**What is an IDE and compiler? List out differences.**

An IDE provides an environment to create, build and test a software application. It consists of a code editor, compiler or [interpreter](http://pediaa.com/difference-between-compiler-interpreter-and-assembler/#Interpreter), and a debugger with a [Graphical User Interface (GUI)](http://pediaa.com/difference-between-gui-and-cli/#GUI).

Also, it has features such as source code formatting, error diagnostics, and intelligent code completion, reporting. On the other hand, a compiler is a special program that converts the source code to executable machine code.

The programmer can run the appropriate language compiler according to the programming language used to write the source code.

The **main difference** between IDE and compiler is that the **IDE is a software suite that consists of tools required to develop and test software applications while the compiler is a program that translates the source code written in a high-level**[programming language](http://pediaa.com/what-is-the-difference-between-markup-language-and-programming-language/#Programming%20Language)**into a low-level machine code.**

**What is a bootloader and how does it work?**

Data of an [operating system](https://www.ionos.com/digitalguide/server/know-how/what-is-an-operating-system/) must be loaded into the [working memory](https://www.ionos.com/digitalguide/server/know-how/what-is-ram/) during device start-up. This is made possible by a so-called bootloader, also known as a boot program or bootstrap loader. For this purpose, immediately after a device starts, a bootloader is generally launched by a **bootable medium** like a hard drive, a CD/DVD, or a USB stick. The boot medium receives information from the computer’s **firmware** (e.g. BIOS) about where the bootloader is. The whole process is also described as “booting”.

**How does a bootloader work?**

When you press the start button on a computer, the very first thing you see on the screen is information about the hardware installed. The software responsible for this notification is the device firmware mentioned above, which is usually implemented by manufacturers in **flash memory** on the computer’s **motherboard**.

With most desktop PCs and notebooks this will be the [BIOS](https://www.ionos.com/digitalguide/server/configuration/how-to-enter-bios/) (**B**asic **I**nput/**O**utput **S**ystem) or the more modern [UEFI](https://www.ionos.com/digitalguide/server/know-how/uefi-unified-extensible-firmware-interface/) (**U**nified **E**xtensible **F**irmware **I**nterface). Both applications collect the most diverse hardware data and create a complete list of all of the device’s available drives.

When this process is complete, the firmware goes through the data carriers found in sequence, checking for a bootloader by means of a special signature the **so-called boot signature** (or “boot record”). The search always starts on the **removable media** (CD/DVD, USB stick, external hard drive, etc.), followed by the **hard-coded drives**. With the latter, the bootloader and its signature is generally in the [Master Boot Record](https://www.ionos.com/digitalguide/server/configuration/what-is-mbr/) (MBR), which also contains the data carrier’s partition tables. When a bootloader is found, it is loaded and the system start is initiated. If the search is unsuccessful, the firmware will return an error message.

**What do you understand by OTA update?**

An over-the-air update is the wireless delivery of new software or data to mobile phones and tablets.

Wireless carriers used over-the-air updates to deploy firmware and configure phones for use on their networks.

The initialization of a newly purchased phone, for example, requires an over-the-air update. With the rise of smartphones and tablets, carriers and manufacturers have also turned to over-the-air updates for deploying new operating system to these devices to keep the devices up to date.

**List the difference between baremetal vs RTOS programming?**

Baremetal programming is programming without an operating system to provide system level services.

RTOS is a real time operating system, on an mcu it is not.

**RTOS** pre-emption is easy to achieve in RTOS and switching between task will take a lot less time than expected.

RTOS can be a lot less time consuming to work with if you are familiar with it. But if we are not used to working with RTOS it can produce behavioral bugs that will take ages to be fixed.

Code reusability is not guaranteed but it can be achieved by firmware development with a little more effort.

Middle ware is available in RTOS naturally such as file system and USB and can be integrated easily. RTOS uses a lot more memory and code space

**Bare-metal** Pre-emption is possible in bare-metal also by using interrupts and prioritizing the interrupt but the time of switching between interrupts will be a lot more than the RTOS.

Bare-metal is time consuming but it is easy to work with as it cannot misbehave and all the work starting from registers to end user peripherals is done by the developer.

Code reusability is not possible and it does not comes naturally but it can be achieved with extra effort. Memory and code space used by bare metal is only of application as no OS is present.

**How to choose between baremetal and RTOS for project?**

Performance, features, cost, ecosystem, middleware, vendor and the engineering team are just examples of characteristics that should be evaluated when a development team is ready to select an RTOS for their product.

**Characteristic #1 – Performance**

RTOS performance is a critical factor to consider when selecting an RTOS. All RTOSes are NOT created equal and an attempt to save a few dollars can cost orders of magnitude more. When it comes to performance, developers have a variety of factors that need to be considered. First, memory requirements such as ROM, flash and RAM footprints need to be considered. RTOSes are powerful and with that power comes additional code and data needs. Second, processing speed such as interrupt latency and context switch times should be reviewed. A high quality RTOS will document these parameters for a variety of architectures and clock speeds.

**Characteristic #2 – Features**

Every RTOS doesn’t have the exact same features or the features implemented in the most optimal manner. Developers need to evaluate which features are the most critical to the systems success and select an RTOS that has those features. Developers may want to consider scalability, safety certifications or even the efficiency of memory protection schemes. A little thought about feature is whether the RTOS conforms to standard interfaces such as POSIX. Even the way in which RTOS tasks and objects are allocated could be an important feature. Many RTOSes use dynamic memory allocation for tasks which for resource constrained embedded system could be dangerous due to heap fragmentation and other issues. An RTOS that can statically allocate tasks might be the better choice.

**Characteristic #3 – Cost**

Undoubtedly one of the largest, if not only thought about RTOS characteristic is cost. Despite the huge efforts required in labor to develop robust software, no one wants to pay for it! Developers need to get over it and probably evaluate what an RTOS may really cost. A few considerations for commercial RTOSes are the upfront costs for licensing and any recurring licenses such as royalties. In additional to these obvious costs, developers need to also consider the total cost of ownership for the RTOS. That is, the cost to learn, setup, integrate and debug the selected operating system. Total cost of ownership for an open source RTOS can potentially exceed that of a commercially purchased RTOS due to lack of support, poor code quality and so forth.

**Characteristic #4 – Ecosystem**

Having the best performance, features and cost doesn’t mean a thing if there isn’t a large and vibrant community to support the RTOS. A software products ecosystem is a critical piece of the selection process in order to ensure ease of integration, support and product lifetime. When developers investigate an RTOS ecosystem, they should determine whether the RTOS is supported and adopted by their industry and the embedded software industry as a whole. They should determine whether there is support for a variety of architectures and processors or whether the RTOS is just a one trick pony. The availability of numerous examples and ports is also an important indicator that the RTOS is supported and has a strong community of users around it.

**Characteristic #5 – Middleware**

Many RTOSes come with middleware components or have third parties who have developed components that integrate into the RTOS. Developers should evaluate their RTOSes middleware and determine what the integration effort might be. Sometimes the integration is seamless while other times it is an obvious nightmare. Some RTOSes lack support for middleware and open source components need to be integrated which can lead to a variety of time consuming integration issues. Verify that the middleware has common components such as USB, TCP/IP, file systems and graphics generators before jumping in with both feet.

**Characteristic #6 – Vendor**

Take a good hard look at the vendor who developed, maintains and distributes the RTOS. Examine their source code and documentation. A good supplier with have meticulous documentation that answers many of the questions that would arise while integrating the RTOS into the system. No matter how good documentation gets, it will never be perfect. Testing how fast the vendor is to respond to question and support issues could be critical and save precious time and money getting the product out the door. Don’t just blindly trust. Be an engineer and put the vendor to the test and see if they squirm or roll up their sleeves.

**Characteristic #7 – Engineering Team**

The characteristic of RTOS selection that is probably the most common to overlook is the engineering team. The RTOS that is selected should minimize the labor intensity for the team and allow them to focus on product differentiators rather than increase it as they learn how to integrate and setup an RTOS. As much as we like to grow professionally as engineers, we should attempt to select an RTOS we are familiar with and can work most efficiently with. Sometimes development doesn’t work out that way but we should at least try