
               the pin identifier
  *
   @return true when the GPIO digital value is currently high, false otherwise.
  *
  */
 native public static boolean get(int pin);
 }
© MICROEJ 2024
```





Run the Foundation Library Example on Simulator



MOCKUP IMPLEMENTATION

• The **gpio-mockup** project is a JavaSE Project.

© MICROEJ 2024

The implementation of the **gpio** native methods is done in a class having the same package and same name:



• Each native method is implemented, without the **native** and with the **public** modifiers:

```
public class Gpio {
    private static final Map<Integer, Boolean> GPIO = new HashMap<Integer, Boolean>();

    public static void set(int pin, boolean state) {
        System.out.println("Set GPIO "+pin+" to "+(state?"on":"off"));
        GPIO.put(Integer.valueOf(pin), Boolean.valueOf(state));
    }
    public static boolean get(int pin) {
        // Returns false by default
        return GPIO.getOrDefault(Integer.valueOf(pin), Boolean.FALSE).booleanValue();
    }
}
```



MOCKUP DEPLOYMENT

- Build the Mockup with MMM:
 - Right-Click on the **gpio-mockup** project and select **Build Module**.
 - A .rip named gpio-mockup.rip is generated in the gpio-mockup\target~\artifacts folder.
- Add it to the VEE Port:
 - Unzip the **gpio-mockup.rip**
 - Drop the content of the folder content into the project [platform]-[Version]/source/

Warning: This folder is **overwritten** at each VEE Port build. To avoid that, add the mock module as a VEE Port dependency in the **-configuration/module.ivy**

Note: to ease the mock development phase, use the <u>Resolve Foundation Library in</u> <u>workspace</u> to retrieve mock sources in simulation \rightarrow the above steps can be avoided during the development in MICROEJ SDK.



RUN ON THE SIMULATOR

• The project **gpio-example** contains an example that uses the **gpio** library:

```
private static final int PIN = 0;
private static final long DELAY = 500;
public static void main(String[] args) throws InterruptedException {
    while (true) {
        Gpio.set(PIN, !Gpio.get(PIN));
        Thread.sleep(DELAY);
    }
```

• The **gpio** library has been added as dependency in the module.ivy of **gpio-example**:

```
<dependency org="com.microej.training.gpio" name="gpio" rev="1.1.0"/>
```

- Right click on the MicroEJ project **gpio-example.**
- Run as -> MicroEJ Application.





Run the Foundation Library Example on Device



RUN THE EXAMPLE ON DEVICE

- Build the **gpio-example** project for the device:
 - Go to Run -> Run Configurations.
 - Select the **gpio-example BlinkGpio** Run Configuration.
 - Go to Execution Tab.
 - Select Execute on Device.
 - Click **Run.**
- Compile, Link and Flash with the 3rd party IDE.





GET THE LINKER ERRORS

• The following errors show up during the link step of the BSP:

C:\XXX\com.microej.training.gpio.example.BlinkGpio\SOAR.o:(.text.soar+0x24dc): undefined reference to `Java_com_microej_training_gpio_Gpio_get'

C:\XXX\com.microej.training.gpio.example.BlinkGpio\SOAR.o:(.text.soar+0x24f0): undefined reference to `Java_com_microej_training_gpio_Gpio_set'

- The GPIO set() and get() methods are native methods. They must be implemented in the BSP.
- Add a simple implementation of the 2 methods:

© MICROEJ 2024

```
#include <stdio.h>
#include "sni.h"
jint Java_com_microej_training_gpio_Gpio_get(jint pin){
    printf("GPIO get status of pin: %d \n", pin);
    return 0;
}
void Java_com_microej_training_gpio_Gpio_set(jint pin, jboolean value){
    printf("GPIO set pin %d to %d\n", pin, value);
}
```



RUN THE EXAMPLE ON DEVICE

- Build the **gpio-example** project for the device:
 - Go to Run -> Run Configurations.
 - Select the **gpio-example BlinkGpio** Run Configuration.
 - Go to Execution Tab.
 - Select Execute on Device.
 - Click Run.
- Compile, Link and Flash with the 3rd party IDE.
- Open the Termite serial terminal to get execution traces.

🛞 Termite 3.4 (by CompuPhase)			_		×
COM7 115200 bps, 8N1, no handshake	<u>S</u> ettings	C <u>l</u> ear	<u>A</u> bou	ıt	<u>C</u> lose
MCU revision identifier: 0x1001					^
MCU device identifier: 0x449					
watchdog started					
MicroEJ START					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					
GPIO set pin 0 to 1					
GPIO get status of pin: 0					Υ.
					[+]



ABSTRACTION LAYER INTERFACE: LLAPI

• The LLAPI project defines the natives to be implemented in the BSP project:



ABSTRACTION LAYER IMPLEMENTATION: LLIMPL



- Add the **LLGPIO_impl.h** header file to the compiler path in the 3rd party IDE.
- Implement the LLGPIO_get(int32_t pin) and LLGPIO_set(int32_t pin, uint8_t state) functions in the BSP.

```
static uint8_t GPIO_initialized = 0;
static void LLGPIO_initialize(void)
{
    if(!GPIO_initialized)
    {
        GPIO_initialized = 1;
        _GPIOI_CLK_ENABLE();
        GPIO_InitTypeDef GPIO_InitStruct;
        /* Configure LED pin as output */
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
        GPIO_InitStruct.Pin = GPIO_PIN_1;
        HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
    }
}
```

```
void LLGPIO set(int32 t pin, uint8 t state)
    GPIO PinState value;
   LLGPIO initialize();
    if( state == JFALSE)
        value = GPIO PIN RESET;
    else
        value = GPIO PIN SET;
   HAL GPIO WritePin(GPIOI, GPIO PIN 1, value);
uint8 t LLGPIO get(int32 t pin)
   LLGPIO initialize();
   GPI0_PinState state = HAL_GPI0_ReadPin(GPI0I, GPI0_PIN_1);
   return (state == GPIO PIN RESET ? JFALSE : JTRUE);
```

Se MICROEJ

Se MICROEJ

Packaging and Tests



PACKAGING AND TESTS

BUILD A LIBRARY WITH MICROEJ MODULE MANAGER (MMM)

- Generate a JAR file with the classfiles.
- Generate a zip file with the sources.
- Generate the Javadoc.
- Execute the tests (defined in src/test/java folder).
- Publish the library in an MMM repository.



CONFIGURE THE TESTSUITE

- 1. Right-Click on the **source**/ folder of the VEE Port project.
- 2. Go to **Properties**.
- 3. Copy the location path.
- 4. Open the file **module.ivy** of the **gpio** project.
- 5. Uncomment the definition of the property **platform-loader.target.platform.dir.**
- 6. Paste the path previously copied.



LAUNCH MMM BUILD

- Right-Click on the **gpio** project and select **Build Module**.
- Build result is available in the folder **target~/artifacts**:

Build result is published in a local MMM repository:
 \.ivy2\repository\com\microej\training\gpio





TESTS RESULT

Testsuite report is available in the target~/artifacts/myfoundation-report-\${version}.zip file or in target~\test\html\test:

Tests	Failures		Errors		Ignore	ed	Tried Ag	ain			Success ra	e		Time	
	0	0	1	0			0		100.00%					6.751	_
	Assertions			Failu	ures		Success				5	uccess Rat	te		
			0			0		N	laN						_
te: failures are anticip	ated and checked for with	assertions w	hile errors are un	nanticipated.											
te: <i>ignored</i> tests are e	xecuted but not counted (on the succes	s rate.												
te: <i>tried again</i> tests ar	e executed but not counte	ed on the suc	cess rate.												
-															
skages															
ckages															
ckages	are not computed recursi	vely, they only	sum up all of its	testsuites nur	mbers.										
ckages ite: package statistics	are not computed recursi	vely, they only	sum up all of its	testsuites nur	mbers.				Tests	Errors	Failures Ign	ored Trie	d Time(s) Time Stam	p
ckages	are not computed recursi	vely, they only	r sum up all of its	s testsuites nu Name	mbers.				Tests	s Errors	Failures Ign	red Trie Agai	d Time(s) Time Stam	p
te: package statistics	are not computed recursi	vely, they only	r sum up all of its	s testsuites nu Name	mbers.				Tests	s Errors	Failures Igno	ored Trie Agai 0	d Time(s n 6.751) Time Starr 1550178103	1 P
ckages ote: package statistics m.microej.training.gpi -	are not computed recursi	vely, they only	r sum up all of its	s testsuites nu Name	mbers.				Tests 1	Errors	Failures Ign 0 0	ored Trie Agai 0	d Time(s n 6.751) Time Stam 1550178103	1 p
ckages ble: package statistics m.microej.training.gpi - - - - - - - -	are not computed recursi o.tests raining.gpio.tests	vely, they only	r sum up all of its	s testsuites nur Name	mbers.				Tests	s Errors	Failures Ign 0 0	ored Trie Agai 0	d Time(s n 6.751) Time Stam 1550178103	19
ckages He: package statistics 	are not computed recursi o.tests raining.gpio.tests	vely, they only	r sum up all of its	s testsuites nui Name	mbers.				Tests	Errors 0	Failures Ign	ored Trie Agai 0	d Time(s 6.751) Time Stam 1550178103	19
ckages te: package statistics 	are not computed recursi o.tests raining.gpio.tests	vely, they only	r sum up all of its	s testsuites nui Name	mbers.				Tests 1 Tests	s Errors 0 s Errors	Failures Ign 0 0 Failures Ign	ored Trie Agai 0 ored Trie Agai	d Time(s 6.751 d Time(s) Time Stam 1550178103) Time Stam	ір 19
ckages te: package statistics 	are not computed recursi o.tests raining.gpio.tests	vely, they only	r sum up all of its	s testsuites nui Name	mbers.				Tests 1 Tests	s Errors 0 s Errors 0	Failures Igno 0 0 Failures Igno 0 0	ored Trie Agai 0 ored Trie Agai	d Time(s n 6.751 d Time(s n 6.751) Time Stam 1550178103) Time Stam 1550178103	19 19
ckages te: package statistics 	are not computed recursi o.tests raining.gpio.tests	vely, they only	r sum up all of its	s testsuites nui Name	mbers.				Tests 1 Tests 1	s Errors 0 s Errors 0	Failures Igno 0 0 Failures Igno 0 0	rred Trie Agai 0 rred Trie Agai 0	d Time(s 6.751 d Time(s 6.751 6.751) Time Starr 1550178103) Time Starr 1550178103	19
ckages te: package statistics 	are not computed recursi	vely, they only	r sum up all of its	s testsuites nu. Name	mbers.				Tests 1 Tests 1	s Errors 0 s Errors 0	Failures Ign 0 0 Failures Ign 0 0	rred Trie Agai 0 ored Trie Agai 0	d Time(s 6.751 d Time(s 6.751) Time Starr 1550178103) Time Starr 1550178103	19 19
ckages te: package statistics 	are not computed recursi o.tests raining.gpio.tests estGpio	vely, they only	y sum up all of its	s testsuites nu Name	mbers.				Tests 1 Tests 1	s Errors 0 s Errors 0	Failures Ign 0 0 Failures Ign 0 0	rred Trie Agai 0 ored Trie Agai 0	d Time(s 6.751 d Time(s 6.751) Time Stam 1550178103) Time Stam 1550178103	19 19
ckages te: package statistics m.microej.training.gpi ckage com.microej.t lilTests_Testópio to top cCase_AllTests_T	are not computed recursi o.tests raining.gpio.tests estGpio	vely, they on!	y sum up all of its	s testsuites nu Name Name	mbers.				Tests 1 Tests 1	s Errors 0 Errors 0	Failures Ign 0 0 Failures Ign 0 0	ored Trie Agai 0 ored Trie Agai 0	d Time(s 6.751 d Time(s 6.751) Time Stam 1550178103) Time Stam 1550178103	19 19



V5.1 Aug. 2024 96

JAVADOC

Javadoc is available in **target~/javadoc** folder

OVERVIEW PACKAGE CLASS USE T	REE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES	NO FRAMES
SUMMARY: NESTED FIELD CONSTR ME	ETHOD DETAIL: FIELD CONSTR METHOD
com.microej.training.gpio	
Class Gpio	
java.lang.Object com.microej.training.gpio.Gpi	io
public class Gpio extends java.lang.Object	
GPIO management class.	
Constructor Summary	
Constructors	
Constructor and Description	
Gpio()	
Method Summary	
Method Summary All Methods Static Metho	ods Concrete Methods
Method Summary All Methods Static Methods Modifier and Type	In the second se
Method Summary All Methods Static Methods Modifier and Type static boolean	Concrete Methods Method and Description get(int pin) Gets the value of the digital pin.
Method Summary All Methods Static Meth Modifier and Type static boolean static void	Concrete Methods Method and Description get(int pin) Gets the value of the digital pin. set(int pin, boolean value) Sets a value on the digital pin.
Method Summary All Methods Static Meth Modifier and Type static boolean static void Methods inherited from the state of the st	Mode Method and Description get(int pin) get(int pin) Gets the value of the digital pin. set(int pin, boolean value) sets a value on the digital pin. sets a value on the digital pin. class java.lang.Object Sets a value on the digital pin.
Method Summary All Methods Static Meth Modifier and Type static boolean static void Methods inherited from clone, equals, getClass, h	Method and Description get(int pin) Gets the value of the digital pin. set(int pin, boolean value) Sets a value on the digital pin. class java.lang.Object hashCode, notify, notifyAll, toString, wait, wait
Method Summary All Methods Static Meth Modifier and Type static boolean static void Methods inherited from a clone, equals, getClass, h Constructor Detail	Method and Description get(int pin) Gets the value of the digital pin. set(int pin, boolean value) Sets a value on the digital pin. class java.lang.Object hashCode, notify, notifyAll, toString, wait, wait

© MICROEJ 2024



SNI

Manage Multithreading



GREEN THREAD ARCHITECTURE

- Green threads are threads that are scheduled by the virtual machine instead of natively by the underlying operating system.
- Green threads emulate multithreaded environments without relying on any native OS abilities, enabling them to work in environments that do not have native thread support.





THREAD SYNCHRONIZATION: BLOCKING CASE



- While a native method is executed, other Java threads can't be scheduled.
 - SNI functions stop the Java world.
- Usually, the actions are asynchronous on the BSP side and the result takes times to be returned (e.g., IP/USB/Bluetooth stacks).
- Goal: Execute a native in another task and wait for the result.

GPIO EXERCISE OVERVIEW

- The code of the **gpio-exercise** project does the following actions:
 - Wait for a button event and prints the index of the pressed button (User/Blue button)
 - Toggles the board LED1 each 500ms
 - Each action in performed in a dedicated thread

```
AICROEJ
```

```
public class GpioExercise {
```

```
private static final int PIN = 0;
private static final long DELAY = 500;
```

public static void main(String[] args) throws InterruptedException
{

```
// This thread waits for button actions.
      Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
      while (true) {
            System.out.println("Waiting for a button event...");
            int action = waitButton();
            System.out.println("Button pressed! Action ID=0x" +
            Integer.toHexString(action));
      });
      t.start();
      // The main thread loops indefinitely and blinks the LED.
      while (true) {
            Gpio.set(PIN, !Gpio.get(PIN));
            Thread.sleep(DELAY);
public static native int waitButton();
                                                    V5.1 Aug. 2024 100
```



Run the GPIO Exercise code

SETUP

Important note:

- This exercise uses the User button (blue button) of the STM32F7508 board to demonstrate how to implement a blocking Java native method without blocking the execution of other Java threads.
- The STM32F7508 platform already implements the Button events management in the MicroUI stack (see <u>button_manager.c</u>)
- It is recommended to use the MicroUI library to get button events in the application code (e.g. <u>MicroUI Input</u> <u>Example</u>)
- The next slides are showing how to run the exercise with STM32CubeIDE, it is also possible to use IAR.
- A C implementation is provided in the **LLGPIO_STM32F7508-{version}.zip** package.
- Add **LLGPIO_exercise.c** to the BSP project:
 - Copy / Paste LLGPIO_exercise.c in the stm32f7508_freertos-bsp\projects\microej\gpio\src folder.
 - Remove the previous **LLGPIO.c** implementation.
 - **LLGPIO_exercise.c** redefines an interrupt handler defined in the **ui** folder. The folder needs to be excluded from BSP build to run this sample.
 - Exclude the **ui** folder from the BSP build:
 - In STM32CubeIDE, right-click on the **ui** folder
 - Click on **Properties**
 - Click on Exclude resource from build

- 👻 🥔 gpio
 - 🕆 🗁 inc
 - ILLGPIO_impl.h
 - 🗙 🗁 src
 - > LLGPIO_exercise.c

RUN THE EXERCISE CODE (1/2)

- Compile the application in MICROEJ SDK:
 - Right click on the **GpioExercise.java** class of the **gpio-exercise** project
 - Run as -> Run Configurations..
 - Double click on MICROEJ Application
 - Go to **Execution** tab
 - Select the **STM32F7508** platform
 - Select Execute on Device
 - Click **Run**
- In STM32CubeIDE, click on your project once to select it.
 - Go to **Project** > **Build Project**.
 - Wait for the end of the build.
- Plug the STM32F7508-DK board to the PC
- In STM32CubeIDE, select Run > Run Configurations > STM32 C/C++ Application > application_debug > Run.

😂 MICROEJ

RUN THE EXERCISE CODE (2/2)

- Open the Termite serial terminal.
- Reset the STM32F7508-DK board by pressing the **black** button near to the screen.
- The application starts and waits for a button event.
- LED1 is not blinking each 500ms as expected.
 The waitButton() native blocks the execution of the other Java threads.
- When pressing the button once:
 - The ID of the button event is printed in the console
 - The LED turns on
- When pressing again, the ID of the button event is printed and the LED turns off



COM13 115200 bps, 8N1, no handshake	Settings Clear	About Close
Start		
4CU revision identifier: 0x1001		
ACU device identifier: 0x449		
vatchdog started		
MicroEJ START		
Vaiting for a button event		
5		

Traces when the application starts





GPIO EXERCISE: BLOCKING BEHAVIOR



- In this example, the execution of the **waitButton()** native method will block until the button is pressed.
 - In other words, while Java_com_microej_training_gpio_example_GpioExercise_waitButton() has not returned, no other Java thread can be scheduled.
 - This is because the native function is called in the same RTOS/OS task as the Java application.
- This schematic explains what is going on:



Se MICROEJ

Hand's On

Implement a blocking Java native method without blocking the execution of other Java threads.

HAND'S ON DIRECTIVES

- Only the C code should be updated
- Here is a summary of what should be done in C:
 - Signal the MicroEJ Core Engine to suspend the current thread when the native function returns.
 - Remove the blocking operations from the native function so that it returns immediately.
 - Implement a callback function that returns the index of the pressed button.
 - Register this callback function in the MicroEJ Core Engine to call it when the Java thread is resumed.
 - Resume the Java thread when a button is pressed.
- Tips:
 - Use the SNI functions defined in **sni.h**
 - SNI documentation: <u>https://docs.microej.com/en/latest/PlatformDeveloperGuide/sni.html#sni</u>





THREAD SYNCHRONIZATION: CALLBACK PATTERN


















STEP 1: UPDATE THE C NATIVE FUNCTION



- The Java_com_microej_training_gpio_example_GpioExercise_waitButton() function will now suspend the current Java thread.
 It will also store the information required to resume it and register the callback function.
- The function **SNI_suspendCurrentJavaThreadWithCallback()** returns immediately. The current thread is actually suspended when the native function returns.
- The value returned by the Java_com_microej_training_gpio_example_GpioExercise_waitButton() doesn't matter anymore. The callback function will be in charge of returning the value.

```
jint Java_com_microej_training_gpio_example_GpioExercise_waitButton()
{
    // Initialize the GPIOs
    LLGPIO_initialize();
    java_thread_id = SNI_getCurrentJavaThreadID();
    SNI_suspendCurrentJavaThreadWithCallback(0, (SNI_callback)waitButton_callback, NULL);
    return SNI_IGNORED_RETURNED_VALUE; // Returned value not used
}
```

static int32 t java thread id;

STEP 2: UPDATE THE BUTTON INTERRUPT FUNCTION & MICROEJ

• The role of the button interrupt is now to resume the Java thread when a button event occurs. Update it this way:

STEP 3: IMPLEMENT THE CALLBACK FUNCTION



- The callback function must have the same signature as the SNI native (same parameters and return type): jint waitButton_callback()
- The callback function is automatically called by the Java thread when it is resumed.
- Use the SNI_getCallbackArgs() function to retrieve the arguments that was previously given to the SNI_suspendCurrentJavaThreadWithCallback() or SNI_resumeJavaThreadWithArg() functions.

```
static jint waitButton_callback(){
    int32_t *button_index_addr;
    SNI_getCallbackArgs(NULL, (void**)&button_index_addr);
    return (jint)*button_index_addr; // Actual value returned to Java
}
```

RUN THE UPDATED CODE

- Open the Termite serial terminal.
- Reset the STM32F7508-DK board by pressing the **black** button near to the screen.
- The application starts and waits for a button event.
- LED1 is now blinking each 500ms.
- When pressing the button once:
 - The ID of the button event is printed in the console
- When pressing again, the ID of the button event is printed and the LED turns off



Settinas

Traces when the application starts

COM13 115200 bps, 8N1, no handshake

MCU revision identifier: 0x1001

Waiting for a button event...

Start







Х

←

Close

Clear

About



Resources



ONLINE RESOURCES

- <u>https://developer.microej.com/</u>
 - Examples, platforms, libraries, user guides, application notes...
 - Javadocs (Java API)
 - Addon tools
- <u>https://docs.microej.com</u>
- <u>https://github.com/MICROEJ/</u>
 - Source code repository
- <u>https://forum.microej.com/</u>
- <u>https://repository.microej.com/</u>
 - MICROEJ Central Repository (modules repository)



MAIN RESOURCES

- <u>https://docs.microej.com/en/latest/ApplicationDeveloperGuide/index.html</u> : Describes MICROEJ usage for end developers
- <u>https://docs.microej.com/en/latest/PlatformDeveloperGuide/index.html</u>: Describes how to interact with the platform and integrate MICROEJ to a board
- <u>https://github.com/MICROEJ/Example-Standalone-Foundation-Libraries</u>: Snippets of code for foundation libraries (EDC, BON, Net, MicroUI...)
- <u>https://github.com/MICROEJ/ExampleJava-Widget</u>: Source code for using the widget library



Shortcuts

MICROEJ SDK / Studio



- CTRL + Space
 - Auto completion
 - Probably the most useful one
- CTRL + D
 - Delete row
- ALT + Up/Down Arrow
 - Move the row (or the entire selection) up or down. Very useful when rearranging code
- CTRL+SHIFT+O
 - Organize imports.



Shortcuts

- CTRL+SHIFT+T
 - Open Type.

🔷 Open Type —		×
Enter type name prefix or pattern (*, ?, or camel case):		•
Matching items:		
		_
ОК	Cancel	I



- CTRL+SHIFT+R
 - Open Resource (any file)

් Open Resource	—		×
Enter resource name prefix, path prefix or pattern (?, * or came	case):		•
<u>M</u> atching items:			
② Sho <u>w</u> In ▼ Open Wit <u>h</u> ▼ <u>O</u> pen		Cance	el



- CTRL + O
 - Open Outline (find method or field)
 - Press CTRL + O again to show methods from superclasses





- F2
 - Display the Javadoc
- Hold CTRL + Click on class
 - Go through the definition of class
- CTRL + T
 - On a method: display implementations of the method in subclasses or definitions in superclasses
 - On a class: display class hierarchy (superclasses and subclasses)
- CRTL + 1
 - Extract variable to
 - Local variable

© MICROEJ 2024

Constant



- CTRL + I
 - Correct indentation
- ALT + Shift + R on a class / method / field
 - Rename
- CTRL + F
 - Search in file
- CTRL + H
 - Search plugin of Eclipse























for your attention !



