# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

## Module 4

## KV3: Working with Constrained Resources and the Bare Metal Profile

As we’ve covered before, embedded environments have limited resources, or constrain those resources in order to complete their functions. Developing an application with the full Mbed RTOS also has resource implications. Remember that 4KB stack allocated per thread? For simple programs, or in resource-constrained situations, a clear alternative is to adopt the bare metal profile.

Let’s define this profile, and how it saves on resources.

The bare metal profile is a subset of the Mbed OS, which minimizes the size of the final application. It is intended for constrained hardware, and results in efficient and compact code. It is important to remember, however, that it is not itself an RTOS.

The bare metal profile only uses the set of APIs that applications require. That includes driver, platform and this example subset of the RTOS APIs.

|  |  |
| --- | --- |
| **API​** | **Available in Bare Metal​** |
| Thread ​ | X​ |
| ThisThread ​ | √​ |
| Semaphore ​ | √​ |
| Queue ​ | X​ |
| Mutex ​ | √​ |
| Mail​ | X​ |
| Kernel interface functions ​ | **Get\_ms\_count** only​ |
| IdleLoop ​ | X​ |
| EventFlags​ | √​ |
| ConditionVariable​ | X​ |

Doing this gives better control of the application's final size than the full profile, which relies on the build time linker to remove classes that are not used, and are not a dependency.​

It is very important to know the functionality of these APIs when using them, so familiarize yourself with them as much as you can.

|  |  |
| --- | --- |
| **API​** | **Available in Bare Metal​** |
| AnalogIn​ | √​ |
| AnalogOut​ | √​ |
| BusIn​ | √​ |
| BusOut​ | √​ |
| DigitalIn​ | √​ |
| DigitalOut​ | √​ |
| InterruptIn​ | √​ |
| BufferedSlave​ | √​ |
| SPI​ | √​ |
| SPISlave​ | √​ |

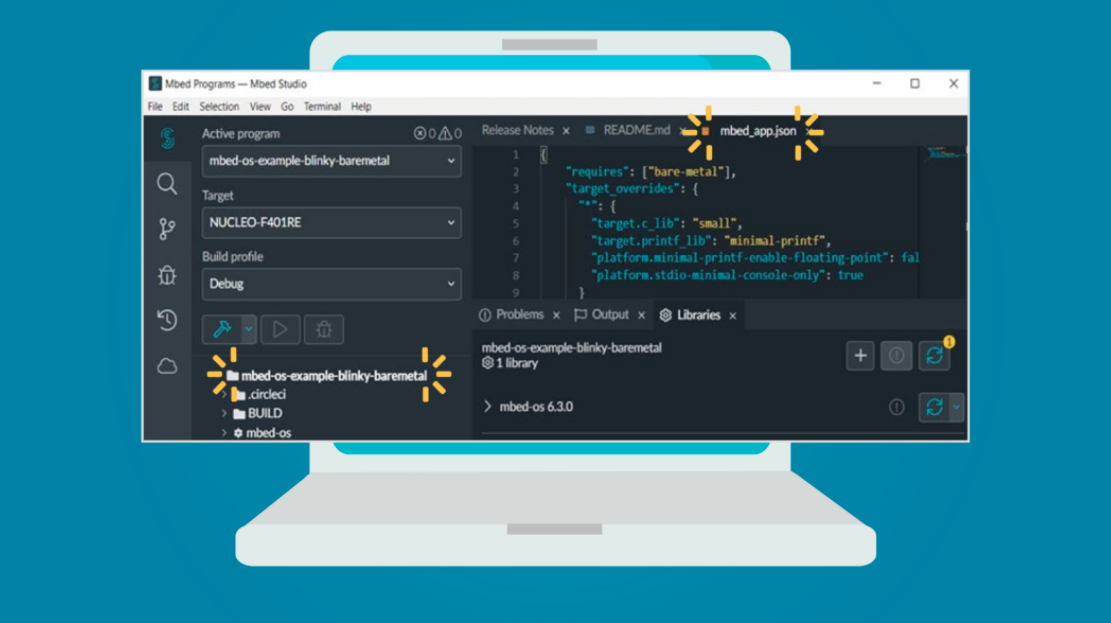
As you can see, most of the regular RTOS APIs are not available, while drivers generally are. You can manually add missing APIs if your application requires them, but make sure they are compatible.

If further minimizing of the application size is needed, the bare metal profile can use the small C libraries, however these are not thread safe.

Remember the scheduling strategies used by an RTOS? The bare metal profile does not require this, so instead, activities are polled or interrupt-driven. This simplifies application code and allows using APIs that are not thread safe. ​

Now, let’s use the bare metal profile. Firstly, a project must have its mbed\_app.json file correctly configured for this implementation.​

The simplest way to launch a bare metal project is to select an available bare metal template, for example in Mbed Studio, as shown here.



In this example, the mbed\_app.json file has already been configured.

Alternatively, you can modify the mbed\_app.json file manually, then develop your program, applying the constraints already identified. ​

Now that you know the basics of working in a resource-constrained environment, the next knowledge video will move on to implementing and evaluating an example of the Mbed RTOS.

**Visit arm.com for more information.**