


```

43 //=====
44 AsyncUpdater::AsyncUpdater()
45 {
46     activeMessage = *new AsyncUpdaterMessage (*this);
47 }
48
49 AsyncUpdater::~~AsyncUpdater()
50 {
51     // You're deleting this object with a background thread while there's an update
52     // pending on the main event thread - that's pretty dodgy threading, as the update
53     // happen after this destructor has finished. You should either use a MessageQueue
54     // deleting this object, or find some other way to avoid such a race condition.
55     jassert (! isUpdatePending())
56         || MessageManager::getInstanceWithoutCreating() == nullptr
57         || MessageManager::getInstanceWithoutCreating()->currentThreadHasPriority();
58
59     activeMessage->shouldDeliver.set (0);
60 }
61
62 void AsyncUpdater::triggerAsyncUpdate()
63 {
64     // If you're calling this before (or after) the MessageManager is
65     // running, then you're not going to get any callbacks!
66     JUCE_ASSERT_MESSAGE_MANAGER_EXISTS
67
68     if (activeMessage->shouldDeliver.compareAndSetBool (1, 0))
69         if (! activeMessage->post())
70             cancelPendingUpdate(); // if the message queue fails, this avoids getting stuck
71                                     // trapped waiting for the message to arrive
72 }
73
74 void AsyncUpdater::cancelPendingUpdate() noexcept
75 {
76     activeMessage->shouldDeliver.set (0);
77 }
78
79 void AsyncUpdater::handleUpdateNowIfNeeded()
80 {
81     // This can only be called by the event thread.
82     JUCE_ASSERT_MESSAGE_MANAGER_IS_LOCKED
83
84     if (activeMessage->shouldDeliver.exchange (0) != 0)
85         handleAsyncUpdate();
86 }
87
88 bool AsyncUpdater::isUpdatePending() const noexcept
89 {
90     return activeMessage->shouldDeliver.value != 0;
91 }

```

```

186 //=====
187 RealTimeAsyncUpdater::RealTimeAsyncUpdater()
188 {
189     activeMessage = *new RealTimeAsyncUpdaterMessage (*this);
190 }
191
192 RealTimeAsyncUpdater::~~RealTimeAsyncUpdater()
193 {
194     // You're deleting this object with a background thread while there's an update
195     // pending on the main event thread - that's pretty dodgy threading, as the update
196     // happen after this destructor has finished. You should either use a MessageQueue
197     // deleting this object, or find some other way to avoid such a race condition.
198     jassert (! isUpdatePending())
199         || MessageManager::getInstanceWithoutCreating() == nullptr
200         || MessageManager::getInstanceWithoutCreating()->currentThreadHasPriority();
201
202     activeMessage->shouldDeliver.set (0);
203 }
204
205 void RealTimeAsyncUpdater::triggerAsyncUpdate()
206 {
207     // If you're calling this before (or after) the MessageManager is
208     // running, then you're not going to get any callbacks!
209     JUCE_ASSERT_MESSAGE_MANAGER_EXISTS
210
211     // Here we just set the atomic flag and wait for it to be serviced
212     activeMessage->postUpdate();
213 }
214
215 void RealTimeAsyncUpdater::cancelPendingUpdate() noexcept
216 {
217     activeMessage->shouldDeliver.set (0);
218 }
219
220 void RealTimeAsyncUpdater::handleUpdateNowIfNeeded()
221 {
222     // This can only be called by the event thread.
223     JUCE_ASSERT_MESSAGE_MANAGER_IS_LOCKED
224
225     if (activeMessage->shouldDeliver.exchange (0) != 0)
226         handleAsyncUpdate();
227 }
228
229 bool RealTimeAsyncUpdater::isUpdatePending() const noexcept
230 {
231     return activeMessage->shouldDeliver.value != 0;
232 }
233
234

```






```

43 //=====
44 AsyncUpdater::AsyncUpdater()
45 {
46     activeMessage = *new AsyncUpdaterMessage (*this);
47 }
48
49 AsyncUpdater::~~AsyncUpdater()
50 {
51     // You're deleting this object with a background thread while there's an up
52     // pending on the main event thread - that's pretty dodgy threading, as th
53     // happen after this destructor has finished. You should either use a Mess
54     // deleting this object, or find some other way to avoid such a race condi
55     jassert (! isUpdatePending())
56         || MessageManager::getInstanceWithoutCreating() == nullptr
57         || MessageManager::getInstanceWithoutCreating()->currentThreadHas
58
59     activeMessage->shouldDeliver.set (0);
60 }
61
62 void AsyncUpdater::triggerAsyncUpdate()
63 {
64     // If you're calling this before (or after) the MessageManager is
65     // running, then you're not going to get any callbacks!
66     JUCE_ASSERT_MESSAGE_MANAGER_EXISTS
67
68     if (activeMessage->shouldDeliver.compareAndSetBool (1, 0))
69         if (! activeMessage->post())
70             cancelPendingUpdate(); // if the message queue fails, this avoids
71                                     // trapped waiting for the message to arriv
72 }
73
74 void AsyncUpdater::cancelPendingUpdate() noexcept
75 {
76     activeMessage->shouldDeliver.set (0);
77 }
78
79 void AsyncUpdater::handleUpdateNowIfNeeded()
80 {
81     // This can only be called by the event thread.
82     JUCE_ASSERT_MESSAGE_MANAGER_IS_LOCKED
83
84     if (activeMessage->shouldDeliver.exchange (0) != 0)
85         handleAsyncUpdate();
86 }
87
88 bool AsyncUpdater::isUpdatePending() const noexcept
89 {
90     return activeMessage->shouldDeliver.value != 0;
91 }

```

```

186 //=====
187 RealTimeAsyncUpdater::RealTimeAsyncUpdater()
188 {
189     activeMessage = *new RealTimeAsyncUpdaterMessage (*this);
190 }
191
192 RealTimeAsyncUpdater::~~RealTimeAsyncUpdater()
193 {
194     // You're deleting this object with a background thread while there's an up
195     // pending on the main event thread - that's pretty dodgy threading, as th
196     // happen after this destructor has finished. You should either use a Mess
197     // deleting this object, or find some other way to avoid such a race condi
198     jassert (! isUpdatePending())
199         || MessageManager::getInstanceWithoutCreating() == nullptr
200         || MessageManager::getInstanceWithoutCreating()->currentThreadHas
201
202     activeMessage->shouldDeliver.set (0);
203 }
204
205 void RealTimeAsyncUpdater::triggerAsyncUpdate()
206 {
207     // If you're calling this before (or after) the MessageManager is
208     // running, then you're not going to get any callbacks!
209     JUCE_ASSERT_MESSAGE_MANAGER_EXISTS
210
211     // Here we just set the atomic flag and wait for it to be serviced
212     activeMessage->postUpdate();
213 }
214
215 void RealTimeAsyncUpdater::cancelPendingUpdate() noexcept
216 {
217     activeMessage->shouldDeliver.set (0);
218 }
219
220 void RealTimeAsyncUpdater::handleUpdateNowIfNeeded()
221 {
222     // This can only be called by the event thread.
223     JUCE_ASSERT_MESSAGE_MANAGER_IS_LOCKED
224
225     if (activeMessage->shouldDeliver.exchange (0) != 0)
226         handleAsyncUpdate();
227 }
228
229 bool RealTimeAsyncUpdater::isUpdatePending() const noexcept
230 {
231     return activeMessage->shouldDeliver.value != 0;
232 }
233
234

```

```
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)
};
```