

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\(string\).](#)

Class Events

Namespace: [VSGlobal](#)

Assembly: VSGlobal.dll

Events allow you to add callbacks to VSGlobal's events:

[OnConnect](#)

[OnDisconnect](#)

[OnError](#)

[OnPayloadReceived](#)

```
public static class Events
```

Inheritance

[object](#) ← Events

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Events

OnConnect

Invoked when VSGlobal has connected Using a lambda:

```
VSGlobal.Events.OnConnect += (e) =>
{
    // Do stuff on connect
    VSGlobal.Subscribe("my_module");
    VSGlobal.Broadcast("Hello World!", "my_module"); //> OnPayloadReceived
};
```

Using a function:

```
public void MyCoolOnConnect(EventArgs e)
{
```

```
        // Do stuff on connect
    }

    // Then later, in a function body somewhere we register the handler.
    VSGlobal.Events.OnConnect += MyCoolOnConnect;
```

```
public static event OnConnectHandler OnConnect
```

Event Type

[OnConnectHandler](#)

OnDisconnect

Invoked when VsGlobal has disconnected (banned, server issue, skill issue)

Using a lambda:

```
Events.OnDisconnect += (e) =>
{
    // Do stuff on disconnect
};
```

Using a function:

```
public void MyCoolOnDisconnect(EventArgs e)
{
    // Do stuff on disconnect
}

// Then later, in a function body somewhere we register the handler.
Events.OnDisconnect += MyCoolOnDisconnect;
```

```
public static event OnDisconnectHandler OnDisconnect
```

Event Type

OnError

```
Events.OnError += (e) =>
{
    // Do stuff on disconnect
    Console.WriteLine(e.Exception);
    Console.WriteLine(e.Message);
};
```

Using a function:

```
public void MyCoolOnDisconnect(WebSocketSharper.ErrorEventArgs e)
{
    // Do stuff on disconnect
}

// Then later, in a function body somewhere we register the handler.
Events.OnError += MyCoolOnDisconnect;
```

```
public static event OnErrorHandler OnError
```

Event Type

OnPayloadReceived

Invoked when VsGlobal receives a payload

Using a lambda:

```
Events.OnPayloadReceived += (e) =>
{
    // This will be called whenever a packet arrives, regardless of module or
    sender.
```

```

        if(e.payload.Module == "my_module_name")
        {
            // Now that we know the payload is for our module, we can try
            converting it to our expected types.
            MyCustomClass? myCustomThing =
e.payload.DeserializePacket<MyCustomClass>();
            if(myCustomThing is MyCustomClass packet)
            {
                DoSomething(myCustomThing.value);
            }
        }
        else
        {
            // It's someone else's packet. Could be handy for extension mods!
        }
    };

```

Using a function:

```

public void ReceiveMessagePacket(OnPayloadReceivedEventArgs e)
{
    // Same as the lambda, we have access to any payload coming in here.
    Message? maybeMessage = e.payload.DeserializePacket<Message>();

    // We can also be quite cheeky and attempt to deserialize it to our custom
    type regardless of module.
    // If it doesn't, it's not ours- So I suppose that's valid as well.
    if(maybeMessage is Message msg)
    {
        // Do something with our received custom message!
    }
}

// Ideally, within `public override void StartClientside(ICoreClientAPI)`
Events.OnPayloadReceived += ReceiveMessagePacket;

```

```

public static event OnPayloadReceivedHandler OnPayloadReceived

```

Event Type

[OnPayloadReceivedHandler](#)

See Also

[OnPayloadReceivedEventArgs](#)

See Also

[VSGlobal.EventArguments](#)

Class Network

Namespace: [VSGlobal](#)

Assembly: VSGlobal.dll

Network allow you to send payloads to those who are subscribed to an endpoint (or multiple endpoints):

[Broadcast<T>\(T, string\)](#).








[Subscribe\(string\)](#).

```
public static class Network
```

Inheritance

[object](#)  ← Network

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Methods

Broadcast<T>(T, string)

Called when you want to broadcast to one of your subscribed locations.

You must call `VSGlobal.Network.Subscribe("name_of_my_module_or_my_mod_id")` prior to using Broadcast. Try not to use "core".

If we are not connected to core, the packet will be dropped safely and the event logged. Example with a custom network message:

```
//First, we define our network packet somewhere like so.  
[ProtoContract(ImplicitFields = ImplicitFields.AllPublic)]  
public class CustomNetworkMessage  
{  
    public bool didSomething;  
    public IClientPlayer sender;  
    public string message = "Default Message";  
}  
  
// Later on, in a function body ...
```

```
// All we have to do is call broadcast. It's generic, so you can throw _anything_ in
// there. string, class, struct- Whatever.
VsGlobal.Broadcast(new CustomNetworkMessage()
{
    didSomething = true,
    sender = api.World.Player,
    message = "Grungus"
}, "broadcast", "my_module");
// What that will do is send the packet to the server and relay it to others.
// Once received, it'll invoke Events.OnPayloadReceived
```

```
public static Task Broadcast<T>(T packet, string module)
```

Parameters

packet T

module [string](#)

Returns

[Task](#)

Type Parameters

T

See Also

[OnPayloadReceived](#)

Subscribe(string)

Subscribe to a module to received broadcasted packets from. Call this from [OnConnect](#)

```
public static Task Subscribe(string module)
```


Parameters

`module` [string](#) 

Our mod's ID, or any underscore separated string. E.g. "global_chat" or "hungry_hungry_hippo"

Returns

[Task](#) 

Namespace VSGlobal.EventArguments

Classes

[OnPayloadReceivedEventArgs](#)

Provides [Payload](#)

Class OnPayloadReceivedEventArgs



Namespace: [VSGlobal.EventArguments](#)

Assembly: VSGlobal.dll









Provides [Payload](#)

```
public class OnPayloadReceivedEventArgs : EventArgs
```

Inheritance

[object](#)  ← [EventArgs](#)  ← OnPayloadReceivedEventArgs

Inherited Members

[EventArgs.Empty](#)  , [object.Equals\(object\)](#)  , [object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  ,
[object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#)  ,
[object.ToString\(\)](#) 

Fields

payload

```
public required Payload payload
```

Field Value

[Payload](#)

Namespace VSGlobal.Proto

Classes

[Payload](#)

[PayloadExtensionMethods](#)

Class Payload

Namespace: [VSGlobal.Proto](#)








Assembly: VSGlobal.dll

```
[ProtoContract]  
public class Payload
```

Inheritance

[object](#)  ← Payload

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

Extension Methods

[PayloadExtensionMethods.DeserializePacket<T>\(Payload\)](#),
[PayloadExtensionMethods.Serialize\(Payload\)](#), [PayloadExtensionMethods.Serialize<T>\(Payload, T\)](#)

Constructors

Payload()

```
public Payload()
```

Payload(string)

```
public Payload(string triggerEvent)
```

Parameters

triggerEvent [string](#) 

Payload(string, string)

```
public Payload(string triggerEvent, string module)
```

Parameters

triggerEvent [string](#) 

module [string](#) 

Properties

Event

```
[ProtoMember(2)]  
public string Event { get; set; }
```

Property Value

[string](#) 

Module

```
[ProtoMember(1)]  
public string Module { get; set; }
```

Property Value

[string](#) 

PacketType

```
[ProtoMember(3)]  
public string PacketType { get; set; }
```

Property Value

[string](#)

PacketValue

```
[ProtoMember(4)]  
public byte[] PacketValue { get; set; }
```

Property Value

[byte](#)[]

Methods

Deserialize(byte[], int)

```
public static Payload Deserialize(byte[] buffer, int responseSize)
```

Parameters

buffer [byte](#)[]

responseSize [int](#)

Returns

[Payload](#)

Class PayloadExtensionMethods

Namespace: [VSGlobal.Proto](#)

Assembly: VSGlobal.dll

```
public static class PayloadExtensionMethods
```

Inheritance

[object](#) ← PayloadExtensionMethods

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Methods

DeserializePacket<T>(Payload)

```
public static T? DeserializePacket<T>(this Payload payload)
```

Parameters

payload [Payload](#)

Returns

T

Type Parameters

T

Serialize(Payload)


```
public static byte[] Serialize(this Payload payload)
```

Parameters

payload [Payload](#)

Returns

[byte](#)[]

Serialize<T>(Payload, T)

```
public static byte[] Serialize<T>(this Payload payload, T packetValue)
```

Parameters

payload [Payload](#)

packetValue T

Returns

[byte](#)[]

Type Parameters

T