

Username: Pralay Patoria **Book:** Pro .NET Performance. 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.

Best Practices for Interop

The following is a summary list of best practices for high-performance interop:

- Design interfaces to avoid native-managed transitions, by chunking (combining) work.
- Reduce round trips with a façade.
- Implement `IDisposable` if unmanaged resources are held across calls.
- Consider using memory pooling or unmanaged memory.
- Consider using unsafe code for re-interpreting data (e.g. in network protocols).
- Explicitly name the function you call and use `ExactSpelling=true`.
- Use blittable parameter types whenever possible.
- Avoid Unicode to ANSI conversions where possible.
- Manually marshal strings to/from `IntPtr`.
- Use C++/CLI for better control and performance for C/C++ and COM interop.
- Specify `[In]` and `[Out]` attributes to avoid unnecessary marshaling.
- Avoid long lifetime for pinned objects.
- Consider calling `ReleaseComObject`.
- Consider using `SuppressUnmanagedCodeSecurityAttribute` for performance-critical full-trust scenarios.
- Consider using `TLBIMP /unsafe` for performance-critical full-trust scenarios.
- Reduce or avoid COM cross-apartment calls.
- If appropriate, use `ASPCOMPAT` attribute in ASP.NET to reduce or avoid COM cross-apartment calls.