

Extended Execution Samples

*This sample is compatible with the Windows 10 Creators Update SDK (15063)*

# Description

These samples show the behavior of extended execution and the events that are related to extended execution. The samples will print to the screen and to debug output a timestamp, thread ID, event occurring (often the function name), and any additional data relevant to that event. These samples can be used to understand the behavior of extended execution and implement extended execution in your own applications.

The user has the ability to toggle whether or not output is also displayed in the Windows Notification Center using the N key, which can be useful for watching events occur when the screen is not visible and there is no debugger attached.

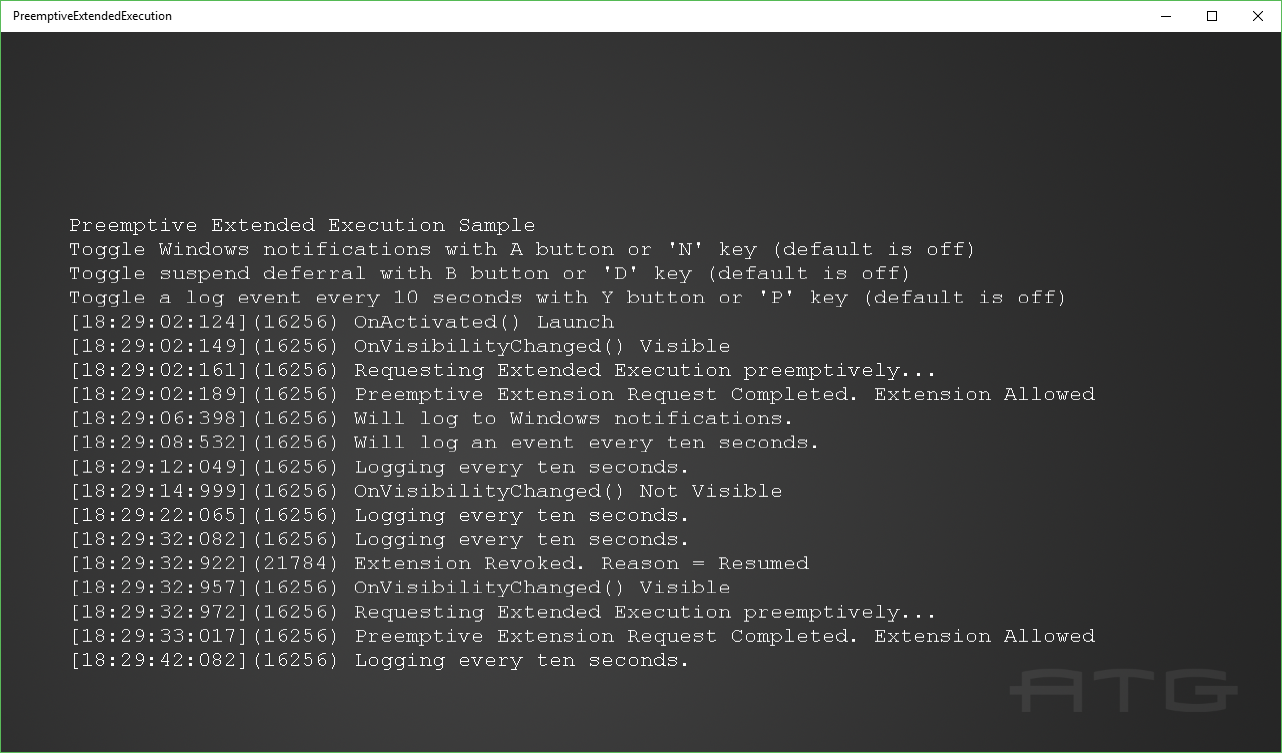
The user can also toggle whether or not the samples log an event every ten seconds. This is simply a way to demonstrate that the extended execution is working and that the app is still running unsuspended in the background.

# Building the Sample

There are no additional steps necessary for building these samples. Simply open the solution in Visual Studio, set the intended project as the Startup Project (either ExtendedExecutionOnSuspend or PreemptiveExtendedExecution), and then compile and deploy them to a Windows 10 machine to be run.

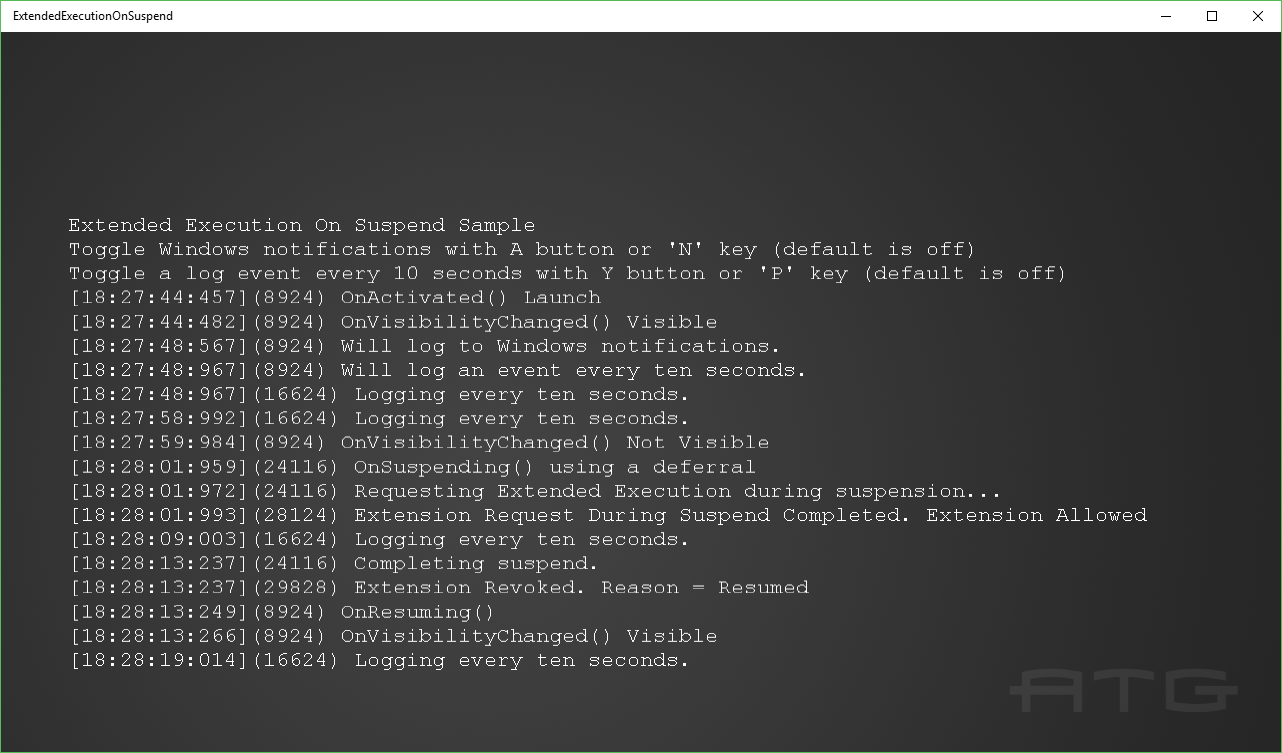
# Using the sample

## Main Screen (PreemptiveExtendedExecution)



|  |  |  |
| --- | --- | --- |
| Action | Keyboard | Xbox One Controller |
| Toggle Windows Notifications | N | A |
| Toggle suspend deferral | D | B |
| Toggle a log event every ten seconds | P | Y |
| Exit | Esc | View |

## Main Screen (ExtendedExecutionOnSuspend)



|  |  |  |
| --- | --- | --- |
| Action | Keyboard | Xbox One Controller |
| Toggle Windows Notifications | N | A |
| Toggle a log event every ten seconds | P | Y |
| Exit | Esc | View |

# Implementation notes

There are two samples for extended execution in order to demonstrate the two different ways an extended execution request can be made and how either method can be used to keep an application running in the background for up to ten minutes.

The ExtendedExecutionOnSuspend sample will request extended execution in the middle of its OnSuspending handler. When this occurs the Reason must be set to SavingData otherwise the request will fail. This sample will also always utilize a suspend deferral because we want to wait() on the result of the extended execution request so we know how to proceed with suspending and we cannot call wait() on the CoreWindow thread. If the extension is granted, then we can continue execution for up to ten minutes, however the CoreWindow thread will not return from the OnSuspending handler. When the extension is revoked, whether the reason is Resumed or SystemPolicy, the app will complete suspension but in the case of Resumed the application will be immediately Resumed. These behaviors can make it difficult to use the ‘OnSuspend’ method of requesting extended execution for some applications compared to the preemptive version.

The PreemptiveExtendedExecution sample will request an extension any time it is made visible. When making a preemptive extended execution request the Reason must be set to either LocationTracking or Unspecified otherwise the request will fail. If the extension is granted, then the next time the application would have been suspended (most commonly when the application is minimized) instead the suspend event will not occur and the application will continue to run as normal for up to ten minutes. When the application would have been resumed or needs to be suspended then the extension revoked handler will be called with a reason of either Resumed or SystemPolicy. If the reason is Resumed then the application will continue to run normally as if it were never in a state to be suspended. If the reason is SystemPolicy then the application will proceed to be suspended. After a preemptive extension request has been made and the application has been put into a state where it would have been suspended then a new extension must be requested in order to extend the next potential suspension. The PreemptiveExtendedExecution does this by requesting extended execution any time it is made visible.

In either sample, if the suspension is ever completed by the app then that will signal to the system that the application has finished with its reason for requesting an extension and will complete the suspension.

If the ExtendedExecutionSession is ever allowed to go out of scope, then any active extension is nullified and the application will suspend normally. A key detail here is that because the request is nullified this means the Revoked callback will not be called before suspending.

These samples currently log for the following events:

CoreApplicationView::Activated

CoreApplication::Suspending

CoreApplication::Resuming

CoreWindow::VisibilityChanged

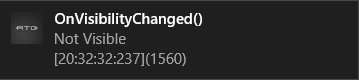
When an extension is requested

When an extension request is completed

When an extension is revoked

Logging of these events occurs in both Main.cpp and in ExtendedExecutionOnSuspend.cpp or PreemptiveExtendedExecution.cpp in the associated functions and event handlers. There are also logs for when different operations are chosen and an initial log to remind the user of the controls. One of the operations the user can choose will also log an event every ten seconds.

Both samples contain an option to log to the Windows Notification Center. When enabled any log that would normally be printed to debug output and to the app’s screen will also be logged as a notification. This notification will cause a small toast to appear in the bottom right corner of the screen by default.



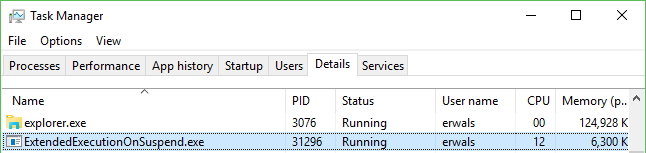
The notification center can also be opened to see a list of all notifications that have occurred.

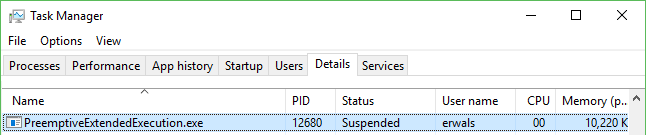


These notifications can be particularly useful to watch the events occurring while the application is not visible and while there is no debugger attached.

The option to log an event every ten seconds is simply to demonstrate that the application is still active and running while not visible.

The PLM state of an application can also be viewed using the Task Manager’s Details tab.





When testing operations related to PLM this can be useful for verifying whether an application is in the Running or Suspended state.

# Known issues

While the sample can be run with the Visual Studio debugger, for accurate testing of extended execution it is recommended to not use the debugger and instead run the application with either the “Start Without Debugging” option or launch it from the start menu. Normally an app that is minimized will be suspended within a couple of seconds however the Visual Studio debugger will prevent a minimized app from being suspended. It does this to prevent suspension from interfering with debugging, but in this specific case we want suspension to occur normally so extended execution can come into play when it normally would.

The Visual Studio Lifecycle Events menu can be used to force an app to suspend and resume however this can also be forced when the app is not in a state where it would normally suspend. For example, Visual Studio can suspend an application that is currently visible on screen. In the samples we do not keep rendering while the extended execution is active because normally the app would only suspend if it was not visible and the extended execution would be revoked and resumed if we ever became visible. This means that if Visual Studio is used to manually suspend the samples while they are visible, even though the extension request is allowed, the apps screen will not update with the logging information until the app is resumed.

It is for these reasons that it is generally recommended to be careful when using the Visual Studio debugger for testing extended execution because it is very easy to unintentionally cause the app to be in a state that would not normally occur when the app is run without the debugger leading to unintended behavior. If you do choose to debug with Visual Studio just keep in mind that the app will not suspend as it normally would when not visible and that the Lifecycle Events menu can force a suspension when it would not normally occur.

One other thing to emphasize is that extended execution will not prevent an application from suspending indefinitely. Whether an application is requesting extended execution while they are suspending or if it was requested preemptively, after ten minutes of being not visible the extension will be revoked for the reason of SystemPolicy and the app will need to suspend. It is also possible for the extension to be revoked sooner than ten minutes or for the request to be denied depending on the state of the system.

# Update history

Initial release June 2016

# Privacy statement

When compiling and running a sample, the file name of the sample executable will be sent to Microsoft to help track sample usage. To opt-out of this data collection, you can remove the block of code in Main.cpp labeled “Sample Usage Telemetry”.

For more information about Microsoft’s privacy policies in general, see the [Microsoft Privacy Statement](https://privacy.microsoft.com/en-us/privacystatement/).