

# MSP-EXP432P401R\_OS3

**Licensing is required when using any Micrium software, regardless of the state of the software (Library or Full Source). This project is only meant for example purposes. For projects using Library versions of the software, please contact our Sales office to obtain the Full Source version at +1 (954) 217-2036.**

## MSP-EXP432P401R Example Project Read-Me

The provided example project for which this Read-Me was made utilizes the TI MSP-EXP432P401R (MSP432P401R) evaluation board from the MSP432 Family. The MCU found on this development board conforms with the ARM\_Cortex\_M4 architecture.

- [Project Download](#)
- [Toolchain IDE Versions](#)
- [Micrium Product Versions](#)
- [Hardware Setup](#)
- [Loading & Running The Project on the Board](#)
  - [IAR Embedded Workbench™](#)
  - [Code Composer Studio 7™](#)
- [μC/OS-III](#)
- [μC/Probe](#)
  - [Running with J-Link](#)
  - [Running with CMSIS-DAP](#)

## Project Download

Download Link	<a href="#">Micrium_MSP-EXP432P401R_OS3.zip</a>
---------------	---

## Toolchain IDE Versions

IDE/Toolchain	Version
IAR EW for ARM	7.80.3
Code Composer Studio	7.0.0

## Micrium Product Versions

Product	Version
μC/CPU	1.31.01
μC/LIB	1.38.02
μC/OS-III	3.06.01

## Hardware Setup

1. Have the board connected via the **J-LINK OR Built-in TI XDS 110** into the board debugging input (**10-Pin JTAG OR Built-in XDS via the MicroUSB connector**, respectively).
2. If Rev 1.0 of the board is used (black PCB), then the JTAG switch (**S101**) must be set to the desired debug interface. If using J-LINK, insert the debugger into the JTAG header (**J102 on Rev 1.0 OR J8 on Rev 2.1**).
3. Use MicroUSB connector for power.

## Loading & Running The Project on the Board

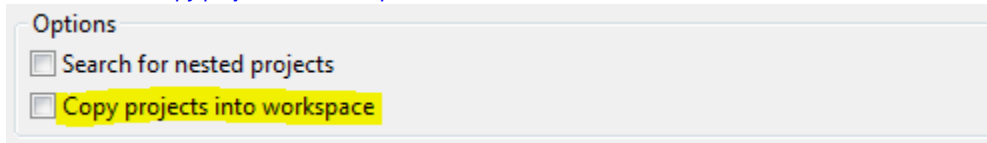
**Make sure to open the example project workspace using the mentioned IDE(s) version or newer.**

### IAR Embedded Workbench™

1. Click on [File->Open->Workspace...](#)
2. Navigate to the directory where the workspace is located: `$\Micrium\Examples\TI\MSP-EXP432P401R\IOS3\IAR\IOS3.eww`
3. Click [Open](#).
4. For safety, clean the project by clicking on [Project->Clean](#) (if available).
5. Compile the project by clicking on [Project->Make](#).
6. Make sure your hardware setup (as previously described) is correct.
7. Download the code to the board by clicking on [Project->Download and Debug](#).
8. Run the project by clicking on [Debug->Go](#). To stop the project from running, click on [Debug->Stop Debugging](#).

### Code Composer Studio 7™

1. Click on [File->Import...](#)
2. Select [Existing Projects into Workspace](#).
3. Navigate to the directory where the workspace is located: `$\Micrium\Examples\TI\MSP-EXP432P401R\IOS3\CCS`
4. Click [OK](#).
5. Make sure the "[Copy projects into workspace](#)" check-box is unchecked.



6. Make sure that the project has been selected under the [Projects](#) check-box.
7. Click [Finish](#).
8. For safety, clean the project by clicking on [Project->Clean](#) (if available).
9. Compile the project by clicking on [Project->Build All](#). The project should build successfully.
10. Make sure your hardware setup (as previously described) is correct.
11. Download the code to the board by right-clicking inside the project directory and selecting [Debug As->Code Composer Debug Session](#).
  - a. Select the appropriate interface inside the [Debugger Tab](#) (if needed).
12. Run the project by clicking on [Run->Resume](#). To stop the project from running click on [Run->Terminate](#).

## μC/OS-III

```
void main (void)
{
    ...
    OSInit(&os_err);                /* Initialize uC/OS-III
*/      (1)

    ...
    OSTaskCreate(&AppTaskStartTCB,    /* Create the start task
*/      (2)
                "App Task Start",
                AppTaskStart,
                0,
                APP_CFG_TASK_START_PRIO,
                &AppTaskStartStk[0],
                APP_CFG_TASK_START_STK_SIZE / 10u,
                APP_CFG_TASK_START_STK_SIZE,
                0u,
                0u,
                0,
                (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                &os_err);

    OSStart(&os_err);                /* Start multitasking
*/      (3)
}

static void AppTaskStart (void *p_arg)
(4)
{
    ....

    while (DEF_TRUE) {                /* Task body, always as
an infinite loop.          */      (5)
        ...
    (6)

        OSTimeDlyHMSM( 0u, 0u, 0u, App_LED1_Dly_ms,
    (7)
                    OS_OPT_TIME_HMSM_STRICT,
                    &os_err);
    }
}

void App_Port1_ISR (void)
{
    ...

    startup_led = BSP_GPIO_LED2_RED;    /* Select red component
of LED2 as first LED to be lit. */
```

```

    if (sw2 == DEF_ON) {
(8)    if (RGB_Ctr < startup_led) {
        RGB_Ctr = startup_led;
    }
        BSP_LED_Toggle(RGB_Ctr);
    RGB_Ctr++;
        RGB_Ctr %= 5u;
    }
    if (sw1 == DEF_ON) {
        App_LED1_Dly_ms += 200u;                /* Increment
AppTaskStart's millisecond delay by 200.    */
        if (App_LED1_Dly_ms > 1000u) {          /* Reset delay if it
reaches 1000.                                */
            App_LED1_Dly_ms = 100u;
        }
    }
}

```

```

        BSP_GPIO_REG_IV(BSP_GPIO_P1_BASE_ADDR);      /* Clear interrupt flag.
Interrupt already serviced.      */
    }

```

Listing - app.c

(1)

OSInit() initializes uC/OS-III and must be called prior to calling OSStart(), which actually starts multitasking.

(2)

OSTaskCreate() creates a task to be managed by uC/OS-III. Tasks can be created either prior to the start of multitasking or by a running task. In this case, the task "AppStartTask" gets created.

(3)

OSStart() starts multitasking under uC/OS-III. This function is typically called from the startup code but after calling OSInit().

(4)

AppTaskStart is the startup task created in (2).

(5)

A task must be written as an infinite loop and must not return.

(6)

In most examples, there is hardware dependent code such as LED blink, etc.

(7)

OSTimeDlyHMSM() allows AppTaskStart to delay itself for a user-specified amount of time. Rescheduling always occurs when at least one of the parameters is nonzero. Placing a break-point here can ensure that uC/OS-III is running, it should get hit periodically every (App\_LED1\_Dly\_ms) milliseconds.

(8)

In this example, the global variable App\_LED1\_Dly\_ms gets initialized to 100 ms but is incremented by 200 inside an ISR (App\_Port1\_ISR) triggered by pressing Button S1 on the board. This causes AppTaskStart to delay at the resulting rates of 100, 300, 500, 700, and 900 ms. Pressing Button S2 on the board will cause LED2 to toggle through its RGB components.

For more information please refer to [uC/OS-III Users' Guide](#).

## μC/Probe

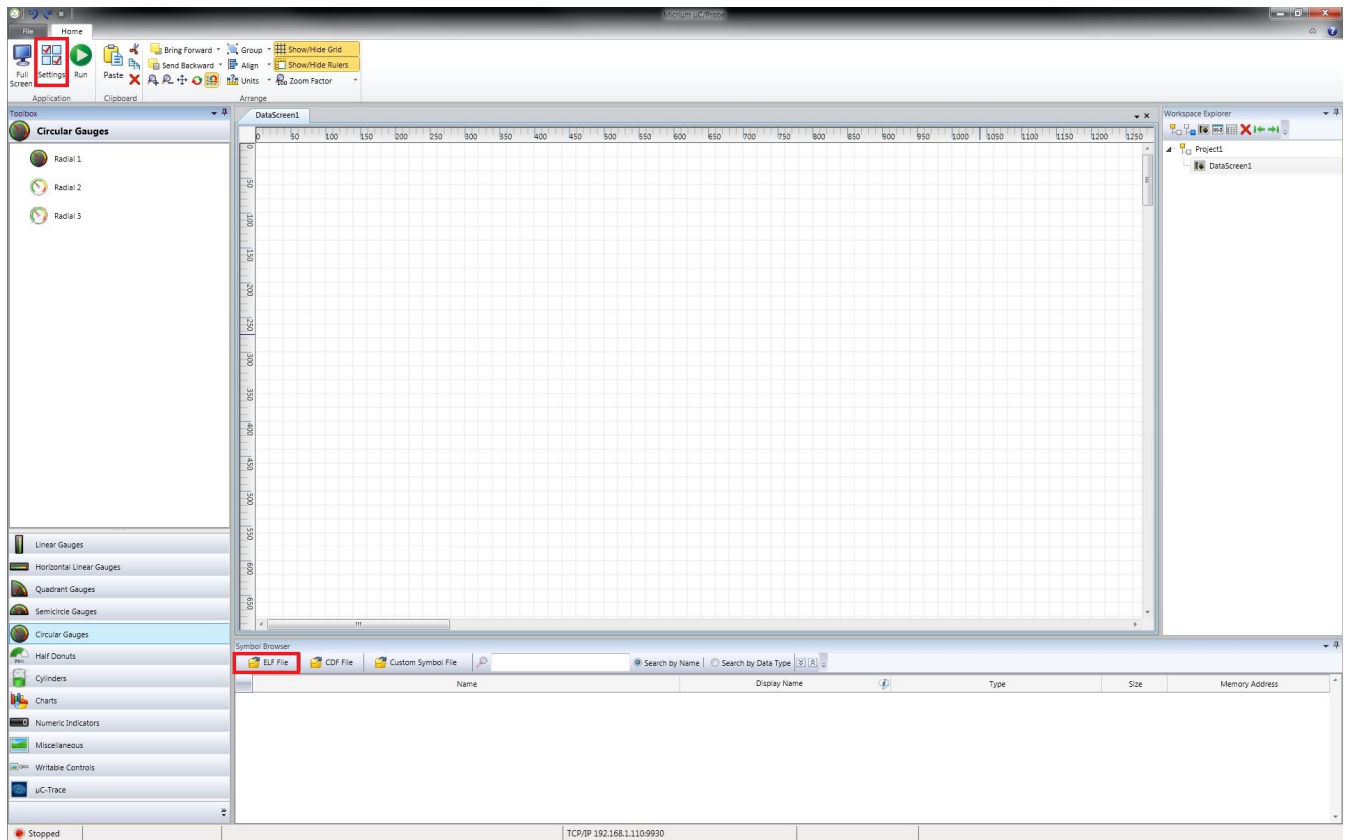
μC/Probe, is a Micrium Windows™ application to graphically view the internals of any embedded system. This example project includes a pre-configured μC/Probe workspace that can be found at:

*\$Micrium\Examples\T1MSP-EXP432P401R\OS3\OS3.wsp*

Please compile the project (as described earlier in this document) prior to opening a pre-configured μC/Probe workspace.

In order for μC/Probe to display symbols, an **ELF file** that is generated by the compiler is required. After the example project has compiled, look for the ELF file that is usually found inside the compiler auto-generated binaries folder.

The following image shows where the **ELF file** (highlighted in **RED**) button is found to search for the project's ELF file.



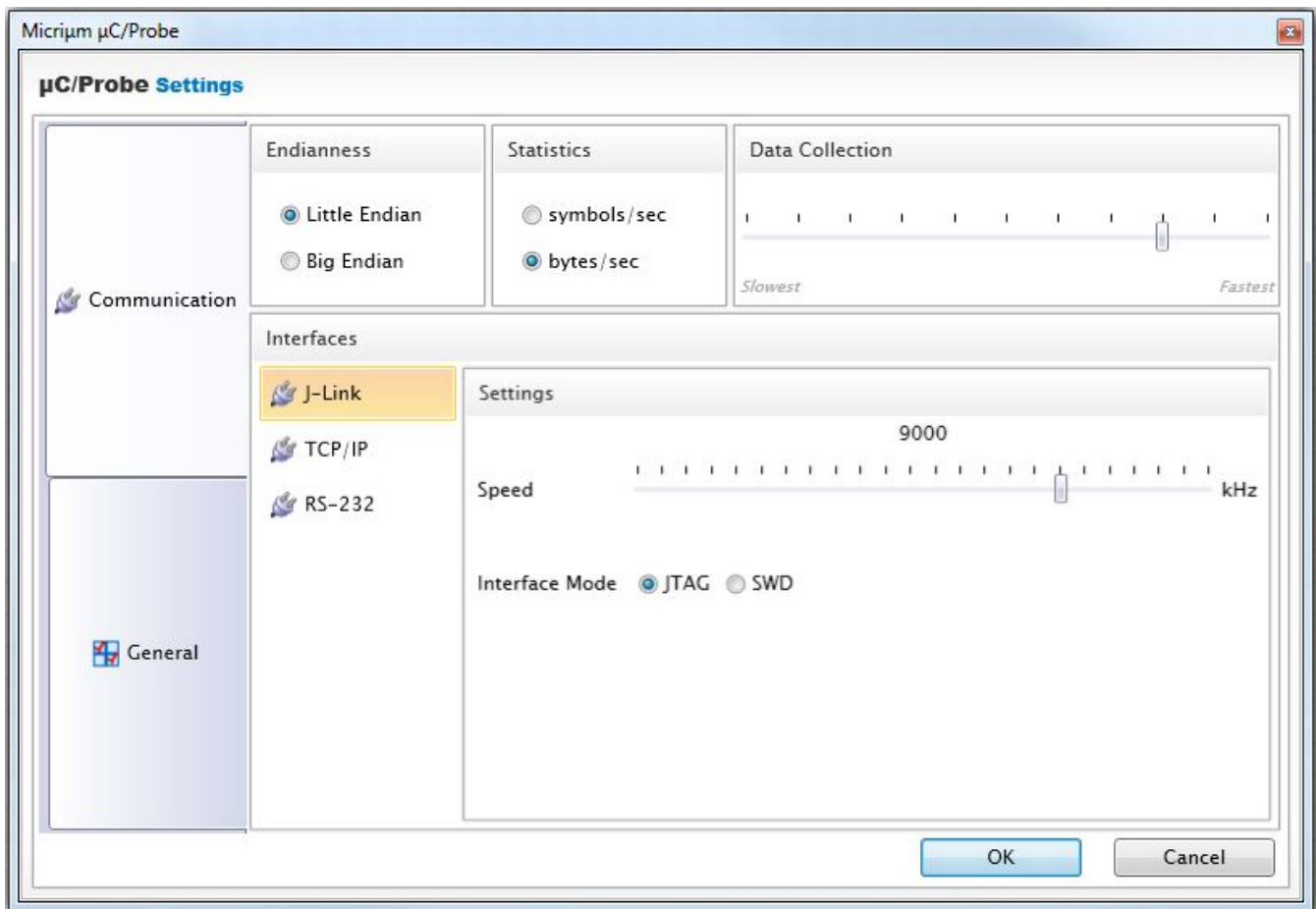
If creating a new  $\mu$ C/Probe workspace, you must configure  $\mu$ C/Probe with the proper communication protocol used in your project. The following communication protocols are currently available for this example project:

## Running with J-Link

When running a Micrium example project that is using the J-Link debugger to interface with  $\mu$ C/Probe, there is no additional set-up necessary other than to configure  $\mu$ C/Probe's settings to "J-Link".

In  $\mu$ C/Probe's settings, under the [Communication](#) tab, select [J-Link](#) under the [Interfaces](#) section and configure the [Speed](#) and [Interface Mode](#) you desire that suits your project's needs. Along with the J-Link settings,  $\mu$ C/Probe also allows you to change the endianness of the device, how to receive statistics, and the rate at which the data collection is done.

The following image illustrates how the settings should look:



## Running with CMSIS-DAP

When running an example project that interfaces with µC/Probe using a CMSIS-DAP connection, there is no additional set-up necessary other than to configure µC/Probe's settings to "CMSIS-DAP".

In µC/Probe's settings, under the [Communication](#) tab, select [CMSIS-DAP](#) under the [Debug Interfaces](#) section and configure the [JTAG/SW Adapter](#), [Port](#), and [Max Clock](#) for your project; if desired, [SWJ](#) can be enabled by checking the check-box. Along with the CMSIS-DAP settings, µC/Probe also allows you to change the endianness of the device, how to receive statistics, and the rate at which the data collection is done.

CMSIS-DAP should run out-of-the-box without the need of target code.

The following image illustrates how the settings should look:

## µC/Probe Settings

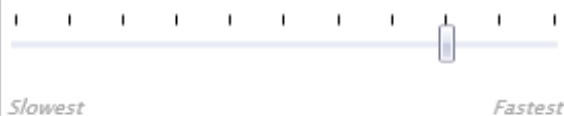
## Endianness

☒ Little Endian☐ Big Endian

## Statistics

☐ symbols/sec☒ bytes/sec

## Data Collection



## Communication

## Interfaces

Refresh

## None

Cloud Only

## Target Resident Code

TCP/IP

RS-232

USB

## Debug Interfaces

J-Link

CMSIS-DAP

Cypress PSoC Prog

## Settings

JTAG/SW Adapter

Serial No A000000001

Firmware Version 1.0

SWJ ☐

Port

Max Clock

## General

OK

Cancel