Username: Pralay Patoria **Book:** Under the Hood of .NET Memory Management. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Conclusion

The world of software advances at an amazing pace. Fortunately for us, the .NET framework advances at an equally amazing pace. As hardware advancements have made 64-bit architectures more prevalent, the framework kept pace, to provide excellent support for 64-bit programs. Future versions will assuredly continue to keep pace, providing ever-improving support for parallel and asynchronous programming to take advantage of similar hardware advances in multicore technology.

The GC is a pillar of the .NET framework, separating it from all other development platforms; indeed the success of the framework is *directly* tied to how effective the GC is. The GC's effectiveness is at least partly a function of how well it stays out of the way of developers, and avoids being overtly noticed by the user. Each new version of .NET has set out to make the GC less and less intrusive, yet more effective at managing memory – a powerful combination.

As a general rule, use Workstation GC for any program that the user will interact with directly. If you are running a server process, obviously you should use Server GC. Always test your application and configuration under the expected load to ensure that you have the optimal configuration. And remember: conventional wisdom may not always hold true.

Because the GC has gotten so good at staying out of our way, it is easy to forget that it is there, and easy to forget all that it is doing for us. This makes it even easier to get into trouble when we venture into areas such as marshaling, where we explicitly bypass it. Any code relying on marshaling should be consolidated to a single location, making it easier to monitor. Such code should be carefully reviewed to make sure that it continues to be well behaved.

If you stay mindful of the impact your code has on the GC, and the impact the GC has on your code's performance, then you'll find your applications will always be the better for it.