This sample demonstrates how to register dynamically emitted types as a component in the Composer. It's a combination of two previous basic sample, Emitted Class and Listener .

Project name: "J.EmittedAsComponent"

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

Description

The functionality and composition of this sample is exactly the same as the Listener sample, but the three components listed below is removed from the code:

- DefaultAdder, which provided IAdder contract, is removed
- DefaultMultiplier, which providied the IMultiplier contract, is removed
- DefaultDivider, which provided the IDivider contract, is removed

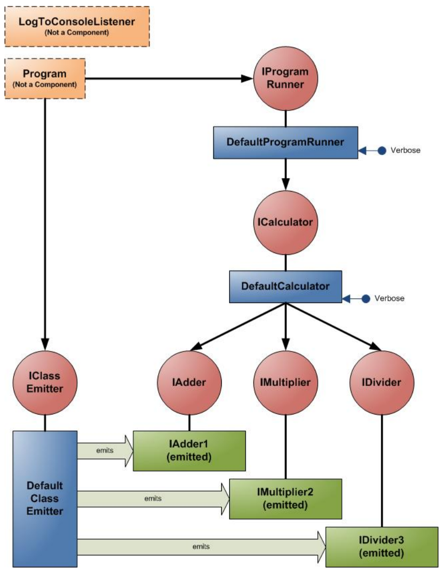
Instead of the above components, three classes each implementing one of the above contracts, are emitted using the method described in the previous sample, Emitted Class .

In this sample, the emitted classes participate in component composition like the components before. For this to be possible, the generated instances are needed to be registered in the ComponentContext as components.

In the main program, after setting up the context using CalculatorComposition.xml, the three emitted classes are generated similar to the previous sample . But instead of being used directly, they are passed to new instances of PreInitializedComponentFactory class, and registered in the context in addition to the other components.

When the program requests an instance of IProgramRunner in the last line of the main program, Composer matches the required plugs of the DefaultCalculator component to the dynamically generated components and plugs them, so that the program can run as before.

Dependency Diagram



Sample output

```
  LISTENER     - OnComponentCreated: IClassEmitter  LISTENER     - OnComponentCreated:
IMethodEmitter  LISTENER     - OnComponentComposed: IMethodEmitter  LISTENER     -
OnComponentRetrieved: IMethodEmitter  LISTENER     - OnComponentCreated:
IPropertyEmitter  LISTENER     - OnComponentComposed: IPropertyEmitter  LISTENER     -
OnComponentRetrieved: IPropertyEmitter  LISTENER     - OnComponentCreated:
IEventEmitter  LISTENER     - OnComponentComposed: IEventEmitter  LISTENER     -
OnComponentRetrieved: IEventEmitter  LISTENER     - OnComponentComposed:
IClassEmitter  LISTENER     - OnComponentRetrieved: IClassEmitter  CONSTRUCTOR  -
DefaultProgramRunner  LISTENER     - OnComponentCreated: IProgramRunner
CONSTRUCTOR  - DefaultCalculator  LISTENER     - OnComponentCreated: ICalculator  SET
PLUG     - DefaultCalculator.Adder(IAdder2)  SET PLUG     -
DefaultCalculator.Multiplier(IMultiplier3)  SET PLUG     - DefaultCalculator.Divider(IDivider4)
SET CONFIG   - DefaultCalculator.Verbose(True)  LISTENER     - OnComponentComposed:
ICalculator  NOTIFICATION - DefaultCalculator: OnCompositionComplete.  LISTENER     -
```

```
OnComponentRetrieved: ICalculator  SET PLUG     -
DefaultProgramRunner.Calculator(DefaultCalculator)  SET CONFIG   -
DefaultProgramRunner.Verbose(True)  LISTENER     - OnComponentComposed:
IProgramRunner  NOTIFICATION - DefaultProgramRunner: OnCompositionComplete.
LISTENER     - OnComponentRetrieved: IProgramRunner    METHOD CALL  -
DefaultProgramRunner.Run()    METHOD CALL  - DefaultCalculator.Add(67, 12)  INVOCATION
  - TestEmittedTypeHandler.HandleCall      reflectedType = IAdder      methodName    = Add
argumentTypes = Int32, Int32      arguments     = 67, 12      resultType    = Int32  67 + 12 = 79
METHOD CALL  - DefaultCalculator.Subtract(67, 12)  INVOCATION   -
TestEmittedTypeHandler.HandleCall      reflectedType = IAdder      methodName    = Add
argumentTypes = Int32, Int32      arguments     = 67, -12      resultType    = Int32  67 - 12 = 55
METHOD CALL  - DefaultCalculator.Multiply(67, 12)  INVOCATION   -
TestEmittedTypeHandler.HandleCall      reflectedType = IMultiplier      methodName    = Multiply
    argumentTypes = Int32, Int32      arguments     = 67, 12      resultType    = Int32  67 * 12 =
804    METHOD CALL  - DefaultCalculator.Divide(67, 12)  INVOCATION   -
TestEmittedTypeHandler.HandleCall      reflectedType = IDivider      methodName    = Divide
argumentTypes = Int32, Int32      arguments     = 67, 12      resultType    = Int32  METHOD
CALL  - DefaultCalculator.Remainder(67, 12)  INVOCATION   -
TestEmittedTypeHandler.HandleCall      reflectedType = IDivider      methodName    =
Remainder      argumentTypes = Int32, Int32      arguments     = 67, 12      resultType    = Int32
67 / 12 = 5 (with remainder = 7)
```