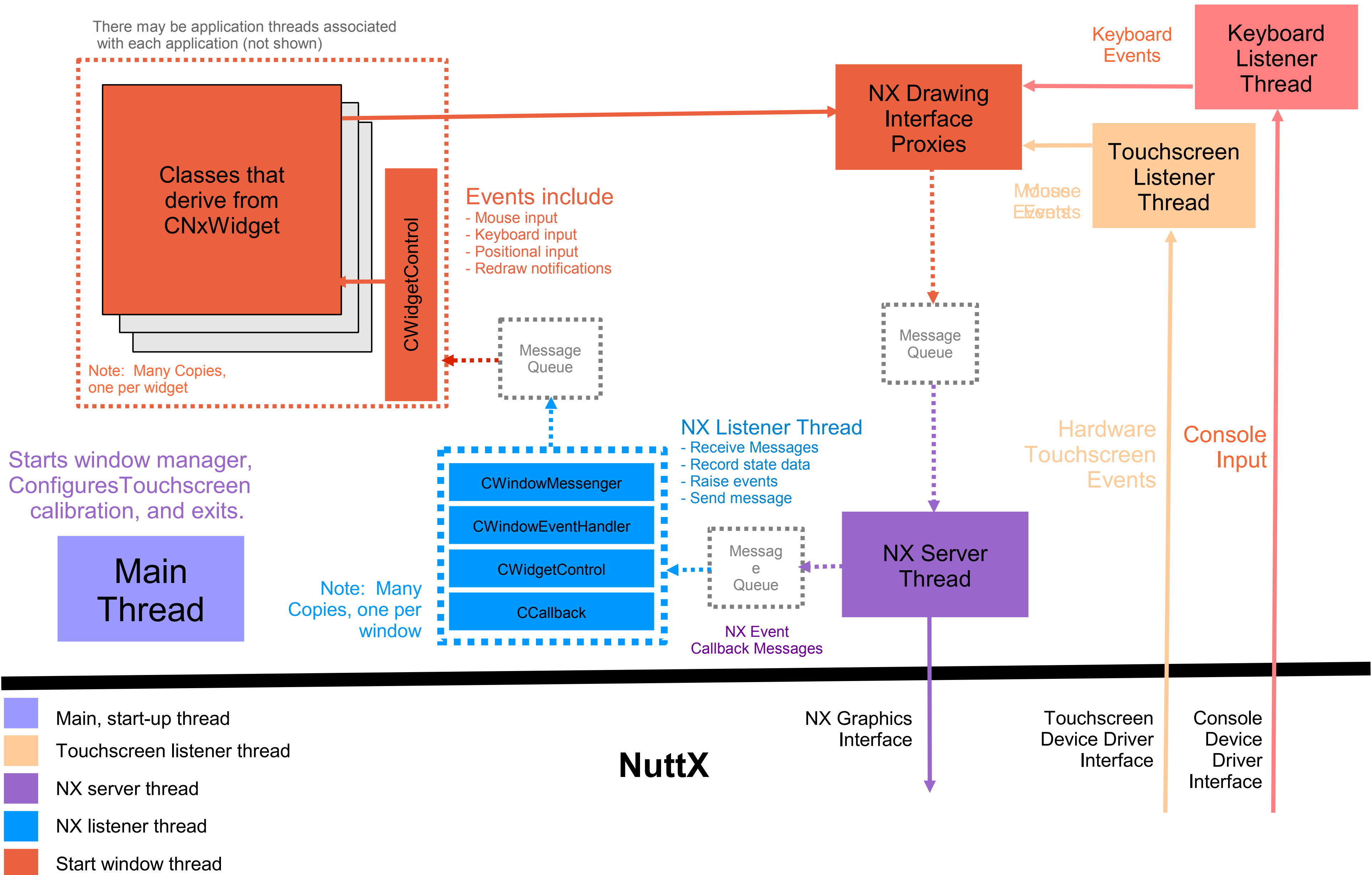
NxWM Threading Model



Start Window Task

The start window task drives all widgets. The function receives window events from the NX listener threads indirectly through this sequence:

1) NX listener thread receives a windows event. This may be a positional change notification, a redraw request, or mouse or keyboard input.

2) The NX listener thread performs the callback by calling a NXWidgets::CCallback method associated with the window.

3) NXWidgets::CCallback calls into NXWidgets::CWidgetControl to process the event.

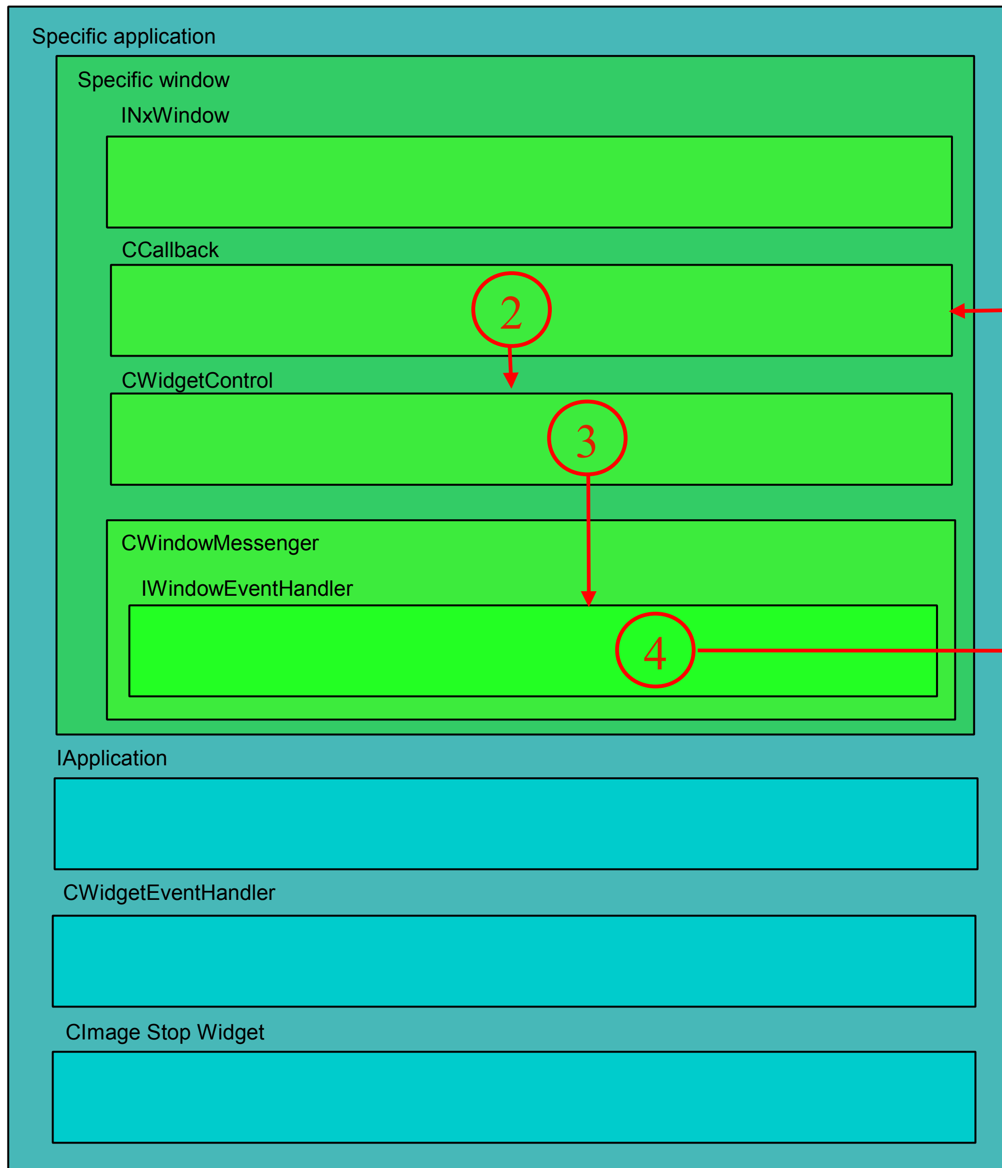
4) NXWidgets::CWidgetControl records the new state data and raises a window event.

5) NXWidgets::CWindowEventHandlerList will give the event to NxWM::CWindowMessenger.

6) NxWM::CWindowMessenger will send the a message on a well-known message queue.

7) This CStartWindow::startWindow task will receive and process that message.

Window Events



1. Redraw, position change, keyboard or mouse input or blocked event received and a static CCallback method is called.

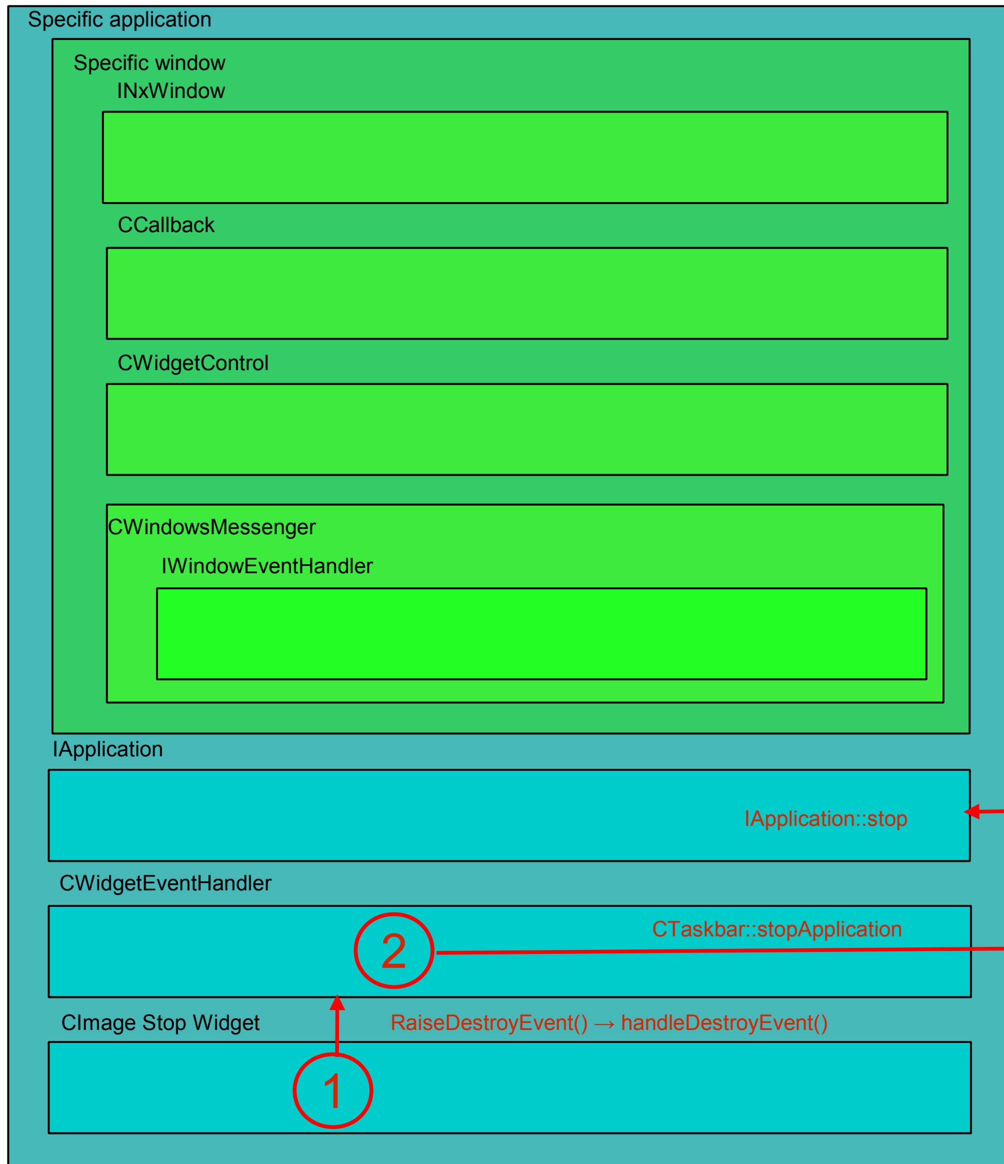
2. static CCallback method invokes a method in CWidgetControl.

3. CWidgetControl saves event related data and raises the associated event.

4. Most window events arecaught by CWindowMessenger that inherits from IWindowEventHandler. CWindowMessengeer sends message to the start window thread from further processing.

What happens in the start window thread depends on the event. For most.events, a message is sent to the start window thread for additional processing.

Window Destruction (Part 1 of 2)



Window Destruction is more complicated. Window destruction is initiated by pressed the STOP button on the toolbar.

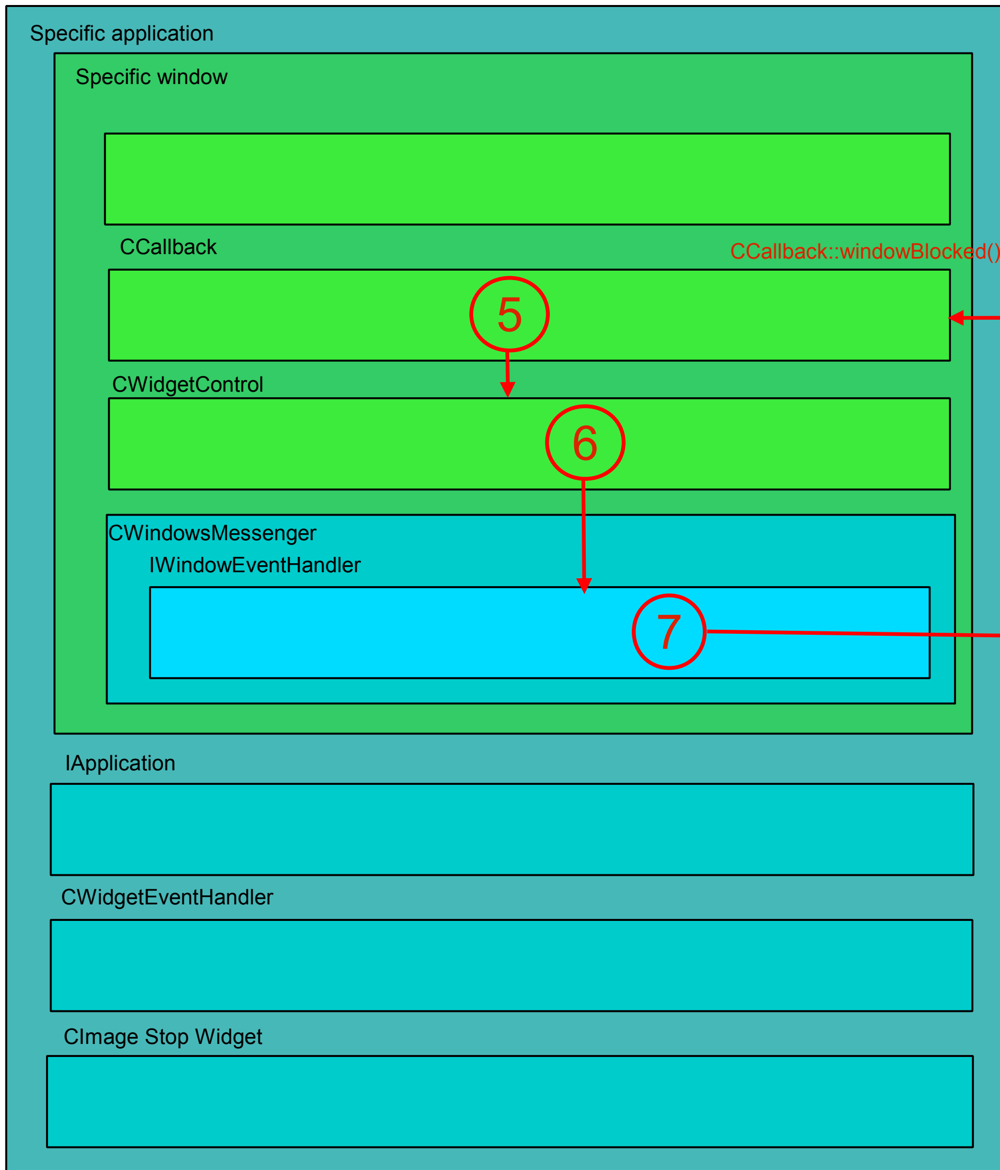
1. Widget events are raised by the CWidgetEventHandlerList that is inherited by every widget and caught by the NxWM application's CWidgetWeventHandler methods. The CWidgetWeventHandler is really part of the application's toolbar (CToolbar, but that is not illustrated)

2. The stop icon touch event causes the application tool bar's event handler to invoke the CTaskbar::stopAppliation method in the task bar.

3. The task bar then invokes(1) the application's stop() method which stops any threads and cleans-up any active resources. Then the task bar calls the nxtk_block() which will stop all communications with the window and will flush the message queues.

The application then waits for the message queues to be flushed in a deactivated/disabled state.

Window Destruction (Part 2 of 2)



The call to nxtk_block() will block sending of messages to the window. nxtk_block() will send one final message that, when received, indicates both that the communications with the window are blocked and that there are no further queued messages for the window.

4. The blocked message is received and the static CCallback::windowBlocked() method is called.

5. The static CCallback::windowBlocked() method invokes the CWidgetControl::windowBlocked() method.

6. The method does nothing except to raise the *blocked* event.

7. CWindowMessenger::handleBlockedEvent catches the *blocked* event can then sends a message to the start window window thread.

In the start window thread, the application can be safely deleted the application.