```
class RealTimeAsyncUpdater::RealTimeAsyncUpdaterMessage  : public ReferenceCountedObject
{
public:
    RealTimeAsyncUpdaterMessage (RealTimeAsyncUpdater& au)
        : owner (au)
    {
        dispatcher->add (*this);
    }

    ~RealTimeAsyncUpdaterMessage()
    {
        dispatcher->remove (*this);
    }

    void postUpdate()
    {
        shouldDeliver.set (1);
    }

    void serviceMessage()
    {
        if (shouldDeliver.compareAndSetBool (0, 1))
            owner.handleAsyncUpdate();
    }

    RealTimeAsyncUpdater& owner;
    Atomic<int> shouldDeliver;
    SharedResourcePointer<RealTimeAsyncUpdater::RealTimeAsyncUpdateDispatcher> dispatcher;

    JUCE_DECLARE_NON_COPYABLE (RealTimeAsyncUpdaterMessage)
};
```

Real-time safe

```
class RealTimeAsyncUpdater::RealTimeAsyncUpdaterMessage  : public ReferenceCountedObject
{
public:
    RealTimeAsyncUpdaterMessage (RealTimeAsyncUpdater& au)
        : owner (au)
    {
        dispatcher->add (*this);
    }

    ~RealTimeAsyncUpdaterMessage()
    {
        dispatcher->remove (*this);
    }

    void postUpdate()
    {
        shouldDeliver.set (1);
    }

    void serviceMessage()
    {
        if (shouldDeliver.compareAndSetBool (0, 1))
            owner.handleAsyncUpdate();
    }

    RealTimeAsyncUpdater& owner;
    Atomic<int> shouldDeliver;
    SharedResourcePointer<RealTimeAsyncUpdater::RealTimeAsyncUpdateDispatcher> dispatcher;

    JUCE_DECLARE_NON_COPYABLE (RealTimeAsyncUpdaterMessage)
};
```

$\longleftarrow$ **Real-time safe**

Trade-offs

- We now have a real-time safe way of posting a message