```
class RealTimeAsyncUpdater::RealTimeAsyncUpdateDispatcher    : private HighResolutionTimer,
                                                               private AsyncUpdater
{
public:
    RealTimeAsyncUpdateDispatcher()
    {
        startTimer (5);
    }

    ~RealTimeAsyncUpdateDispatcher()
    {
        cancelPendingUpdate();
    }

    void add (RealTimeAsyncUpdaterMessage&);
    void remove (RealTimeAsyncUpdaterMessage&);

private:
    void hiResTimerCallback() override
    {
        triggerAsyncUpdate();
    }

    void handleAsyncUpdate() override
    {
        serviceUpdaters();
    }

    void serviceUpdaters();

    CriticalSection lock;
    Array<RealTimeAsyncUpdaterMessage*> updaters;
};
```

```cpp
class RealTimeAsyncUpdater::RealTimeAsyncUpdateDispatcher   : private HighResolutionTimer,
                                                              private AsyncUpdater

{
public:
    RealTimeAsyncUpdateDispatcher()
    {
        startTimer (5);
    }

    ~RealTimeAsyncUpdateDispatcher()
    {
        cancelPendingUpdate();
    }

    void add (RealTimeAsyncUpdaterMessage&);
    void remove (RealTimeAsyncUpdaterMessage&);

private:
    void hiResTimerCallback() override
    {
        triggerAsyncUpdate();
    }

    void handleAsyncUpdate() override
    {
        serviceUpdaters();
    }

    void serviceUpdaters();

    CriticalSection lock;
    Array<RealTimeAsyncUpdaterMessage*> updaters;
};
```

```
class RealTimeAsyncUpdater::RealTimeAsyncUpdateDispatcher   : private HighResolutionTimer,
                                                              private AsyncUpdater
{
public:
    RealTimeAsyncUpdateDispatcher();
   ~RealTimeAsyncUpdateDispatcher();

    void add (RealTimeAsyncUpdaterMessage&);
    void remove (RealTimeAsyncUpdaterMessage&);

    void signal()
    {
        needsToService.store (true);
    }

private:
    void hiResTimerCallback() override
    {
        if (needsToService.exchange (false))
            triggerAsyncUpdate();
    }

    void handleAsyncUpdate() override
    {
        serviceUpdaters();
    }

    void serviceUpdaters();

    CriticalSection lock;
    Array<RealTimeAsyncUpdaterMessage*> updaters;
    std::atomic<bool> needsToService { false };
};
```