Fixture is created as a separate class which extends ::benchmark::Fixture.

A fixture provides setUp & tearDown.
setUp is called once per benchmark instance, & same for tearDown.

→ extends Benchmark

```
class FIXTURE_BECHMARK_NAME : public ::benchmark::Fixture {
 public:
  void SetUp(const ::benchmark::State& state) override {
    if (state.thread_index() == 0) {
      assert(data.get() == nullptr);
      data.reset(new int(42));
    }
  }

  void TearDown(const ::benchmark::State& state) override {
    if (state.thread_index() == 0) {
      assert(data.get() != nullptr);
      data.reset();
    }
  }

  ~FIXTURE_BECHMARK_NAME() override { assert(data == nullptr); }

  std::unique_ptr<int> data;
};
```

↑ class → ANY Name, referenced in benchmark registration

```
BENCHMARK_DEFINE_F(FIXTURE_BECHMARK_NAME, Bar)(benchmark::State& st) {
  if (st.thread_index() == 0) {
    assert(data.get() != nullptr);
    assert(*data == 42);
  }
  for (auto _ : st) {
    assert(data.get() != nullptr);
    assert(*data == 42);
  }
  st.SetItemsProcessed(st.range(0));
}
BENCHMARK_REGISTER_F(FIXTURE_BECHMARK_NAME, Bar)->Arg(42);
BENCHMARK_REGISTER_F(FIXTURE_BECHMARK_NAME, Bar)->Arg(42)->ThreadPerCpu();
```

BENCHMARK_DEFINE_F creates a new class extending the user provided Fixture.

```
#define BENCHMARK_PRIVATE_DECLARE_F(BaseClass, Method)           \
 class BaseClass##_##Method##_Benchmark : public BaseClass {     \
  public:                                                        \
   BaseClass##_##Method##_Benchmark() {                          \
     this->SetName(#BaseClass "/" #Method);                      \
   }                                                             \
                                                                 \
  protected:                                                     \
   void BenchmarkCase(::benchmark::State&) override;             \
 };
```

→ extends Fixture, which extends Benchmark.

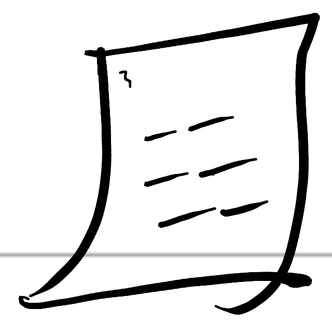
```
#define BENCHMARK_REGISTER_F(BaseClass, Method) \
  BENCHMARK_PRIVATE_REGISTER_F(BENCHMARK_PRIVATE_CONCAT_NAME(BaseClass, Method))

#define BENCHMARK_PRIVATE_REGISTER_F(TestName)           \
  BENCHMARK_PRIVATE_DECLARE(TestName) =                  \
      (::benchmark::internal::RegisterBenchmarkInternal( \
          ::benchmark::internal::make_unique<TestName>()))
```

↳ Name of the generated class

BENCHMARK_REGISTER_F creates a unique pointer to the newly generated class, & adds the pointer to BenchmarkFamilies.

Small note on benchmark execution



Be it the function benchmark

Or a fixture benchmark,

the benchmark library will call the f^n containing a for-loop over state multiple times.

Initially the function (/benchmark function) will be called with state.repetitions = 1, then 10,

then 100 ... & so on.

The benchmark function contains a for-loop over state & this function is called multiple times by benchmarking library.

If benchmark involves setup & tearDown, it will be called every time the benchmark fn gets invoked (teardown when finishes)

Fields of Benchmark class

↑ SOME

Name	Name of the benchmark
Arg	Argument to the benchmark It can be accessed using State.Range(0)
Unit	Use the timeunit for generated output
Args	Array of arguments to benchmark
Setup Teardown	} setup & teardown of benchmark.

Consider the following:

1. BENCHMARK(BM_Empty) → Arg(20);

2. BENCHMARK(BM_Empty) → Args({500, 1000});

1. Will create an instance of FunctionBenchmark, register with BenchmarkFamilies.

2. While executing, a Benchmark Instance will be created & 20 will be passed as argument to the benchmark function.

In 2, we have 2 arguments:

500 & 100.

1. A pointer to FunctionBenchmark will be created & registered with BenchmarkFamilies

2. While executing, 2 Benchmark-instances will be created, One with argument 500 & other with 1000. A benchmark instance with argument 500 will run multiple times (as

explained earlier), & then benchmark instance with argument 1000 will run.