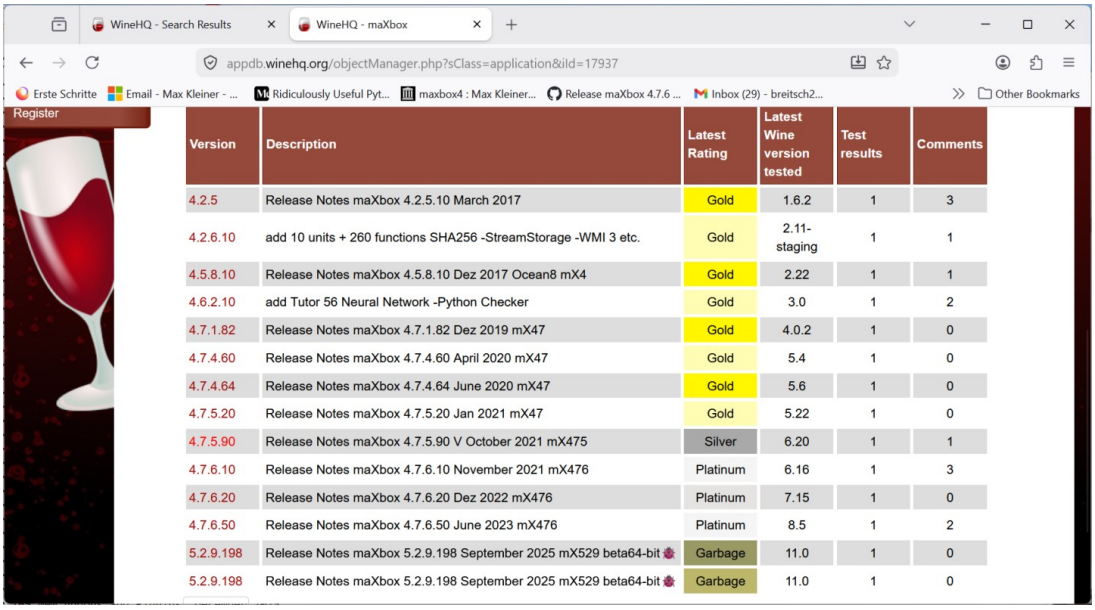
Wine Debugging

maXbox5.2 on Wine 10 or 11

As I got one page fault exception in Version 5.2 and couldn't solve the problem, so I summarised my experience to track or debug the problem, which still exists.

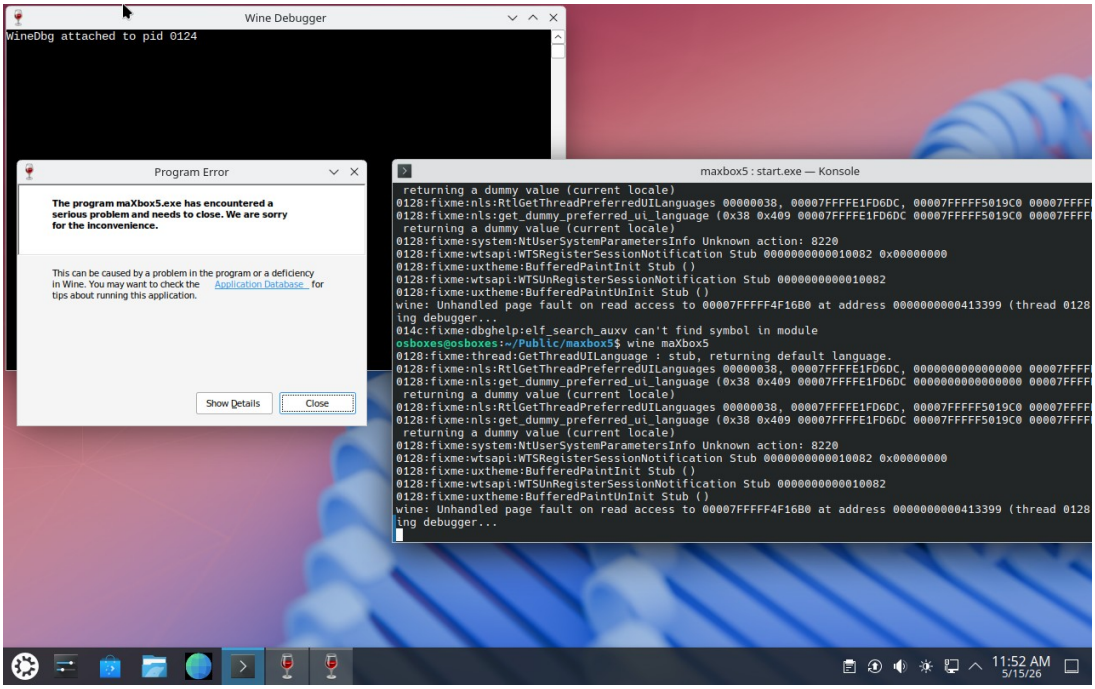
I suspect the SynEdit component which provokes an unhandled page fault to DirectWrite in the Library DirectX as a signal wine raises an exception to stop and crashes.



Version	Description	Latest Rating	Latest Wine version tested	Test results	Comments
4.2.5	Release Notes maXbox 4.2.5.10 March 2017	Gold	1.6.2	1	3
4.2.6.10	add 10 units + 260 functions SHA256 -StreamStorage -WMI 3 etc.	Gold	2.11-staging	1	1
4.5.8.10	Release Notes maXbox 4.5.8.10 Dez 2017 Ocean8 mX4	Gold	2.22	1	1
4.6.2.10	add Tutor 56 Neural Network -Python Checker	Gold	3.0	1	2
4.7.1.82	Release Notes maXbox 4.7.1.82 Dez 2019 mX47	Gold	4.0.2	1	0
4.7.4.60	Release Notes maXbox 4.7.4.60 April 2020 mX47	Gold	5.4	1	0
4.7.4.64	Release Notes maXbox 4.7.4.64 June 2020 mX47	Gold	5.6	1	0
4.7.5.20	Release Notes maXbox 4.7.5.20 Jan 2021 mX47	Gold	5.22	1	0
4.7.5.90	Release Notes maXbox 4.7.5.90 V October 2021 mX475	Silver	6.20	1	1
4.7.6.10	Release Notes maXbox 4.7.6.10 November 2021 mX476	Platinum	6.16	1	3
4.7.6.20	Release Notes maXbox 4.7.6.20 Dez 2022 mX476	Platinum	7.15	1	0
4.7.6.50	Release Notes maXbox 4.7.6.50 June 2023 mX476	Platinum	8.5	1	2
5.2.9.198	Release Notes maXbox 5.2.9.198 September 2025 mX529 beta64-bit	Garbage	11.0	1	0
5.2.9.198	Release Notes maXbox 5.2.9.198 September 2025 mX529 beta64-bit	Garbage	11.0	1	0

winehq db list

Wine is a compatibility layer that translates Windows system calls to Linux-friendly ones, allowing Windows programs to run efficiently without the overhead of a complete emulator. It's been around since 1993 and is constantly evolving. Wine is open-source, allowing contributions from the Linux community and companies like Valve (more on that later), so support for many Windows programs has gotten better with time.



Unhandled Page Fault

A Wine “page fault” is usually an `STATUS_ACCESS_VIOLATION` crash, and the right way to handle it is to capture the backtrace, identify whether the fault is in Wine or in the app, and then narrow it down with Wine’s debug logs. Wine’s own debugging guide recommends starting with `WINEDEBUG=+seh`, then using `winedbg` for a backtrace if needed, and checking for heap corruption with `warn+heap` or `+heap` when relevant.

Practical workflow

1. Run the program from a terminal so you can see the crash output.
2. Add `WINEDEBUG=+seh` to capture exception context; for example, this helps you find the first `c0000005` page fault.
3. If the app still hides the problem, attach `winedbg` and use `bt` to get a backtrace.
4. If the crash looks like memory corruption, try `WINEDEBUG=warn+heap` or `WINEDEBUG=+heap`.

What to look for

If the backtrace points into Wine code, the bug is likely in Wine or in a dependency it uses; if it points into the application, the app may be making a bad API call or dereferencing a bad pointer. A `NULL` pointer dereference often means Wine did not provide something the app expected, while crashes in `ntdll/heap.c` often mean heap corruption somewhere earlier. The Wine docs also note that `CoGetClassObject` failures followed by a null

dereference often indicate a missing COM object implementation.

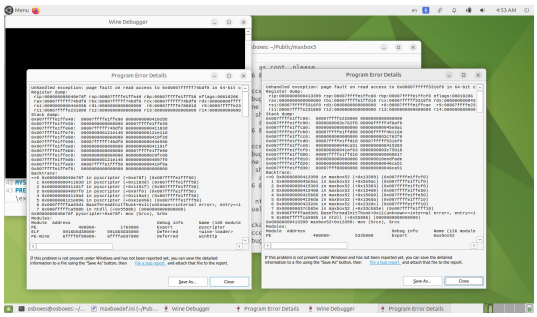
Common fixes

For many “unhandled page fault” reports, the fix is actually outside Wine: graphics-driver issues, broken GPU switching on hybrid laptops, or a bad prefix can be the real cause. In one WineHQ thread, the crash disappeared after fixing GPU/Optimus configuration, and another report linked the issue to an Intel virtual-heads Xorg config file that had to be removed.

The problem is, when I run my application from the IDE then everything works correctly, but if I try to start the application from File Explorer then I get runtime error 217 in both Debug and Release modes.

Runtime error 217 is thrown when an unhandled exception is raised before the exception handling framework is installed at startup, or after it is removed at shutdown.

So most likely the error is being caused in Initialization section of one of your units and I traced it down to the packages SynEdit in dependance to DirectWrite.



Wine Debuger at Kubuntu

Example command

A good first probe is:

```
bashWINEDEBUG=+seh wine yourapp.exe
```

If you want deeper tracing, use **`winedbg`** after the crash and grab **`bt`** for the stack trace.

PlayOnLinux

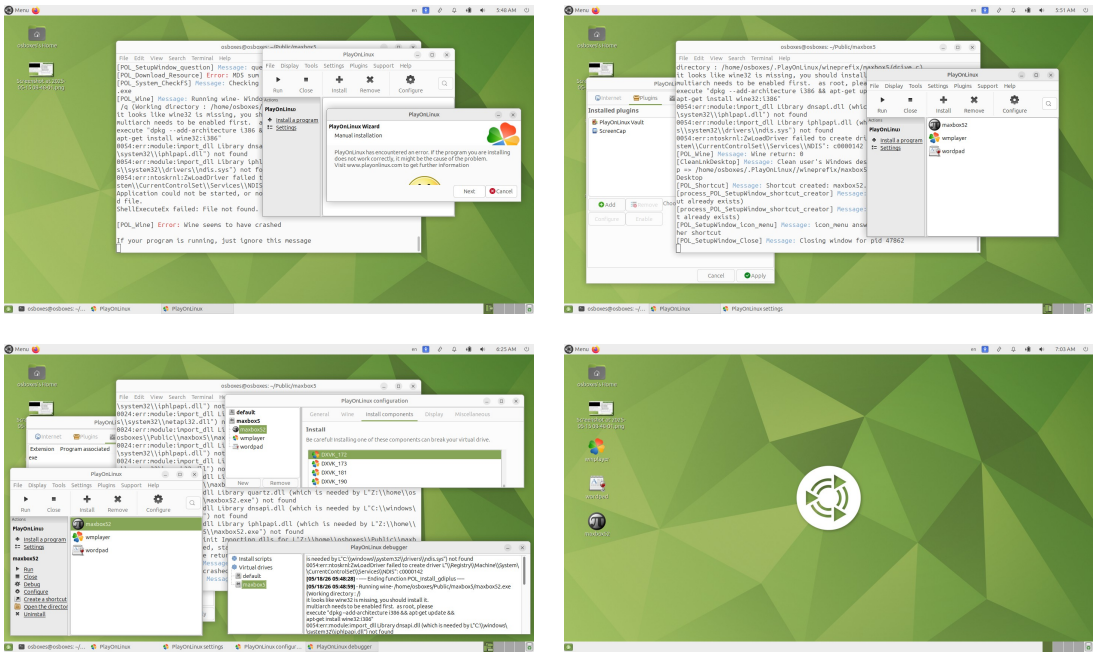
PlayOnLinux is a free and open-source program that uses Wine as its compatibility layer but provides a more user-friendly graphical interface. It

simplifies the installation and configuration of Windows applications and games on Linux. PlayOnLinux supports a wide range of applications, including Microsoft Office, Adobe Acrobat Reader, and various games.

Example of installing PlayOnLinux on Ubuntu

sudo apt update

sudo apt install playonlinux



Play on Linux Debug Session

Handle or bypassing a page fault exception

You usually **don't bypass** a Wine page fault exception; you fix the cause, because the message is just Wine reporting a crash, not something meant to be ignored. In many real cases on Linux, the fix is graphics- and Wine-stack related: Xorg/driver configuration, switching GPU mode, rebuilding the Wine prefix, or changing the Wine/DXVK version.ubuntuforums+5

What to try

- Check whether it happens with only one app or with every Wine app. If it is every app, the problem is more likely Wine, drivers, or your graphics setup than the program itself.forum.winehq+1
- Try a clean prefix: back up and remove `~/.wine`, then let Wine recreate it. Several reports mention corrupted prefixes or old configs causing repeated page faults.forum.endeavouros+1
- Test another Wine version, and if you use DXVK/Lutris/Proton-GE, try a

different build or run once without DXVK. One report fixed the issue by changing Wine/DXVK combinations.forum.endeavouros

- If you have Intel/NVIDIA/AMD graphics switching, verify the correct GPU is active and that Xorg config is present and loaded correctly. Missing or wrong Xorg device config has been linked to Wine page faults.bbs.archlinux+1
- On older setups, TLS-related Xorg settings were a fix for Wine crashes; one report resolved it with `Option "UseFastTLS" "2"` in `xorg.conf`.ubuntuforums

When a “bypass” makes sense

If by “bypass” you mean “keep the app running even though Wine crashes,” that is generally not practical for a page fault. The closest alternatives are debugging with `winedbg` or `gdb`, or isolating the exact module causing the crash so you can replace or disable that component.stackoverflow+1

Practical next step

The fastest useful approach is: create a fresh Wine prefix, try a different Wine build, and confirm your GPU/Xorg setup. If the problem is specific to one application, the likely fix is app-specific compatibility settings rather than a global bypass.forum.winehq+2

You debug Wine crashes by using winedbg (Wine’s built-in debugger) for simple Windows-style debugging, or by attaching gdb to the Wine process for deeper low-level inspection. The key difficulty is that Wine’s loader can complicate things; a common workaround is to set `WINELOADERNOEXEC=1` so gdb sees symbols correctly.ubuntuforums

1. Using `winedbg` (Wine’s debugger)

Basic usage

Run the program directly under winedbg:

```
bashwinedbg your_app.exe
```

or with a custom prefix:

```
bashWINEPREFIX=~/mywineprefix winedbg your_app.exe
```

Once inside winedbg:

- `c` – continue execution
- `s` – step into next instruction
- `n` – step over
- `bt` – backtrace (show call stack)
- `info registers` – show registers
- `list` – show source (if available)
- `break function` – set breakpoint, e.g. `break CreateFileA`
- `run` – start the programwinehq+1

Important: winedbg requires you to set breakpoints *before* the crash happens if you want to stop early. If you just run the program without breakpoints, it may exit immediately and you only see the crash state.winehq

For built-in tools:

```
bashwinedbg notepad.exe
```

works even if your own exe crashes before winedbg can attach.winehq

When winedbg isn't enough

- Some crashes happen during process startup, before you can set breakpoints.
- winedbg may not reliably break before the entry point in all cases.news.ycombinator+1
- For complex crashes (e.g. page faults, stack corruption), gdb is usually more powerful.

---

2. Using gdb to debug Wine apps

A. Simple approach: attach to a running Wine process

1. Start your Wine app normally:bash`wine your_app.exe &`
2. Find its PID:bash`ps aux | grep your_app.exe`
3. Attach gdb:bash`gdb -p <PID>`

Inside gdb:

- `bt` – backtrace
- `info registers`
- `frame N` – select stack frame

- `print var` – inspect variables
- `continue` – resumereddit

This is good for investigating a crash that already happened, but you can't easily set breakpoints before the crash.

B. Better: run Wine itself under gdb (with `WINELOADERNOEXEC=1`)

This is the most robust method for debugging startup crashes and page faults.

1. Set the environment variable to prevent Wine from re-executing its preloader:bash`export WINELOADERNOEXEC=1`
2. Start gdb with Wine and your program:bash`gdb --args wine your_app.exe`Or with a custom prefix:bash`WINEPREFIX=~/mywineprefix gdb --args wine your_app.exe`
3. In gdb, set up fork/exec behavior and run:text`set follow-fork-mode child set follow-exec-mode new run`If you see fork/exec catchpoints, you can use:text`catch fork catch exec continue`until you're in your actual program.ubuntuforums
4. Set breakpoints:text`break CreateFileA break RtlAnsiStringToUnicodeString break your_dll!YourFunction`Or by address if symbols aren't available:text`break *0x7bc714dd`
5. Run and inspect on crash:text`run # When it crashes: bt info registers frame N print var`

With `WINELOADERNOEXEC=1`, gdb can see Wine DLL symbols (like `ntdll.dll`, `kernelbase.dll`) and produce useful backtraces instead of just `??` frames.ubuntuforums

---

C. Adding Wine DLL symbols manually (if needed)

If gdb doesn't automatically resolve Wine DLL functions:

1. Find the PID of the Wine process.
2. Check `/proc/<PID>/maps` to find where `ntdll.dll.so`, `kernelbase.dll.so`, etc. are mapped:bash`cat /proc/<PID>/maps |`

`grep ntdll`

3. In gdb, load symbols at the mapped base address:text`add-symbol-file /usr/lib/wine/ntdll.dll.so 0x7bc00000 add-symbol-file /usr/lib/wine/kernelbase.dll.so 0x7b000000`Adjust paths and addresses to your system.ubuntuforums

---

3. Debugging a page fault / access violation

When Wine reports something like:

Unhandled exception: page fault on read/access/execute

1. Run under gdb as above.
2. When it crashes, immediately do:text`bt info registers info args`
3. Look at:
   - The instruction pointer (`eip`/`rip`)
   - The faulting address (often in registers or in the signal info)
   - Which Wine DLL or your app's module it's instackoverflow+2

This often reveals:

- Corrupted stack / invalid pointer
- Bad module load address
- GPU/driver-related code path (e.g. shader compiler, DXVK)

You can then:

- Try a fresh Wine prefix (`rm -rf ~/.wine` and recreate).
- Test different Wine / DXVK versions.
- Disable graphics features (hardware acceleration, etc.) to isolate the crash path.

---

4. Practical workflow summary

- For quick checks:
  `winedbg your_app.exe` → set breakpoints → `c` → `bt` on crash.
- For real crash investigation:
  `WINELOADERNOEXEC=1 gdb --args wine your_app.exe`
  → `set follow-fork-mode child` → `run` → `bt` on crash.
- If every Wine app crashes, the issue is likely Wine/drivers/Xorg, not the app; focus on prefix, Wine version, and graphics config instead of just

debugging.forum.winehq+1

If you tell me your Linux distro, Wine version, and whether the crash is 32-bit or 64-bit, I can give exact commands tailored to your setup.

On Ubuntu 24.04 with Wine 9.0 crashing on 64-bit apps (often with "page fault" errors), the most effective approach is to **debug with gdb under `WINELOADERNOEXEC=1`** and then use what you find to decide whether to **switch Wine versions/branches, fix graphics drivers, or recreate the prefix**.

Below is a tailored, step-by-step guide.

---

1. Prepare your system for debugging

1.1. Enable 32-bit architecture (required for full Wine on 64-bit)

Even if you only run 64-bit apps, Wine on Ubuntu needs multiarch:

```
bashsudo dpkg --add-architecture i386
sudo apt update
```

1.2. Install gdb and Wine debug tools

```
bashsudo apt install gdb wine64 wine32:i386 winehq-stable   # if you want WineHQ stable
```

If you're using the Ubuntu repo Wine (9.0~repack), that's already Wine 9.0 stable; you can also install **`winehq-staging`** to test a different build side-by-side.reddit+1

---

2. Debug with `winedbg` (quick checks)

2.1. Run the app under winedbg

```
bashWINEPREFIX=~/wine64prefix winedbg your_app.exe
```

If winedbg starts but the app crashes immediately:

- Inside winedbg:text**`bt info registers`**
- Try setting a breakpoint before running:text**`break NtCreateFile c`**

winedbg can act as a drop-in Dr. Watson and show basic crash

info.manpages.ubuntu

Limitation: For startup crashes and page faults, **winedbg** often isn't enough; **gdb** is more powerful.

---

3. Best method: run Wine under gdb with WINELOADERNOEXEC=1

This is the most reliable way to get a real backtrace on a Wine 9.0 64-bit crash under Ubuntu 24.04.

3.1. Set the environment variable

```
bashexport WINELOADERNOEXEC=1
```

This prevents Wine from re-executing its preloader, which otherwise breaks gdb symbol resolution.ubuntuforums

3.2. Start gdb with Wine and your app

```
bashgdb --args wine64 your_app.exe
```

Or with a custom prefix:

```
bashWINEPREFIX=~/wine64prefix gdb --args wine64 your_app.exe
```

Inside gdb:

```
textset follow-fork-mode child
set follow-exec-mode new
run
```

If you see fork/exec catchpoints:

```
textcatch fork
catch exec
continue
```

Keep continuing until you're in your actual app and it crashes.

3.3. When it crashes

Immediately run:

```
textbt full
```

```
info registers
info args
frame 0
```

Look for:

- The instruction pointer (`rip`)
- The faulting address (often in registers or in the signal message)
- Which module is crashing: your app's EXE/DLL or a Wine DLL (`ntdll.dll.so`, `kernelbase.dll.so`, `wow64.dll`, etc.)

This backtrace will show whether the crash is:

- In Wine itself (e.g., `ntdll`, `kernelbase`, `wow64`)
- In your app's code
- In a graphics/directx path (DXVK, Vulkan, etc.)

---

4. Common causes on Ubuntu 24.04 + Wine 9.0 64-bit

Based on similar reports with Wine 9.0 and page faults:

4.1. Corrupted or incompatible Wine prefix

Many users fix page faults by removing and recreating the prefix:

```
bashmv ~/.wine ~/.wine.bak
wine64 your_app.exe   # creates a new prefix
```

If you use a custom prefix:

```
bashrm -rf ~/mywineprefix
WINEPREFIX=~/mywineprefix wine64 your_app.exe
```

Then test again.

4.2. Wine version / branch issues

Some apps crash on Wine 9.0 stable but work on:

- `winehq-staging`, or
- a newer Wine (e.g., 9.x development / 10.x), or
- Proton via Steam

You can install Wine Staging side-by-side:

```
bashsudo apt install --install-recommends winehq-staging
```

Then test:

```
bashWINEPREFIX=~/wine-staging-prefix /opt/wine-staging/bin/
wine your_app.exe
```

One user reported that a modern game that crashed on Wine 9.0 with page faults worked well on Wine 10.0 with a clean 64-bit prefix.crateentertainment

4.3. Graphics / DXVK / Wayland issues

Page faults on “execute access” often tie to GPU/driver/DXVK:

- If you’re on **Wayland**, try **X11**:
  - At login, choose “Ubuntu on Xorg”.
- If you use **DXVK**, try:
  - Disabling DXVK, or
  - Switching DXVK versions.
- Update your GPU drivers:
  - NVIDIA: `sudo ubuntu-drivers install` or use the official driver PPA.
  - Intel/AMD: ensure you have the latest mesa drivers.

Multiple reports link Wine crashes to GPU driver issues, especially with Intel Arc and hybrid graphics.discourse.ubuntu+1

4.4. Missing wine32 / multiarch issues

On Ubuntu 24.04, Wine 9.0 often complains:

it looks like wine32 is missing, you should install it.

Even for 64-bit apps, a full Wine install needs `wine32:i386`:

```
bashsudo apt-get install wine32:i386 wine64
```

Otherwise, some apps may crash or misbehave.winehq+1

---

5. Using the backtrace to decide next steps

After you get the gdb backtrace:

- If the crash is in Wine DLLs (`ntdll`, `kernelbase`, `wow64`):
  - Try a different Wine branch/version (stable → staging, or newer Wine).
  - Recreate the prefix.
- If the crash is in your app's code:
  - Check for app patches, compatibility settings, or specific DLL overrides.
  - Use `winetricks` to install missing runtimes (e.g., `vcrun`, `dotnet`) if relevant.
- If the crash is in graphics/directx paths:
  - Disable DXVK or test without hardware acceleration.
  - Update GPU drivers and try X11 instead of Wayland.

---

6. Minimal working recipe for your case

Given Ubuntu 24.04 + Wine 9.0 64-bit crashes:

1. Ensure multiarch and wine32:bash`sudo dpkg --add-architecture i386 sudo apt update sudo apt install wine32:i386 wine64 gdb`
2. Try a clean 64-bit prefix:bash`mv ~/.wine ~/.wine.bak wine64 your_app.exe`
3. If it still crashes, debug with gdb:bash`export WINELOADERNOEXEC=1 gdb --args wine64 your_app.exe`Inside gdb:text`set follow-fork-mode child set follow-exec-mode new run # on crash: bt full info registers`
4. If the backtrace shows Wine/graphics issues:
   - Install Wine Staging and test:bash`sudo apt install --install-recommends winehq-staging WINEPREFIX=~/wine-staging-prefix /opt/wine-staging/bin/wine your_app.exe`
   - Or test with Steam Proton if it's a game.

If you tell me in maXbox:

- the exact app/game name,

- whether it's a game or a regular Windows app,
- and whether you're on NVIDIA/Intel/AMD graphics,

I can give you a precise set of commands and configuration changes (including specific DLL overrides and DXVK settings) for your case.