Slate Sleeping and Active Timers

The Active Timer system allows Slate to enter a Sleep state when no UI needs to update.

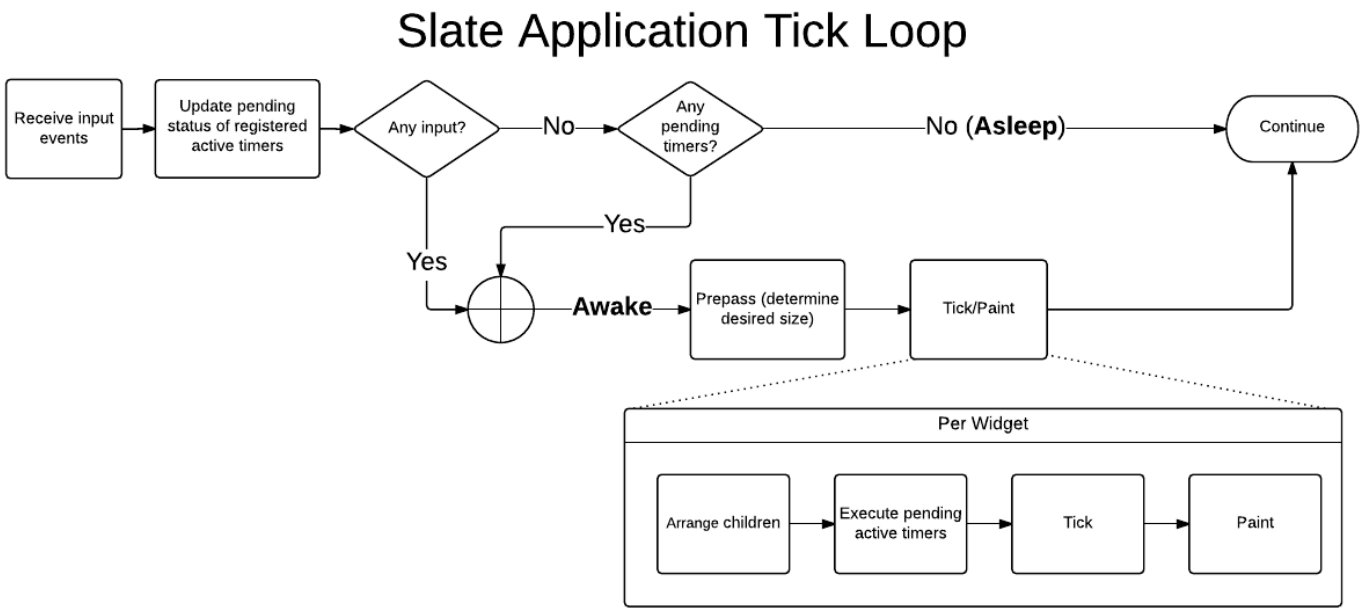
Slate is an "immediate mode" UI framework, meaning it redraws the entire UI each frame. This is great for very dynamic interfaces that are rich with graphics and animations, but results in unnecessary processor usage when nothing in the UI needs to change. Using the Active Timer system in Slate allows it to enter a Sleep state when no UI needs to update. The Active Timer feature should be utilized when working on editor UI, but not when working on UI for a game with a real-time viewport.

Slate sleeps whenever both of the following are true for a given frame:

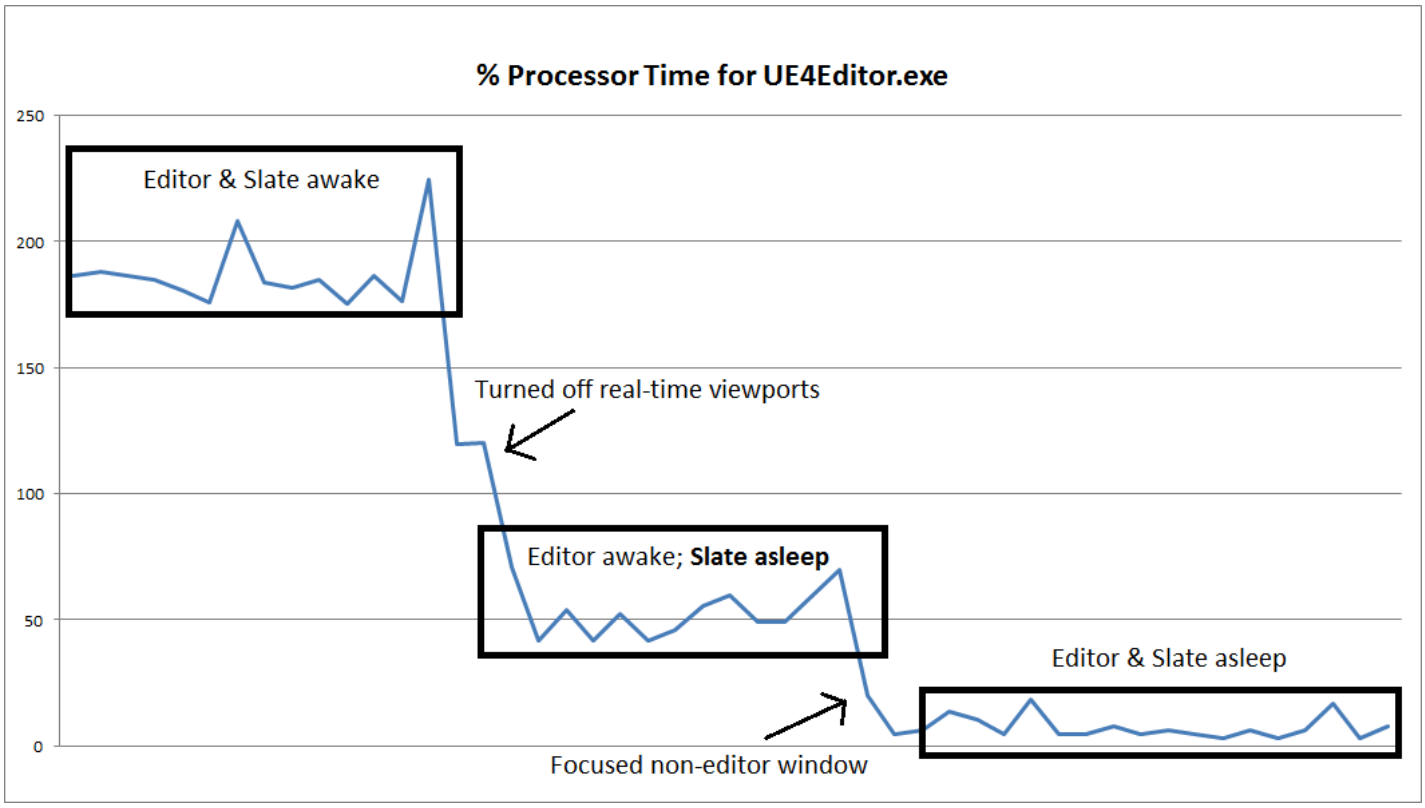
- There is no user action, and
- No active timer need to execute

A user action is any mouse movement, click, or key press.

The following diagram illustrates how the Slate application now ticks each frame:



This diagram shows how the editor's processor time changes once Slate sleeping is implemented.



Active Timers

An active timer is a delegate function explicitly registered by a widget that causes a Slate tick/paint pass upon execution, even in the absence of user action (thus the "active" nature of the timer). Active timers will continue to execute indefinitely at the frequency determined by their execution period until unregistered.

The original `Tick()` function is still present as a "passive" tick. It will be called by Slate as before, but only when Slate is awake. At some point in the future, it may be renamed to `PassiveTick()` while `Tick()` is deprecated to more clearly reflect this.

To register an active timer:

1. Define a function with the following signature: `EActiveTimerReturnType Foo(double InCurrentTime, float InDeltaTime)`
2. Bind it to an `FWidgetActiveTimerDelegate`.
3. Pass the delegate and the time period between timer executions (0 to call every frame) to `SWidget::RegisterActiveTimer()`.

There are three methods that can be used to unregister an active timer.

- Return `EActiveTimerReturnType::Stop` from the delegate.
- Pass the `FActiveTimerHandle` that was returned by `SWidget::RegisterActiveTimer()` to `SWidget::UnRegisterActiveTick()`.
- Destroy the widget that the active timer is registered under.

Currently active timers are an all-or-nothing situation, so if a single active timer needs to execute, all of Slate will be ticked. Furthermore, there is no limit to the number of active timers a widget can have registered concurrently. This can be extremely useful, but also introduces the possibility of duplicate registrations. To prevent them, track registration status in one of the following ways:

- Keep a flag in the widget to track whether the active timer is registered.
  - Search for `bIsActiveTimerRegistered` in the UE4 codebase for examples.
- Store a weak pointer to the `FActiveTimerHandle` returned by `RegisterActiveTimer()` and only register when it's invalid.
  - Search for `TWeakPtr<FActiveTimerHandle> ActiveTimerHandle` in the UE4 codebase for examples.
  - To save memory, only use this method if the active timer ever needs to be explicitly unregistered.

Common Use Cases

For the following common use cases, here are some suggested active timer setups:

- **When triggering some action:**
  - Register an active timer w/ period 0 that always returns `EActiveTimerReturnType::Stop`.
- **When performing some sort of animation or interpolation that is not controlled by an FCurveSequence:**
  1. When inertial scrolling begins, register an active timer with period 0 to update the scroll each frame.
  2. Until the destination is reached, return `EActiveTimerReturnType::Continue`.
  3. Upon reaching the target destination, return `EActiveTimerReturnType::Stop` to unregister.

  > An example of this is inertial scrolling; see `SScrollBox` for an example.

- **When taking action after a delay, like opening a different sub-menu, or docking a tab:**
  - Register with a positive non-zero delay

  > The period on an active timer cannot be changed once registered. To "reset" the delay before execution, the active timer must be explicitly unregistered and re-registered.

- **When performing an action periodically and indefinitely:**
  - Register with a positive non-zero period and just keep returning `EActiveTimerReturnType::Continue`.

FCurveSequence and Active Timers

The API of `FCurveSequence` has been updated to simplify the process of registering active ticks when a widget is being animated. Playing the sequence now requires a reference to the widget that will be animated by it. An empty active tick is automatically registered on behalf of the passed widget for the duration of the sequence playback. Whether the sequence

should loop is now specified when playing the sequence. However, doing so will register the active tick indefinitely, so use it with caution. Whether looping or not, the active tick will unregister when calling `Pause()`, `JumpToStart()`, or `JumpToEnd()`.

Testing Slate Sleeping

There are two console variables related to Slate sleeping:

- `Slate.AllowSlateToSleep` controls whether Slate can ever enter a sleep state. It is currently enabled by default.
- `Slate.SleepBufferPostInput` specifies some amount of time following the last user action that Slate will continue to stay awake. It defaults to 0. This should only be used for debugging, as it may not be a permanent feature of the sleep system.

Since the goal is to make Slate that can sleep indistinguishable from Slate that never sleeps, it can be hard to tell if Slate is sleeping. Currently, the best way to monitor this is to show the editor frame rate (**Editor Settings→Miscellaneous→Show Frame Rate and Memory**). When it freezes, Slate is sleeping.