Using asynchroous callbacks safely in Xojo

Thomas Tempelmann
MBS Xojo Conference, Munich
September 7, 2018

Xojo Threads

- Xojo manages its own set of threads, consisting of the *main* thread and additional threads created in Xojo code with new or as part of a Window.
- These threads are managed by a so-called *scheduler*, which makes sure (via a mutex) that only one of these threads will run at any time.
- This ensures that one thread's code can't accidentally mess with data that's being manipulated by another thread's code, leading to data corruption and crashes.
- The scheduler will switch threads only at the end of loops (do, while, for) or when calling App.YieldToNextThread.

Foreign Threads

- Any thread created outside of Xojo (and there are always a few of those around) are unknown to Xojo’s scheduler and are therefore allowed to “run free”, in parallel (*concurrently*) to any Xojo thread.

- These foreign threads may not modify Xojo objects (including Strings and Arrays) because Xojo’s runtime may be modifying them at the same time, which would then lead to data corruption.

Callbacks into Xojo code

- Some iOS and macOS functions, available through Xojo's Declare statement, allow the passing of Xojo functions (via AddressOf operator).
- Some, like Appkit's [NSView enumerateObjectsUsingBlock:] and Foundation's [NSString enumerateLinesUsingBlock:], call us back from within. In these cases, we can assume that the callback will run on the same thread we're calling it from, which is safe.
- Others, e.g. callbacks for animation and networking, may be invoked after a delay, and often run on a different, i.e. *foreign*, thread (unless they're scheduled on the main thread, in which case they're called when the app is in idle state, i.e. is not handling an Event). We say that these callbacks are *asynchronous*.

Asynchronous callbacks

- To prevent Xojo code running on a foreign thread from causing data corruption, it may not access Xojo objects but only global (shared) properties of simple types (Integer, Boolean, Ptr but not Strings!).
- The method used for a callback must be global or shared, and it needs to prevent stackoverflow exceptions and Xojo thread switches by:

  ```
  #pragma StackOverflowChecking false
  #pragma DisableBackgroundTasks
  ```

  - Same goes for any function that is called by the callback method.
- A callback function could set a shared Boolean property or use declares to work with CoreFoundation objects on macOS.
- The Xode program needs to use a polling method (e.g. a Timer) to learn of the modifications the callback function performed.

Making callbacks safe

With the following technique, async callbacks can be made safe to access Xojo objects:

1. In the callback method, before accessing Xojo objects, a semaphore is used to suspend (sleep) the thread.

2. Once Xojo's thread scheduler is invoked to potentially switch to another Xojo thread, we know it's safe to access Object objects because no object is getting handled at that point in time by it.

3. The scheduler is suspended and the waiting foreign thread of the async callback is resumed.

4. The callback method can now safely access Xojo objects.

5. Before returning, the callback method signals the scheduler that it may resume its work.

TTsSafeCallbacks module

- **TTsSafeCallbacks.Initialize** - call this once from the main thread.
- **TTsSafeCallbacks.Enter** - call this from a foreign thread before accessing Xojo objects
- **TTsSafeCallbacks.Leave** - call this once done accessing Xojo objects and before returning from the foreign thread.
- See the *Safe threaded callbacks* project for a demo.

References

- Sample Code:
  - http://files.tempel.org/RB/MacBlockTest-simple.xojo_binary_project
  - http://files.tempel.org/RB/MacBlockTest.xojo_binary_project
  - http://files.tempel.org/RB/Threading
- Forum posts about handling pre-emptive threads
  - https://forum.xojo.com/49704
  - https://forum.xojo.com/20313 (GCD example)
- Feedback case requesting a #pragma implementing this technique
  - feedback://showreport?report_id=53200