# Invalidation Box

An overview of Invalidation Box, which helps developers optimize UI Widgets.

**PREREQUISITE TOPICS**

In order to understand and use the content on this page, make sure you are familiar with the following topics:

- Widget Type Reference

# Description

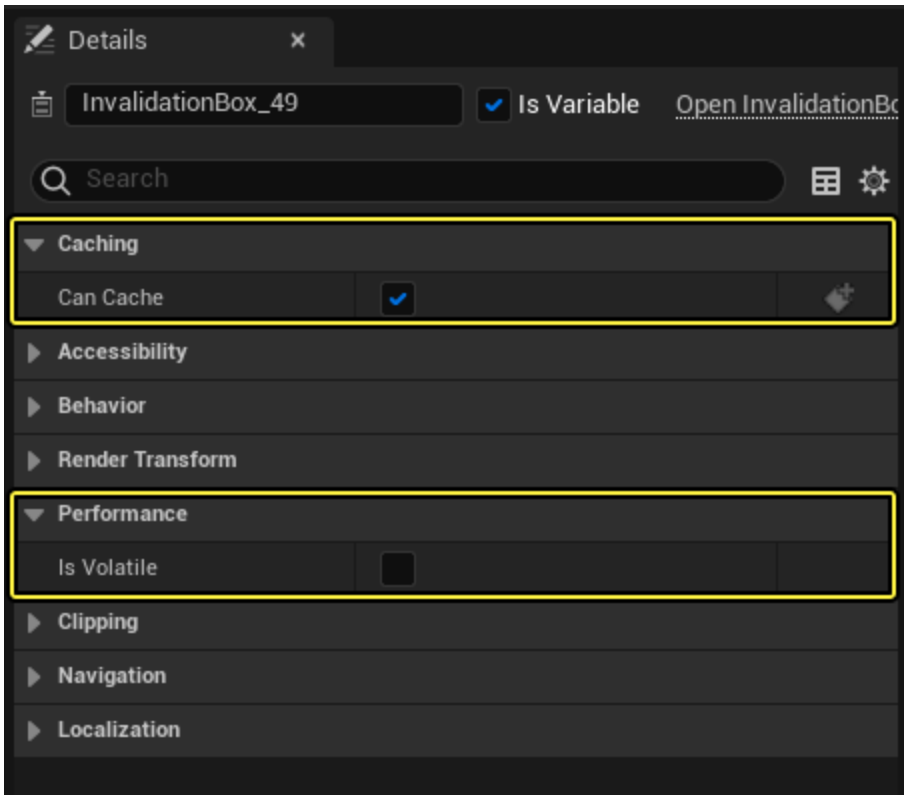
Widgets that are wrapped with an **Invalidation Box** allows the child widget geometry to be cached to speed up Slate rendering. Any widgets that are cached by an Invalidation Box are not pre-passed, ticked, or painted. In general, if you are looking to optimize your project, wrapping certain widgets with Invalidation Boxes may boost your performance (particularly for mobile projects or complicated UI displays). For widgets that do not change constantly, they can be placed inside an Invalidation Box and cached instead of considered during paint, tick, or prepass.

If the widget changes, however, it will become invalid and you will need to manually invalidate the cache which will throw it away essentially and force it to redraw on the next paint pass.

Anything that changes the visual appearance of the widget requires it to be invalidated. The only exception to this is a change to the appearance that is not stored in the vertex buffer for those widgets (for example Materials, as changing a Material Parameter does not require invalidating the widget).

# Details

In the **Details** panel for a placed **Invalidation Box**, there are a couple of specific options that can be set that pertain to the widget:




| Option | Description |
| --- | --- |
| **Cache Relative Transforms** | Caches the locations for child draw elements relative to the invalidation box which adds extra overhead to drawing them every frame. However, in cases where the position of the invalidation boxes changes every frame, this can be a big saving. |
| **Is Volatile** | If true, prevents the widget or its child's geometry or layout information from being cached. If this widget changes every frame, but you want it to still be in an invalidation panel, you should make it as volatile instead of invalidating it every frame, |

| Option | Description |
| --- | --- |
| | which would prevent the invalidation panel from actually ever caching anything. |

Regarding the **Is Volatile** check box, any widget can be set to be volatile. Volatile widgets act like normal Slate widgets pre-invalidation. They're redrawn every frame, including all their children. When combined with the invalidation panel, it allows you to care only about redrawing the most dynamic bits of the UI, as invalidating a whole panel could be far more costly.

# Functions

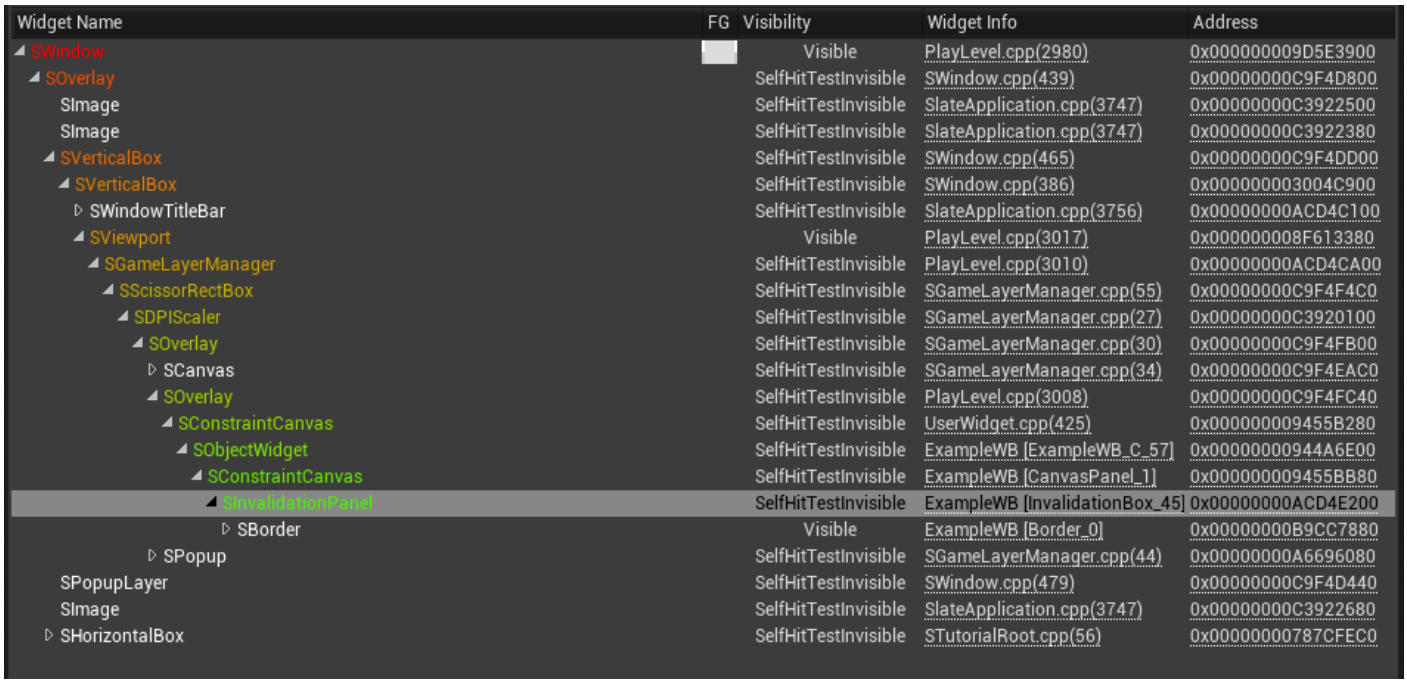
When using an **Invalidation Box**, it is up to the user to call `Invalidate` through C++ or the **Invalidate Layout And Volatility** node (pictured below) on a child widget to force invalidation.


> Currently, some core widgets do this automatically when certain properties are changed, however more will do it over time.

# Debugging

You can debug your Invalidation Boxes using the **Widget Reflector** (CTRL+Shift+W) and clicking the **Invalidation Debugging** toggle.

To display the legend when `SlateDebugger.Invalidate.Enable 1` is invoked, use `SlateDebugger.Invalidate.ToggleLegend`.




*Click image for full view.*

With the Widget Reflector up and Invalidation Debugging on, you will see the following colors:

| Color | Description |
| --- | --- |
| **Yellow** | Paint invalidated this frame. |
| **Gray** | Volatility invalidated this frame. |
| **Cyan** | ChildOrder invalidated this frame. |
| **Black** | RenderTransform invalidated this frame. |
| **White** | Visibility invalidated this frame. |
| **Pink** | Layout invalidated this frame for Invalidation Box. |
| **Blue** | Every widget was reordered. |

| Color | Description |
| --- | --- |
| **Red** | Fully re-updated. |

To debug InvalidationBox behavior, use `SlateDebugger.InvalidationRoot.Enable`.

Example below shows an image that is marked as volatile inside of a Border which is wrapped with an Invalidation Box. Since the image is marked as volatile, it can be updated dynamically during gameplay while the Border (or any other art assets that you wanted to appear around the image that do not change) is cached.