ATask – Managed Task Lifecycle Utility

ATask provides a managed, registered, and cancellable abstraction for background task execution in .NET/C#. It supports both fire-and-forget and value-returning asynchronous operations, integrating with a central TaskManager for lifecycle tracking. This ensures all tasks can be monitored, cancelled, and cleaned up – especially important in modular, reloadable applications.

# Key Features

- Named task instances for traceability  
- Registration/deregistration with TaskManager  
- Cooperative cancellation via CancellationToken  
- Awaitable and fire-and-forget task variants  
- Exception handling and lifecycle logging

# ATask Code Examples

1. Fire-and-Forget ATask (no result):

public class ATask  
{  
 public string InstanceName { get; }  
 private CancellationTokenSource \_cts = new();  
 private Task \_task;  
  
 public bool IsRunning => !\_task.IsCompleted && !\_cts.IsCancellationRequested;  
  
 public ATask(string instanceName, Action<CancellationToken> work)  
 {  
 InstanceName = instanceName;  
 \_task = new Task(() => work(\_cts.Token), \_cts.Token);  
 TaskManager.Register(this);  
 }  
  
 public void Start()  
 {  
 if (\_task.Status == TaskStatus.Created)  
 \_task.Start();  
 }  
  
 public void Stop()  
 {  
 \_cts.Cancel();  
 TaskManager.Deregister(this);  
 }  
}

2. Generic ATask<TResult> (awaitable):

public class ATask<TResult>  
{  
 public string InstanceName { get; }  
 private CancellationTokenSource \_cts = new();  
 private Task<TResult> \_task;  
  
 public bool IsRunning => !\_task.IsCompleted && !\_cts.IsCancellationRequested;  
  
 public ATask(string instanceName, Func<CancellationToken, TResult> work)  
 {  
 InstanceName = instanceName;  
 \_task = new Task<TResult>(() => work(\_cts.Token), \_cts.Token);  
 TaskManager.Register(this);  
 }  
  
 public void Start()  
 {  
 if (\_task.Status == TaskStatus.Created)  
 \_task.Start();  
 }  
  
 public async Task<TResult> WaitAsync()  
 {  
 return await \_task;  
 }  
  
 public void Stop()  
 {  
 \_cts.Cancel();  
 TaskManager.Deregister(this);  
 }  
}

# How to Use

1. Create a new ATask with a name and a work delegate, passing the CancellationToken:  
 var myTask = new ATask("Worker1", token => { while (!token.IsCancellationRequested) { ... } });  
 myTask.Start();  
2. For result-returning tasks:  
 var myTask = new ATask<int>("Calc", token => { /\* ... \*/ return 42; });  
 myTask.Start();  
 var result = await myTask.WaitAsync();  
3. Stop any task at any time with myTask.Stop();  
All instances are automatically registered for management and can be safely terminated on app shutdown or project rebuild.

# Best Practices

- Always use the provided CancellationToken to exit loops and support cooperative cancellation.  
- Prefer ATask over direct Task usage for resource safety and app lifecycle management.

# Diagram: Task Lifecycle

TBA

# Summary

ATask makes asynchronous and background code safer and more maintainable in dynamic, modular apps.