This sample demonstrates how Composer can emit new types dynamically, and connect their instances to an IEmittedTypeHandler.

Project name: "I.EmittedClass"

For information on how to get the code, and run the sample, please see About Basic Samples.

Description

The main program logic (functionality) of this sample is similar to previous samples, but the design is totally different. In this sample, there are no components, and calculations are performed by by dynamically created classes. This sample does NOT demonstrate Composer's functionality about plugging the components to each other.

The main program, in the Program.cs file, creates a new instance of ComponentContext, and gets an IClassEmitter component from it. This component is one of the Composer's built-in components that support runtime type generation.

The IClassEmitter instance is then used to create three classes, emitted at runtime, each implementing one of the following interfaces:

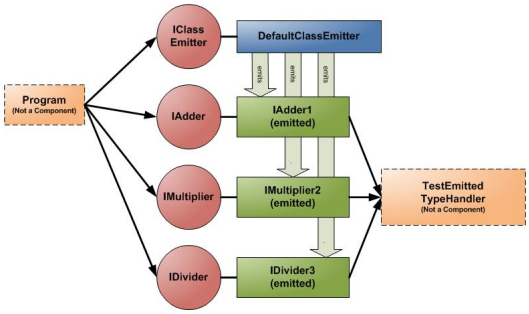
- IAdder
- IMultiplier
- IDivider

Each instance of the dynamically created types needs an IEmittedTypeHandler object that is used to handle the calls to the methods of the original interface. For example, when adder.Add(67, 12) is called, the dynamically created implementation calls HandleCall method of the given IEmittedTypeHandler and tells it that a method with the name of "Add", with two integer parameters, and with values 67 and 12 is called.

The TestEmittedTypeHandler class in the sample demonstrates an implementation of the IEmittedTypeHandler that is able to handle calls from all three interfaces mentioned above. It checks the method name, and takes the appropriate action based on the name of the method called. The main program creates a new instance of TestEmittedTypeHandler class and passes it to the classEmitter to be used in the dynamic class instances.

Note that, since none of the above interfaces contain any properties or events, only HandleCall method of the TestEmittedTypeHandler class contains implementation. Other methods just throw a NotImplementedException, and will never be called in our sample.

Dependency Diagram



Sample Output

INVOCATION - TestEmittedTypeHandler.HandleCall reflectedType = IAdder methodName = Add argumentTypes = Int32, Int32 arguments = 67, 12 resultType = Int32 67 + 12 = 79 INVOCATION - TestEmittedTypeHandler.HandleCall reflectedType = IMultiplier methodName = Multiply argumentTypes = Int32, Int32 arguments = 67, 12 resultType = Int32 67 * 12 = 804 INVOCATION - TestEmittedTypeHandler.HandleCall reflectedType = IDivider methodName = Divide argumentTypes = Int32, Int32 arguments = 67, 12 resultType = Int32 67 / 12 = 5 INVOCATION - TestEmittedTypeHandler.HandleCall reflectedType = IDivider methodName = Remainder argumentTypes = Int32, Int32 arguments = 67, 12 resultType = Int32 67 % 12 = 7