WPF Client overlay limitations (HWND-hosted controls)

Why a few WPF Client controls always paint above overlapping WPF visuals, and how to lay out displays that include them.

Symptom

On the WPF Client (RichClient and SmartClient), a small number of controls always paint *above* any WPF visual whose bounding rectangle overlaps them, regardless of `Panel.ZIndex`, `Canvas.ZIndex`, or sibling stacking order. Typical visible effects:

- A button, image, or shape placed on top of the control disappears behind it — only the affected control is visible in the overlapping region.
- A popup, tooltip, drop-down, menu, or context menu that opens over the control is cropped or hidden where it overlaps.
- The display header, footer, or page scrollbar is obscured where the control's rectangle reaches into them.
- A semi-transparent WPF overlay drawn across the display passes *through* the control as if it were not there.

The effect is consistent and reproducible — it is not a timing or rendering bug.

Why it happens

The affected controls are hosted in their own native Win32 window (an `HWND`) that sits on top of the WPF window's composition surface. WPF and Win32 use two different rendering pipelines, and a top-level HWND child always paints above the WPF compositor. WPF cannot draw over it, regardless of element order in the visual tree or any Z-index property.

This is a documented Windows + WPF interop constraint commonly called the *WPF Airspace Problem*. It applies to every WPF application that hosts native Win32 content — it is not specific to FrameworX or to any one control. The HTML5/Web Client uses a different rendering model (everything is HTML/CSS in one browser document) and is **not** affected by this constraint.

Affected controls

Three controls in the FrameworX library host native HWND content on the WPF Client and are subject to the airspace constraint:

- WebBrowser Control Reference — hosts the WebView2 (Chromium) browser engine.
- MapsGMap Control Reference — hosts the GMap.NET WinForms map control.
- WPF Control Reference — hosts a customer-supplied WPF/WinForms control loaded from a .DLL. Whether the airspace constraint applies depends on whether the embedded control itself uses an HWND host (e.g. WindowsFormsHost, WebBrowser, or any control wrapping a native window). Pure managed WPF controls loaded through this surface are not affected.

No other display control in the FrameworX library is subject to this limitation today.

Mitigations

The airspace constraint cannot be lifted at the FrameworX layer — the fix would require either rewriting the underlying engine to participate in WPF composition (not always possible) or adopting an experimental Windows API that the engine vendors do not yet officially support. The supported approach is to lay out displays so that WPF content does not need to overlap these controls. Practical patterns:

- **Keep WPF overlays outside the control's rectangle.** Place buttons, labels, status indicators, and other WPF visuals in a region adjacent to the control, not on top of it. The display header and footer should not extend over the control's rectangle.
- **Reserve the full control area for the embedded content.** If the embedded web page, map, or custom control needs to expose interactive UI of its own, build that UI *inside* the embedded content (HTML buttons inside the WebBrowser; map overlays inside MapsGMap) where it renders in the same compositing surface as the rest of the control.
- **Target the HTML5 Client when overlay UI is required across both runtimes.** The HTML5 Client renders the WebBrowser via an HTML iframe and the other affected controls via their browser-native equivalents, all in one CSS-stacked document. Layouts that rely on WPF overlays over these controls work correctly on the HTML5 Client.
- **Do not raise tickets for individual overlap cases.** The constraint is a Windows interop limitation, not a per-control defect. A ticket reporting "popup is cropped over the WebBrowser" or "header disappears behind the map" will be closed as out of scope with a pointer to this page.

Further reading

- Technology regions overview (Microsoft Learn) — Microsoft's overview of the WPF/Win32 interop constraint that gives this limitation its common name.
- WPF and Win32 interoperation (Microsoft Learn) — reference for how the two rendering pipelines coexist in a hybrid application.

In this section...