**The safety system in the C# Job System**

**Race conditions**

When writing [multithreaded code](https://docs.unity3d.com/Manual/JobSystemMultithreading.html), there is always a risk for [race conditions](https://en.wikipedia.org/wiki/Race_condition). A race condition occurs when the output of one operation depends on the timing of another process outside of its control.

A race condition is not always a bug, but it is a source of indeterministic behavior. When a race condition does cause a bug, it can be hard to find the source of the problem because it depends on timing, so you can only recreate the issue on rare occasions. Debugging it can cause the problem to disappear, because breakpoints and logging can change the timing of individual threads. Race conditions produce the most significant challenge in writing multithreaded code.

**Safety system**

To make it easier to write multithreaded code, the Unity C# Job System detects all potential race conditions and protects you from the bugs they can cause.

For example: if the C# Job System sends a [reference](https://docs.microsoft.com/en-us/cpp/cpp/references-cpp) to data from your code in the main thread to a job, it cannot verify whether the main thread is reading the data at the same time the job is writing to it. This scenario creates a race condition.

The C# Job System solves this by sending each job a copy of the data it needs to operate on, rather than a reference to the data in the main thread. This copy isolates the data, which eliminates the race condition.

The way the C# Job System copies data means that a job can only access [blittable data types](https://en.wikipedia.org/wiki/Blittable_types). These types do not need conversion when passed between [managed](https://en.wikipedia.org/wiki/Managed_code) and native code.

The C# Job System can copy blittable types with [memcpy](http://www.cplusplus.com/reference/cstring/memcpy/) and transfer the data between the managed and native parts of Unity. It uses memcpy to put data into native memory when scheduling jobs and gives the managed side access to that copy when executing jobs. For more information, see [Scheduling jobs](https://docs.unity3d.com/Manual/JobSystemSchedulingJobs.html).