Getting Started

Installation - WIP :^)

- Download and install the VSGlobal mod from the ModDB.
- Add the following package reference to your .csproj file and make sure it is exclude it from the build process.

```
<PackageReference Include="VSGlobal" Version="1.0.0">
    <Private>true</Private>
    <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Once that's done, you're ready to work with the VSGlobal API. Here's an example of what a simple mod may look like:

```
public class MyModSystem : ModSystem
{
    string channel = "my_string_channel";

    public override void StartClientSide(ICoreClientAPI api)
    {
        // Add our custom payload handler that is called whenever we receive
a packet.
        // We can even say `+= async (e)` here and await a long running task.
        Events.OnPayloadReceived += (e) =>
        {
            if (e.payload.Module == channel)
            {
                string message = e.payload.DeserializePacket<string>() ?? "VSG:
Couldn't parse message!";
                Console.WriteLine(message);
                // If we wanted to do anything in the game though, we'll need to be
on the main thread!
                api.Events.EnqueueMainThreadTask(() =>
api.ShowChatMessage($"Received a payload: {message}"), "MTT_MyModPayloadReceived");
            }
        };

        // Add our custom event for when VSGlobal connects (VSGlobal loads at
level 0)
```

```
        Events.OnConnect += async (args) =>
        {
            await Network.Subscribe(channel);
            // I'm sure this can be made async but.... Time is not a luxury I have.
            this.api.Event.OnSendChatMessage += MyChatMessageHandler;
        };

        base.StartClientSide(this.api);
    }

    private void MyChatMessageHandler(int groupId, ref string message, ref
EnumHandling handled)
    {
        // All this can be called `async` and is awaitable. I highly recommend you
do this to enhance performance.
        Task.Run(async () => await Network.Broadcast($"Message from VSGlobal's
server! - {threadedMessage}", channel));
        // handled = EnumHandling.PreventSubsequent; // If we wanted to brick all
native functionality and just test, this would be the way to do it.
    }
}
```

Keep in mind, we can replace giving it a string value for any type we'd like to send. We can send native Vintage Story Protobuf packets, we can send structs, classes and potentially even functions/tasks we can invoke once it's received.

```
// Sending large data is possible, but not entirely recommended. Bigger packets,
bigger cost. Keep it simple if possible.
Broadcast(api.World.Player, "my_channel") //> OnPayloadReceived -
e.payload.DeserializePacket<IPlayer>() == Their player.

// Untested use case but probably possible.
Broadcast(EnumThingo.Grundle, "my_channel") //> OnPayloadReceived -
e.payload.DeserializePacket<EnumThingo>() == EnumThingo.Grundle

// Untested use case but probably possible.
Broadcast(()=>{ return "Someone test this for me, please! - Lila"; }, "my_channel");
//> OnPayloadReceived - e.payload.DeserializePacket<??????>().Invoke();
// The list goes on. It's generic; go apeshit.
```

Now you're familiar with the basics, you can look into the detailed API docs!

Namespace VSGlobal

Classes

Events

Events allow you to add callbacks to VSGlobal's events:
OnConnect
OnDisconnect
OnError
OnPayloadReceived

Network

Network allow you to send payloads to those who are subscribed to an endpoint (or multiple endpoints):
Broadcast<T>(T, string)
Subscribe(string)