What’s the guidance on when to use rundll32? Easy: Don’t use it

devblogs.microsoft.com/oldnewthing/20130104-00

January 4, 2013

Raymond Chen

Occasionally, a customer will ask, “What is Rundll32.exe and when should I use it instead of just writing a standalone exe?”

The guidance is very simple: Don’t use rundll32. Just write your standalone exe.

Rundll32 is a leftover from Windows 95, and it has been deprecated since at least Windows Vista because it violates a lot of modern engineering guidelines. If you run something via Rundll32, then you lose the ability to tailor the execution environment to the thing you’re running. Instead, the environment is set up for whatever Rundll32 requests.

- Data Execution Prevention policy cannot be applied to a specific Rundll32 command line. Any policy you set applies to all Rundll32 commands.
- Address Space Layout Randomization cannot be applied to a specific Rundll32 command line. Any policy you set applies to all Rundll32 commands.
- Application compatibility shims cannot be applied to a specific Rundll32 command line. Any application compatibilty shim you enable will be applied to all Rundll32 commands.
- SAFER policy cannot be applied to a specific Rundll32 command line. Any policy you set applies to all Rundll32 commands.
- The Description in Task Manager will be Rundll32’s description, which does not help users identify what the specific Rundll32 instance is doing.
- You cannot apply a manifest to a specific Rundll32 command line. You have to use the manifest that comes with Rundll32. (In particular, this means that your code must be high DPI aware.)
- The Fault Tolerant Heap cannot be enabled for a specific Rundll32 command line. Any policy you set applies to all Rundll32 commands.
- All Rundll32.exe applications are treated as the same program for the purpose of determining which applications are most frequently run.
- Explorer tracks various attributes of an application based on the executable name, so all Rundll32.exe commands will be treated as the same application. (For example, all windows hosted by Rundll32 will group together.)

- You won’t get any Windows Error Reporting reports for crashes in your Rundll32.exe command line, because they all got sent to the registered owner of Rundll32.exe (the Windows team).
- Many environmental settings are implied by the executable. If you use Rundll32, then those settings are not chosen by you since you didn’t control how Rundll32 configures its environment.
    - Rundll32 is marked as `TSAWARE`, so your Rundll32 command must be Terminal Services compatible.
    - Rundll32 is marked as `LARGEADDRESSAWARE`, so your Rundll32 command must be 3GB-compatible.
    - Rundll32 specifies its preferred stack reserve and commit, so you don’t control your stack size.
    - Rundll32 is marked as compatible with the version of Windows it shipped with, so it has opted into all new behaviors (even the breaking ones), such as automatically getting the `HeapEnableTerminationOnCorruption` flag set on all its heaps.
- Windows N+1 may add a new behavior that Rundll32 opts into, but which your Rundll32 command line does not support. (It can’t, because the new behavior didn’t exist at the time you wrote your Rundll32 command line.) As you can see, this has happened many times in the past (for example, high DPI, Terminal Services compatibility, 3GB compatibility), and it will certainly happen again in the future.

You get the idea.

Note also that Rundll32 assumes that the entry point you provide corresponds to a task which pumps messages, since it creates a window on your behalf and passes it as the first parameter. A common mistake is writing a Rundll32 entry point for a long-running task that does not pump messages. The result is an unresponsive window that clogs up broadcasts.

Digging deeper, one customer explained that they asked for guidance making this choice because they want to create a scheduled task that runs code inside a DLL, and they wanted to decide whether to create a Rundll32 entry point in their DLL, or whether they should just create a custom executable whose sole job is loading the DLL and calling the custom code.

By phrasing it as an either/or question, they missed the third (correct) option: Create your scheduled task with an `IComHandlerAction` that specifies a `CLSID` your DLL implements.