Mini-Tutorial: Using TokenManager for Robust Task Cancellation

To ensure that all running tasks, threads, or asynchronous operations can be properly cancelled and cleaned up during rebuilds or unloads, use a central TokenManager. This approach helps you avoid resource leaks and ensures system stability, especially in dynamic environments.

# 1. Creating and Using Cancellation Tokens

Use TokenManager.AToken() whenever you start a long-running task, async operation, or thread. This will register a new CancellationTokenSource internally and provide a CancellationToken for use in your code.

Example:

public static async Task ASleep(int delay)  
{  
 await Task.Delay(delay, TokenManager.AToken());  
}

# 2. Cancelling All Tasks During Rebuild/Unload

When you perform a rebuild or unload your project, call TokenManager.CancelAll(). This will cancel all registered tokens and ensure that all dependent tasks or threads are stopped gracefully.

Example:

TokenManager.CancelAll();

# 3. Best Practices

- Always acquire your CancellationToken from TokenManager for any task that should be cancellable.  
- Ensure your async methods, threads, or loops regularly check for token.IsCancellationRequested.  
- Document this requirement for all developers working on the project.

# 4. Example: Complete TokenManager Implementation

public static class TokenManager  
{  
 private static readonly List<CancellationTokenSource> \_sources = new();  
  
 public static CancellationToken AToken()  
 {  
 var cts = new CancellationTokenSource();  
 \_sources.Add(cts);  
 return cts.Token;  
 }  
  
 public static void CancelAll()  
 {  
 foreach (var cts in \_sources)  
 cts.Cancel();  
 \_sources.Clear();  
 }  
}

# Summary

Using a central TokenManager ensures all background work can be cancelled safely and reliably, especially during reloads or shutdowns. This prevents resource leaks and keeps your application robust.