

ClipboardFusion Crash Dump Analysis

ClipboardFusion.exe.19912.dmp — .NET 10 application on Windows 11

The process died with a **native access violation** while copying clipboard bitmap data. The CLR turned that into a fatal `System.ExecutionEngineException`. It did **not** die from a normal managed exception.

What crashed

Process	ClipboardFusion.exe 6.3.99.106, PID 19912 , x64
Runtime	.NET 10.0.11 (coreclr.dll 10.0.1126.37416)
OS	Windows 11 (build 26200)
Faulting thread	OS 20112 (0x4E90), SOS thread 7, managed ID 9
Native exception	0xC0000005 ACCESS_VIOLATION , flags 0x1 (noncontinuable)
Fault type	Read of 0x000002475B8A1000 (page-aligned — typical “walked off the end of a mapping”)
Faulting IP	coreclr.dll+0x1A273B (0x7FFC04A2273B)
Managed exception	System.ExecutionEngineException (HRESULT 0x80131506)

The app had been up since 2026-09-06 14:27 UTC; the dump is from 2026-09-08 12:04 UTC. Trigger was the **Show Preview Popup** hotkey.

Call stack (faulting thread)

Top frames, then the WinForms / hotkey path:

```
KERNELBASE / CRT memmove (InlinedCallFrame)
System.Buffer.MemmoveInternal(ref byte, ref byte, nuint) // Buffer.cs:132
h0Q.f0u.GetClipboardDataBitmap(IntPtr, bool, string) // ClipboardFusion.dll <+426>
VLS.gLo.TL7()
tjF0.SjFx.BeginInvokeCustomVoid(...)
VLS.gLo.ShowClipboardContentsPopup()
VLS.gLo.ProcessHotKeyShowPreviewPopup(object)
xjvW.bjv1.WndProcMessageReceivedAfterMessageHandlers(...)
tjF0.SjFx.WndProc(ref Message)
Control.ControlNativeWindow.WndProc(...)
NativeWindow.Callback(...)
IL_STUB_ReversePInvoke(...)
user32!DispatchMessage
Application.LightThreadContext.FPushMessageLoop(...)
Application.ThreadContext.RunMessageLoopInner(...)
Application.Run(Form)
vE2.fEH.<>c__DisplayClass11_0.HZ6.MoveNext() // async Main
... Thread.StartHelper / ExecutionContext.RunInternal
```

Crash function: `System.Buffer.MemmoveInternal` → native `QCall memmove` in `coreclr.dll`.

App function that issued the copy: `GetClipboardDataBitmap` in `ClipboardFusion.dll` (obfuscated type `h0Q.f0u`).

`Buffer.cs:132` is the `QCall` wrapper:

```
internal static unsafe void MemmoveInternal(ref byte dest, ref byte src, nuint len)
{
    fixed (byte* pDest = &dest)
    fixed (byte* pSrc = &src)
        MemmoveInternal(pDest, pSrc, len); // native memmove in coreclr
}
```

An AV inside that QCall is treated as a **fatal runtime error**, not a catchable `AccessViolationException`.

Why it happened

Decompiled IL of `GetClipboardDataBitmap` (despite obfuscation) does this for clipboard bitmaps:

1. Open the clipboard.
2. Try **CF_DIB (8)** then **CF_DIBV5 (17)** via `GetClipboardData`.
3. `size = GlobalSize(handle)`
4. `byte[] buf = new byte[size]`
5. `Marshal.Copy(handle, buf, 0, size)` ← this is the `MemmoveInternal` on the stack
6. `FromBitmapDib(buf)`
7. **Fallback:** `GetClipboardData(CF_BITMAP=2) + FromHBitmap`

There is **no** `GlobalLock / GlobalUnlock`. `GetClipboardData` for DIB formats returns an **HGLOBAL**. `Marshal.Copy(IntPtr, byte[], ...)` treats that `IntPtr` as a **raw pointer**, not a moveable handle.

That is undefined if:

- the **HGLOBAL** is `GMEM_MOVEABLE` (normal for clipboard data) and was never locked, or
- `GlobalSize` is larger than the bytes actually mapped, so `memmove` reads off the end.

The fault address being **page-aligned** (`...1000`) matches “copy ran off a valid mapping onto the next unmapped page.”

The method has a catch that logs and optionally rethrows, but it never runs: the AV is inside the runtime QCall, so the CLR raises `ExecutionEngineException` and terminates.

Other exceptions in the dump (not the crash)

Several other threads show `System.BadImageFormatException` with message “**Bad method token**” (`HRESULT 0x8007000B`). Those objects have **no stack traces**. The assembly is heavily obfuscated (control-flow IL with invalid token type `0x46`). SOS/DAC cannot resolve those tokens. That is dump-analysis noise, not the faulting exception.

The thread that actually faulted is **only** the one with `ExecutionEngineException`.

What to fix

In `GetClipboardDataBitmap`:

1. `ptr = GlobalLock(hGlobal)` before copying; `GlobalUnlock` in a finally.
2. Do not trust `GlobalSize` alone — copy `min(GlobalSize, actual DIB size from BITMAPINFOHEADER)` (`biSizeImage + header/palette`).
3. Prefer `LockBits / GDI+` for `CF_BITMAP (HBITMAP)`, never `Marshal.Copy` on an `HBITMAP`.
4. Treat any remaining AV as uncatchable while the copy goes through `Buffer.MemmoveInternal`; validate pointers and sizes **before** the copy.

Repro path: `hotkey` → `ProcessHotKeyShowPreviewPopup` → `ShowClipboardContentsPopup` →

`GetClipboardDataBitmap` while the clipboard holds a DIB/DIBV5 (or a malformed/huge bitmap from another app).