Getting Started 🚀

Welcome to our **Bliss Tutorial**! In this guide, we'll walk you through setting up your graphics device, creating a window, and initializing key components for your game or application. Let's dive in! 😀

Configure Graphics Device Options 🎨

These settings optimize the rendering process by configuring your graphics device. You can adjust parameters like debugging, swapchain behavior, and resource binding for improved performance.

```
GraphicsDeviceOptions options = new GraphicsDeviceOptions() {
    Debug = false, // Disable debug mode for better performance.
    HasMainSwapchain = true, // Enable the main swapchain.
    SwapchainDepthFormat = PixelFormat.D32FloatS8UInt, // Depth format.
    SyncToVerticalBlank = this.Settings.VSync, // Enable VSync if set in settings.
    ResourceBindingModel = ResourceBindingModel.Improved, // Use the improved resource
binding model.
    PreferDepthRangeZeroToOne = true, // Use a depth range from 0 to 1.
    PreferStandardClipSpaceYDirection = true, // Use standard clip space Y direction.
    SwapchainSrgbFormat = false // Disable sRGB format for the swapchain.
};
```

Creating Window

Now, we'll create a window with a title, specified dimensions, and a resizable state. This window will serve as the canvas for your rendering.

```
Window window = Window.CreateWindow(
    WindowType.Sdl3,          // Using SDL3 as the window type.
    1280,                     // Width in pixels.
    720,                      // Height in pixels.
    "Hello World!",           // Window title.
    WindowState.Resizable,    // Window can be resized.
    options,                  // Graphics device options from the previous step.
    Window.GetPlatformDefaultBackend(), // Platform-specific backend.
    out GraphicsDevice graphicsDevice   // Output graphics device.
);
```

Handling Window Resize 📏

When the window is resized, it's important to adjust the graphics device's swapchain accordingly. This method will be called whenever the window's dimensions change.

```
protected virtual void OnResize(Rectangle rectangle) {
    this.GraphicsDevice.MainSwapchain.Resize((uint) rectangle.Width,
(uint) rectangle.Height);
}
```

Initialize Essential Components ⚙️

This section sets up the core components of your application, including the command list, global resources, and input handling. It also demonstrates a simple game loop.

```
// Initialize command list.
CommandList commandList = graphicsDevice.ResourceFactory.CreateCommandList();

// Initialize global resources.
GlobalResource.Init(graphicsDevice);

// Initialize input.
if (window is Sdl3Window) {
    Input.Init(new Sdl3InputContext(window));
} else {
    throw new Exception("This type of window is not supported by the InputContext!");
}

// Run game loop.
while (window.Exists) {
    window.PumpEvents();  // Process window events.
    Input.Begin();        // Start input processing.

    if (!this.MainWindow.Exists) {
        break;  // Exit loop if the main window no longer exists.
    }

    this.Update();        // Update game logic.
    this.Draw(graphicsDevice, commandList); // Render the frame.
}

Logger.Warn("Application shuts down!");
this.OnClose();
```

The Update Method 🔄

Place your game logic or update routines here. This method is called every frame before drawing.

```
protected virtual void Update() {
    // Insert your update logic here (e.g., game state updates, physics, etc.).
}
```

The Draw Method 🎥

This method handles the rendering of your frame. It starts by preparing the command list, clearing the screen, executing your drawing commands, and finally submitting the commands to the graphics device.

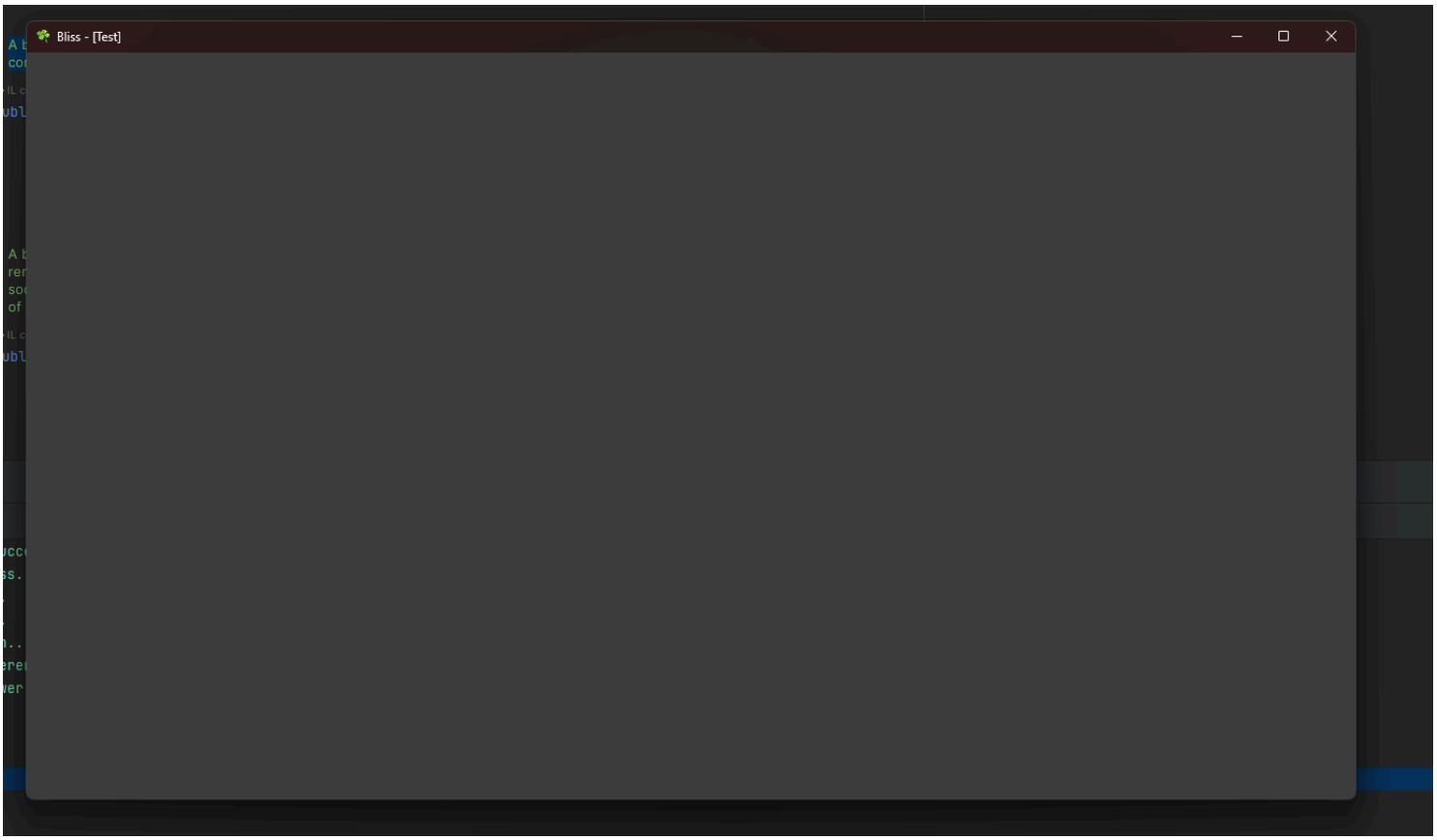
```
protected virtual void Draw(GraphicsDevice graphicsDevice, CommandList commandList) {
    commandList.Begin();  // Start recording commands.
    commandList.SetFramebuffer(graphicsDevice.SwapchainFramebuffer);
    commandList.ClearColorTarget(0, Color.DarkGray.ToRgbaFloat()); // Clear the screen with
a dark gray color.
    commandList.ClearDepthStencil(1.0F);

    // Insert your draw calls here...
    // For example: SpriteBatch, PrimitiveBatch, Mesh rendering, ImmediateRenderer, etc.

    commandList.End();  // Finish recording commands.
    graphicsDevice.WaitForIdle(); // A blocking method that returns when all submitted
CommandList objects have fully completed.
    graphicsDevice.SubmitCommands(commandList); // Submit the commands for execution.
    graphicsDevice.SwapBuffers(); // Present the rendered frame.
}
```

Launch Your Program 🚀

Your application is now ready to run! For a visual overview, check out the image below:


Bliss - [Test]

Namespace Bliss.CSharp

Classes

Disposable

GlobalResource