Consuming NET Core Background Service with DI

To properly consumes a background service from a controller or another service, it must be registered with DI.

As well, since it’s a background service… it must be running… and probably a singleton.

> NOTE: If you are looking for how to generate an instance of ServiceProvider, outside of NET Core runtime, like for a unit test, see this: Duplicating .NET Core DI

See this reference for Host Service vs Background Service: A Complete Guide to Hosted Service(s) in .NET 6 using C# 10

> NOTE: If you are looking for how to access DI services, see this: HowTo Access DI Services

> NOTE: A Background Service actually derives from IHostedService.
> So, it is easiest to derive from BackgroundService and simply override methods as needed.

Without Dependencies

Here’s how to register a background service without any dependencies of its own…

```
public class Startup
{
    //rest of class
    public void ConfigureServices(IServiceCollection services)
    {
        //rest of method

        // Add the WSHost Manager Service...
        services.AddSingleton<WSHostMgr_Service>();
```

```
        services.AddSingleton<IHostedService>(svcprovider => svcprovider.GetService<WSHostMgr_Service>());
    }
}
```

> NOTE: The above has two steps.

First, the class is registered as a singleton. This will cause the DI to return the same instance everytime we ask for it.
As well, the runtime will not actually create the singleton until it is first requested. But, we do that in the next line.
The next line registers the singleton instance as a hosted service.
This is done by getting the singleton instance from the service provider, and returning it in the lambda.
It is then registered as a hosted service, and its StartAsync and ExecuteAsync will be called when the host starts.

With Dependencies

Some background services require dependencies.

Here’s how to register it, if it has dependencies…

```
public class Startup
{
    //rest of class
    public void ConfigureServices(IServiceCollection services)
    {
        //rest of method

        services.AddSingleton<ILoggerService>(sp =>
        {
            var hostAppLifetime = sp.GetService<IHostApplicationLifetime>();
            return new DatabaseLoggerService(hostAppLifetime);
        });
        services.AddHostedService(sp => sp.GetService<ILoggerService>() as DatabaseLoggerService);
    }
}
```

> NOTE: The above example includes a lambda, which will retrieve any dependencies the singleton requires, and instantiate it, explicitly.

Then, the singleton will be added as a Hosted Service.
NOTE: This is a two-step process. One step, registers the service to it is accessible with DI requests. The second step, gives it to the IHost to start and stop it as a Hosted Service.

Passing Service Provider

Since hosted services are long-lived, it is sometimes easier for them to be responsible for their own scoped dependencies.

To allow for this, we can simply give the hosted service an instance of IServiceProvider in its constructor.
Below is an example of doing that...

```
public class Startup
{
    //rest of class
    public void ConfigureServices(IServiceCollection services)
    {
        //rest of method


        // Add a singleton to DI, so controllers can access it....
        services.AddSingleton<Diagnostic_ForwardingService>(sp =>
        {
            return new Diagnostic_ForwardingService(sp);
        });
        // Add the service as a hosted service, but do it via DI...
        services.AddHostedService(sp => sp.GetService<Diagnostic_ForwardingService>() as
Diagnostic_ForwardingService);
    }
}
```

The above example includes a lambda in the Add Singleton call that will explicitly create an instance of the service, including giving it the Service Provider instance.

Then, the service is retrieved from DI, and added as a Hosted Service.

Fail-Early Startup

Below is an example of registering a Hosted Service that includes some setup and starting it, before the IHost does, to ensure it fails quickly and controllably.

```
public class Startup
{
	//rest of class
	public void ConfigureServices(IServiceCollection services)
	{
		//rest of method

        services.AddSingleton<WSHostMgr_Service>();
        services.AddHostedService<WSHostMgr_Service>(svcprovider =>
        {
            // Get the singleton instance that we just registered...
            var svc = svcprovider.GetService<WSHostMgr_Service>();

            // NOTE: The singleton instance has not been started yet.
            // So, we can set it up, now.
            // Give it the delegate for client version evals...
            svc.DelVersionCompare =
WSHost_AppVersionEvaluator.Determine_WSHostVersionRange_forClientVersion;
            // Tell it to startup...
            if(svc.StartupMgr() != 1)
            {
                OGA.SharedKernel.Logging_Base.Logger_Ref?.Error($"WSHostMgr_Service failed to startup.");
                Console.Error.WriteLine($"WSHostMgr_Service failed to startup.");
                throw new Exception("WSHostMgr_Service failed to start.");
            }
            // If here, the service has started.

            // Register the hosted service...
            return svc;
        });
    }
}
```

In the above example, the service is registered as a singleton.

Then, a lambda is used in the Add Hosted Service call, which will get the singleton from DI, do some setup of it, start it to ensure it works, and return it to the Add Hosted call.

Revision #2
Created 30 April 2025 01:54:02 by glwhite
Updated 30 April 2025 02:12:01 by glwhite