DotNet Assembly Searching

Here are notes about the Assembly Helper classes that are part of OGA.SharedKernel.Lib and OGA.Common.Lib.

Use Cases

There are several use cases for searching loaded and referenced assemblies for classes and types. Here are some examples:

- Identify top-level derived configuration types
  This is handy, to determine the most-derived type in an application, so all base types of it can point to it.
- Find all classes that derive from a common base
  One example is finding all API controller types, so they can be hooked up for DI.
- Find all types decorated with a particular attribute
  This is another way of finding types for a shared purpose, in case they don't all share a common base.
- Determine if a type is a base for a set of classes
  This is handy when we need to determine
- Output assembly list to the log or for dependency admin
  This is good diagnostic for a starting process, so a developer can review the startup section of the log to determine if extraneous dependencies exist.
- Output runtime dependencies to a Central Catalog Service
  Collecting the list of used and referenced assemblies, of a running process, helps determine what processes need to be updated to current library versions.
- Determine if any referenced assemblies are duplicates with different version, PublicKeyToken, or CultureInfo.
  This would be libraries that are referenced by other library dependencies, but not loaded, and cannot be located.
- Determine if any referenced assemblies are missing.
  This usually occurs when two libraries (or a library and the process) have a dependency on a library, but are expecting different versions of it.

How Managed

Since the logic to properly iterate assemblies and references is non-trivial, we've centralized it to a single class type called, Assembly Helper.

> NOTE: See Revision Info section, below, for updated types and interfaces.

The implementation of this class type, currently exists in OGA.Common.Lib.
But, a static reference is exposed in OGA.SharedKernel, so that libraries can use it as needed.

Doing so, requires three things:

- IAssemblyHelper - Assembly Helper Interface
- Assembly Helper Static Reference (held by called: AssemblyHelper_Base)
- Assembly Helper Implementation (called, AssemblyHelper_v3)

IAssemblyHelper

This interface provides the call surface for all consuming libraries and code to work query assembly data.

It is named, IAssemblyHelper, and located in OGA.SharedKernel.
The interface holds the list of queries that can be performed against assemblies, to support the use cases above.

Static Reference in AssemblyHelper_Base

Also in the OGA.SharedKernel library, is the AssemblyHelper_Base type.
This type includes the public static reference to the available Assembly Helper instance for the process.

The static reference is the calling point for all consuming libraries and code.

The base type is also a sentinel instance of the static reference, until a live Assembly Helper is created.
The idea being that, if the running process (or unit test) doesn't initialize the Assembly Helper at startup, the base type will throw exceptions for the developer.

Assembly Helper Implementation

There's a class type, called AssemblyHelper_v2, that inherits from the base and interface, above, that includes the actual query logic.

It currently lives in OGA.Common.Lib.

Initialization

For consumers to have assembly data available, the assembly helper must be initialized on process startup.

This includes process start and unit test startup. See below for how to accomplish each.

During Process Start

During process start, an instance of the Assembly Helper needs to be created and applied to the static reference, so the process and any dependent libraries can use it.

This is done when a Program.cs:Main calls the Consolidate_Main method of Program_Base.cs (from OGA.Common.Lib), here:

```
// Setup the Assembly Helper, so it's available for any early startup logic...
{
  var ah = new OGA.Common.Process.AssemblyHelper_v3();
  OGA.SharedKernel.Process.AssemblyHelper_Base.SetRef(ah);
}
```

The above logic is part of Consolidated_Main. It creates and applies the instance for use.

During Unit Tests

Similar to process start, unit tests also need to stand up an instance of the Assembly Helper, so any dependent libraries can use it.

This is done by the Assembly Initialize method of the `TestAssembly_Base` from OGA.Testing.Lib.

To leverage this, just include a copy of the TestTemplate_Assembly in any unit test project you create.
It will automatically standup the Assembly Helper instance before tests run.

Revision Info

The Assembly Helper is on its third version.

Here are the implementation versions:

- V1 - AssemblyHelper (original, before improvements)
- V2 - AssemblyHelper_v2 (first update)
- V3 - AssemblyHelper_v3 (as of 20250119)

V1

V1 was the first implementation of the Assembly Helper.
It existed as only the AssemblyHelper class type, with no interface or base class.

V2

V2 was introduced when there became a need for libraries to use the functionality of AssemblyHelper, but not require a dependency on OGA.Common.Lib.
To accomplish this, V2 split the Assembly Helper into the three components:

- Interface
- Base
- Implementation

This obviously raised the complexity of the Assembly Helper, but allows libraries to reference just OGA.SharedKernel.Lib, while the parent process references OGA.Common.Lib, and initializes the AssemblyHelper implementation.
As well, this allows future modifications to the class type, while maintaining a common interface for all versions.

Second is a base class that holds the static public instance of the Assembly Helper, once initialized. This similar to how V1 exposed its static instance.

The interface and base class are both part of OGA.SharedKernel.Lib, so they can be referenced by all libraries and process types.

The implementation (AssemblyHelper_v2) lives in OGA.Common.Lib, and gets initialized on process start or Unit test start.

V3

V3 was created to address the diagnostic need to identify missing libraries and duplicate libraries with differing versions.

This version created a new Interface (v3) that adds a few diagnostic public property lists for duplicate and missing assemblies.
The interface also includes a public property to identify the class version of the implementation, making it easier to future proof.

This version also created a new implementation (v3) that performs the startup checks for duplicate and missing assemblies, and publishes lists of any found.

This version (v3) inherits the V2 interface, providing the same functionality of v2.

Revision #7
Created 19 January 2025 10:15:40 by glwhite
Updated 19 March 2025 15:41:50 by glwhite