How to manage long running Periodic Tasks

This article explains how to manage the limited time provided by a PeriodicTask on Windows Phone, ensuring that operations that may need to run across several cycles are performed safely and efficiently.

Introduction

A Windows Phone `PeriodicTask` represents a task that is executed for a small amount of time according to a regular schedule (nominally 25 seconds runtime at a minimum 30 minute interval). They are considered suitable for tasks like non-essential notifications - but may also be useful for activities where updating across a number of cycles is acceptable.

The documentation states that there are conditions where tasks can occur late (or not at all) and the execution time can be cut short. Even if the task runs to schedule and gives the expected execution time, 25 seconds is not a particularly large amount of time, and some tasks may need to run across multiple cycles in order to complete their operations (indeed even a "short" task may not complete within the allocated time if it is dependent on external factors like network connectivity).

Developers should code with the following assumptions in mind:

- Tasks may run late or not at all
- Tasks may get less time to execute
- Tasks may not be able to complete for external reasons (for example if they rely on a network connection that is not available).

If any of the above occur then an activity may need to run across multiple task cycles, possibly when the app or the device itself has been restarted. This article explains the main strategies for preserving your transient data across iterations.


Note: No code examples are provided because the actual implementation will depend on the operations being performed/data being sent.

How reliable is PeriodicTask

The documentation outlines constraints for all Scheduled Task Types and specifically for Periodic Agents. These do not provide "hard" performance figures, but indicate that the schedule can drift by as much as 10 minutes under heavy load and that in low power modes the task may not run at all. They also indicate that the task execution time can be cut short under some circumstances (although it is not stated what the circumstances are).

Testing of a Lumia 820 over a number of days indicates that the vast majority of tasks occur within seconds of the scheduled time under heavy load and/or in low power mode. The testing was also unable to cause the execution time to be cut short - this was always reliably 25 seconds.

While not exhaustive, and of course other devices might have poorer performance, this testing indicates that `PeriodicTask` is fairly reliable. It cannot be trusted where absolute reliability is required, but it is suitable where occasional unreliability can be tolerated.

Safe data strategies

Always-saved strategy

In this approach parts of the task are stored separately as a queue in `IsolatedStorage` and some or all of the queue is loaded when the task starts. As each part is confirmed to be complete it is deleted from storage (and from the queue). This ensures that operations are permanently stored until completed, so that even if the execution time is cut short or the device is reset, no data is lost. At most one "completed part" will remain in storage and need to be executed again in the next cycle.

As a "real world" example of this strategy consider the case where a periodic task is used to send data to a server. The data is stored in isolated storage and packets are sent to the server - they are deleted from `IsolatedStorage` only when an acknowledgement is received from the server. This guarantees that the data will be held safely on the device until acted on even if the device reboots, and will be sent again the next time the task gets a chance to run.

This approach is fairly efficient. At most, the device will only ever need to resend one package and the server will at most have to drop one re-sent packet. Note that size of packets should be tuned to ensure the maximum data can be sent in in each cycle.

Save-at-end strategy

The preceding strategy is robust and simple to code. However it does involve a file write in order to delete completed operations every time an operation completes - which might be a problem if these prove to be slow and your packet size mandates that you have a lot of them.

In this case you might consider a variation of the above strategy where deletion of items from the queue is batched up, or even done all at one point at the end of the task. If a task is cut short then operations might not be deleted from the queue and would have to be repeated next cycle. In practice testing shows this is rare, so this approach is in reality similarly robust to the original.

Summary

This article shows that where it is important that a task complete (at some point) the task needs to be stored in Isolated Storage, and only removed when it is complete.

Two approaches for safely saving task information are outlined. The former will require more updates to the file system but may be simpler to code.

References

This article originates from the discussion board topic Saving data before exiting in PeriodicTask