Vanilla

```
class RateBase
{
  virtual ~RateBase() {}
  virtual bool sleep() = 0;
  virtual bool is_steady() const = 0;

  virtual void reset() = 0;
};

template<class Clock = std::chrono::high_resolution_clock>
class GenericRate : public RateBase
{
  explicit GenericRate(double rate)
  explicit GenericRate(std::chrono::nanoseconds period)
  virtual bool sleep()
  virtual bool is_steady() const

  virtual void reset()
  std::chrono::nanoseconds period() const
};

using Rate = GenericRate<std::chrono::system_clock>;

using WallRate = GenericRate<std::chrono::steady_clock>;
```

NOTE: Does not require ROS context to be initialized

Proposed in PR2123

```
class RateBase
{
  virtual ~RateBase() {}
  virtual bool sleep() = 0;
  [[deprecated]] virtual bool is_steady() const = 0;
  virtual rcl_clock_type_t get_type() const = 0;
  virtual void reset() = 0;
};

template<class Clock = std::chrono::high_resolution_clock>
class [[deprecated]] GenericRate : public RateBase
{
  explicit GenericRate(double rate)
  explicit GenericRate(std::chrono::nanoseconds period)
  virtual bool sleep()
  [[deprecated]] virtual bool is_steady() const
  virtual rcl_clock_type_t get_type() const ← because of base mandatory virtual method
  virtual void reset()
  std::chrono::nanoseconds period() const
};

class Rate : public RateBase
{
  explicit Rate(
    const double rate,
    Clock::SharedPtr clock = std::make_shared<Clock>(RCL_SYSTEM_TIME))
  explicit Rate(
    const Duration & period, ← Duration to be auto-constructed from std::chrono
    Clock::SharedPtr clock = std::make_shared<Clock>(RCL_SYSTEM_TIME))
  virtual bool sleep()
  [[deprecated]] virtual bool is_steady() const
  virtual rcl_clock_type_t get_type() const
  virtual void reset()
  Duration period() const ← std::chrono has no constructor from Duration

class WallRate : public Rate
{
  explicit WallRate(const double rate)
  : Rate(rate, std::make_shared<Clock>(RCL_STEADY_TIME))
  explicit WallRate(const Duration & period)
  : Rate(period, std::make_shared<Clock>(RCL_STEADY_TIME))
};

class ROSRate : public Rate ←new class
{
  explicit ROSRate(const double rate)
  : Rate(rate, std::make_shared<Clock>(RCL_ROS_TIME))
  explicit ROSRate(const Duration & period)
  : Rate(period, std::make_shared<Clock>(RCL_ROS_TIME))
};
```

NOTE: All classes as using Clock, require working context

Legend:

- The method/class to be deprecated in new version
- New parameter/class/method was introduced. Does not break old API.
- Parameter/ret.value type was changed. It does not cause API incompatibility. **Or new API will be incompatible with previous one.**
- *Important notice*