

message thread calling handleAsyncUpdate() as soon as it can.

```
34
35
36     @tags[Events]
37 */
38 class JUCE_API AsyncUpdater
39 {
40 public:
41     //=====
42     /** Creates an AsyncUpdater object. */
43     AsyncUpdater();
44
45     /** Destructor.
46      If there are any pending callbacks when the object is deleted, these are lost.
47     */
48     virtual ~AsyncUpdater();
49
50     //=====
51     /** Causes the callback to be triggered at a later time.
52
53      This method returns immediately, after which a callback to the
54      handleAsyncUpdate() method will be made by the message thread as
55      soon as possible.
56
57      If an update callback is already pending but hasn't happened yet, calling
58      this method will have no effect.
59
60      It's thread-safe to call this method from any thread, BUT beware of calling
61      it from a real-time (e.g. audio) thread, because it involves posting a message
62      to the system queue, which means it may block (and in general will do on
63      most OSes).
64     */
65     void triggerAsyncUpdate();
66
67     /** This will stop any pending updates from happening.
68
69      If called after triggerAsyncUpdate() and before the handleAsyncUpdate()
70      callback happens, this will cancel the handleAsyncUpdate() callback.
71
72      Note that this method simply cancels the next callback - if a callback is already
73      in progress on a different thread, this won't block until the callback finishes, so
74      there's no guarantee that the callback isn't still running when the method returns.
75     */
76     void cancelPendingUpdate() noexcept;
77
78     /** If an update has been triggered and is pending, this will invoke it
79      synchronously.
80
81      Use this as a kind of "flush" operation - if an update is pending, the
82      handleAsyncUpdate() method will be called immediately; if no update is
83      pending, then nothing will be done.
84
85      Because this may invoke the callback, this method must only be called on
86      the main event thread.
87     */
88     void handleUpdateNowIfNeeded();
89
90     /** Returns true if there's an update callback in the pipeline. */
91     bool isUpdatePending() const noexcept;
92
93     //=====
94     /** Called back to do whatever your class needs to do.
95
96      This method is called by the message thread at the next convenient time
97      after the triggerAsyncUpdate() method has been called.
98     */
99     virtual void handleAsyncUpdate() = 0;
100
101 private:
102     //=====
103     class AsyncUpdaterMessage;
104     friend class ReferenceCountedObjectPtr<AsyncUpdaterMessage>;
105     ReferenceCountedObjectPtr<AsyncUpdaterMessage> activeMessage;
106
107     JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (AsyncUpdater)
108 };
109
110 } // namespace juce
```



Basically, one or more calls to the triggerAsyncUpdate() will result in the message thread calling handleAsyncUpdate() as soon as it can.

```
17
18
19
20     @tags[Events]
21 */
22 class RealTimeAsyncUpdater
23 {
24 public:
25     //=====
26     /** Creates a RealTimeAsyncUpdater object. */
27     RealTimeAsyncUpdater();
28
29     /** Destructor.
30      If there are any pending callbacks when the object is deleted, these are lost.
31     */
32     virtual ~RealTimeAsyncUpdater();
33
34     //=====
35     /** Causes the callback to be triggered at a later time.
36
37      This method returns immediately, after which a callback to the
38      handleAsyncUpdate() method will be made by the message thread as
39      soon as possible.
40
41      If an update callback is already pending but hasn't happened yet, calling
42      this method will have no effect.
43
44      It's thread-safe to call this method from any thread.
45     */
46     void triggerAsyncUpdate();
47
48     /** This will stop any pending updates from happening.
49
50      If called after triggerAsyncUpdate() and before the handleAsyncUpdate()
51      callback happens, this will cancel the handleAsyncUpdate() callback.
52
53      Note that this method simply cancels the next callback - if a callback is already
54      in progress on a different thread, this won't block until the callback finishes, so
55      there's no guarantee that the callback isn't still running when the method returns.
56     */
57     void cancelPendingUpdate() noexcept;
58
59     /** If an update has been triggered and is pending, this will invoke it
60      synchronously.
61
62      Use this as a kind of "flush" operation - if an update is pending, the
63      handleAsyncUpdate() method will be called immediately; if no update is
64      pending, then nothing will be done.
65
66      Because this may invoke the callback, this method must only be called on
67      the main event thread.
68     */
69     void handleUpdateNowIfNeeded();
70
71     /** Returns true if there's an update callback in the pipeline. */
72     bool isUpdatePending() const noexcept;
73
74     //=====
75     /** Called back to do whatever your class needs to do.
76
77      This method is called by the message thread at the next convenient time
78      after the triggerAsyncUpdate() method has been called.
79     */
80     virtual void handleAsyncUpdate() = 0;
81
82 private:
83     //=====
84     class RealTimeAsyncUpdateDispatcher;
85     class RealTimeAsyncUpdaterMessage;
86     friend class ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage>;
87     ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage> activeMessage;
88
89     JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (RealTimeAsyncUpdater)
90 };
91
```












```

34     message thread calling handleAsyncUpdate() as soon as it can.
35
36     @tags(Events)
37 */
38 class JUCE_API AsyncUpdater
39 {
40 public:
41 //=====
42 /** Creates an AsyncUpdater object. */
43 AsyncUpdater();
44
45 /** Destructor.
46     If there are any pending callbacks when the object is deleted, these are lost.
47 */
48 virtual ~AsyncUpdater();
49
50 //=====
51 /** Causes the callback to be triggered at a later time.
52
53     This method returns immediately, after which a callback to the
54     handleAsyncUpdate() method will be made by the message thread as
55     soon as possible.
56
57     If an update callback is already pending but hasn't happened yet, calling
58     this method will have no effect.
59
60     It's thread-safe to call this method from any thread, BUT beware of calling
61     it from a real-time (e.g. audio) thread, because it involves posting a message
62     to the system queue, which means it may block (and in general will do on
63     most OSes).
64 */
65 void triggerAsyncUpdate();
66
67 /** This will stop any pending updates from happening.
68
69     If called after triggerAsyncUpdate() and before the handleAsyncUpdate()
70     callback happens, this will cancel the handleAsyncUpdate() callback.
71
72     Note that this method simply cancels the next callback - if a callback is already
73     in progress on a different thread, this won't block until the callback finishes, so
74     there's no guarantee that the callback isn't still running when the method returns.
75 */
76 void cancelPendingUpdate() noexcept;
77
78 /** If an update has been triggered and is pending, this will invoke it
79     synchronously.
80
81     Use this as a kind of "flush" operation - if an update is pending, the
82     handleAsyncUpdate() method will be called immediately; if no update is
83     pending, then nothing will be done.
84
85     Because this may invoke the callback, this method must only be called on
86     the main event thread.
87 */
88 void handleUpdateNowIfNeeded();
89
90 /** Returns true if there's an update callback in the pipeline. */
91 bool isUpdatePending() const noexcept;
92
93 //=====
94 /** Called back to do whatever your class needs to do.
95
96     This method is called by the message thread at the next convenient time
97     after the triggerAsyncUpdate() method has been called.
98 */
99 virtual void handleAsyncUpdate() = 0;
100
101 private:
102 class AsyncUpdaterMessage;
103 friend class ReferenceCountedObjectPtr<AsyncUpdaterMessage>;
104 ReferenceCountedObjectPtr<AsyncUpdaterMessage> activeMessage;
105
106 JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (AsyncUpdater)
107 };
108
109 } // namespace juce

```



```

17 Basically, one or more calls to the triggerAsyncUpdate() will result in the
18 message thread calling handleAsyncUpdate() as soon as it can.
19
20 @tags(Events)
21 */
22 class RealTimeAsyncUpdater
23 {
24 public:
25 //=====
26 /** Creates a RealTimeAsyncUpdater object. */
27 RealTimeAsyncUpdater();
28
29 /** Destructor.
30     If there are any pending callbacks when the object is deleted, these are lost.
31 */
32 virtual ~RealTimeAsyncUpdater();
33
34 //=====
35 /** Causes the callback to be triggered at a later time.
36
37     This method returns immediately, after which a callback to the
38     handleAsyncUpdate() method will be made by the message thread as
39     soon as possible.
40
41     If an update callback is already pending but hasn't happened yet, calling
42     this method will have no effect.
43
44     It's thread-safe to call this method from any thread.
45 */
46 void triggerAsyncUpdate();
47
48 /** This will stop any pending updates from happening.
49
50     If called after triggerAsyncUpdate() and before the handleAsyncUpdate()
51     callback happens, this will cancel the handleAsyncUpdate() callback.
52
53     Note that this method simply cancels the next callback - if a callback is already
54     in progress on a different thread, this won't block until the callback finishes, so
55     there's no guarantee that the callback isn't still running when the method returns.
56 */
57 void cancelPendingUpdate() noexcept;
58
59 /** If an update has been triggered and is pending, this will invoke it
60     synchronously.
61
62     Use this as a kind of "flush" operation - if an update is pending, the
63     handleAsyncUpdate() method will be called immediately; if no update is
64     pending, then nothing will be done.
65
66     Because this may invoke the callback, this method must only be called on
67     the main event thread.
68 */
69 void handleUpdateNowIfNeeded();
70
71 /** Returns true if there's an update callback in the pipeline. */
72 bool isUpdatePending() const noexcept;
73
74 //=====
75 /** Called back to do whatever your class needs to do.
76
77     This method is called by the message thread at the next convenient time
78     after the triggerAsyncUpdate() method has been called.
79 */
80 virtual void handleAsyncUpdate() = 0;
81
82 private:
83 class RealTimeAsyncUpdateDispatcher;
84 class RealTimeAsyncUpdaterMessage;
85 friend class ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage>;
86 ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage> activeMessage;
87
88 JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (RealTimeAsyncUpdater)
89 };
90
91 }

```

```

class RealTimeAsyncUpdateDispatcher;
class RealTimeAsyncUpdaterMessage;
friend class ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage>;
ReferenceCountedObjectPtr<RealTimeAsyncUpdaterMessage> activeMessage;

```



juce::AsyncUpdater

```
101 private:
102     //=====
103     class AsyncUpdaterMessage;
104     friend class ReferenceCountedObjectPtr<AsyncUpdaterMessage>;
105     ReferenceCountedObjectPtr<AsyncUpdaterMessage> activeMessage;
106
107     JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (AsyncUpdater)
108 };
```