💎 C# Assemblies – Notes That Make You Shine 💥

🧠 What is an Assembly?

A **C# Assembly** is the smallest deployable unit in .NET. It's basically the compiled output of your code, typically in .dll or .exe form.

- 🔧 **Contains**: code, metadata, version info, culture info, digital signature (optional).
- 📦 It can have **one or more namespaces, classes, methods, etc.**
- 🧩 Two types:
    - **Private Assembly** – used by a single application.
    - **Shared Assembly** – can be used by multiple applications (usually in the GAC).
    - 🔄 **What is a Shared Assembly?**
    - A **Shared Assembly** is a type of assembly designed to be used by **multiple applications** at the same time. Instead of each application having its own private copy of a DLL, all of them can share **one common version**.
    - ---
    - 🔧 **Where is it stored?**
    - Shared Assemblies are stored in a special location in the Windows operating system called:
    - **GAC – Global Assembly Cache**
      It's a system folder that holds strongly-named assemblies meant for sharing across applications.
    - ---
    - 📜 **What are the requirements to create a Shared Assembly?**
    - To make an assembly shareable via the GAC, it must meet the following conditions:
    - **• Strong Name**

- The assembly must be signed with a **strong name** – a unique identity using a cryptographic key (.snk file) to ensure integrity and trust.
- **• Full Identity (Assembly Metadata)**
- The assembly must include a complete identity consisting of:
- **Name -Version-Culture-Public Key Token**
- ---
- 💡 **When should you use a Shared Assembly?**
- Use a shared assembly when:
- You have a library used by **multiple applications** (e.g., a logging library, security module, or data parser).
- You want **centralized version control** and updates.
- A **team maintains a shared component** that other teams/projects depend on.
- ---
- 🚫 **Potential Drawbacks**
- **Versioning Conflicts:** Updating a shared assembly might break other applications if they're not compatible with the new version.
- **Requires Signing:** You need to sign it and have admin-level access to the GAC.
- **Harder to Trace:** Troubleshooting is harder compared to private assemblies embedded in each app.
- ---
- 🧪 **Real-Life Example:**
- Your company has three applications:
- Accounting System -Purchasing System-Payroll System
- All of them need access to the **same user authentication library**.
- Instead of copying the DLL into each app, you register the library once in the **GAC**, and all three applications use the shared version from there.

🔶 Point	GAC (Global Assembly Cache)	NuGet Packages
Definition	A central location in Windows that stores assemblies shared across applications.	A package management system for open-source libraries and custom project dependencies.
Purpose	To share a single assembly between multiple applications on the same machine.	To easily share, manage, and consume libraries across projects, often from online sources.
Scope/Level	System-wide (Local Machine Level)	Project-specific / Developer level
Versioning Control	Difficult – requires manual registration and binding redirects.	Easy – each project can define its own version without conflict.
Environment	Works only within Windows environments.	Works across any environment that supports .NET and has internet access or local cache.
Distribution	Manual – assemblies must be installed into the GAC manually (e.g., via gacutil).	Online – packages can be published to NuGet.org and accessed globally.
Multiple Versions	Possible, but may lead to version conflicts unless handled via binding redirection.	Fully supported – different projects can use different versions without issues.
Updating	Requires uninstalling the old version and installing the new one.	Simple – just update the version number in csproj or packages.config.
Popular Example	A shared logging library used by 10 different applications on a company server.	Newtonsoft.Json – a widely used library for JSON parsing and serialization.

📦 Assembly = Metadata + IL Code + Resources + Manifest

Component	Description
Metadata	Info about types (classes, methods, properties, etc.)
IL Code	Intermediate Language code (compiled, not yet native)
Resources	Images, strings, files embedded
Manifest	Metadata about the assembly itself (version, culture, etc.)

🔍 Useful Assembly Properties via Reflection

```
var assembly = Assembly.GetExecutingAssembly();
var name = assembly.GetName();
Console.WriteLine("Name: " + name.Name);
Console.WriteLine("Version: " + name.Version);
Console.WriteLine("Culture: " + name.CultureName);
Console.WriteLine("Public Key: " + BitConverter.ToString(name.GetPublicKey()));
Console.WriteLine("Code Base: " + assembly.CodeBase);
Console.WriteLine("Is Dynamic: " + assembly.IsDynamic);
```

📜 Public Key? Signing? 🤔

- Assemblies **can be signed** using a .snk key file.
- This ensures the integrity of the assembly (like a digital signature).
- Signed assemblies can be shared safely and put in the **GAC** (Global Assembly Cache).

```
[assembly: AssemblyKeyFile("key.snk")]
```

📚 What's inside an Assembly?

```
foreach (Type t in assembly.GetTypes())
{
   Console.WriteLine(t.FullName);
}
```

You can inspect:

- ✅ Methods (t.GetMethods(...))
- ✅ Fields (t.GetFields(...))
- ✅ Properties (t.GetProperties(...))
- ✅ Custom Attributes (assembly.GetCustomAttributes())

💬 Properties vs Fields

Property	Field
Uses get/set	Raw variable (direct memory)
More flexible	Simpler, faster
Backed by field	Not backed by property
Can have validation	Can't

Fields usually appear like: <Name>k__BackingField behind auto-properties.

🔐 Binding Flags – Unlocking Reflection

t.GetMethods(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static | BindingFlags.DeclaredOnly)

- Public: include public members
- NonPublic: include private/protected members
- Instance: instance methods

- Static: static methods
- DeclaredOnly: ignore inherited members

Tip: Use DeclaredOnly to avoid inherited base class methods if you want only the ones defined in the current class.

🎯 Why Should I Care About Assemblies?

- ✅ **Deploy your code** as libraries or tools.
- ✅ **Inspect and analyze** external libraries (using reflection).
- ✅ **Build plugins or extensibility** (e.g., load assemblies dynamically).
- ✅ **Secure your code** using signing.
- ✅ **Version control** for libraries.

🧪 Bonus: Load Assembly Dynamically

var asm = Assembly.LoadFrom("MyLibrary.dll");

Great for plugin systems or building extensible architectures 💡

🚀 Real-Life Example Use Cases

- You're loading modules at runtime for a **plugin system**.
- You're creating a tool to **scan and analyze assemblies** for types or methods.
- You want to validate that a third-party library **hasn't been tampered with** (check public key).
- You're using **Roslyn** or dynamic code execution? You'll often deal with **dynamic assemblies** (IsDynamic == true).

🔚 TL;DR: Assemblies = Heart of .NET Runtime

بدون بالـAssembly الكود بتاعنا مجرد سطور. ، الكود بيبقى قابل للتشغيل، التوزيع، التحليل، والأمانAssembly.