

Receiving Structured Data

Client libraries may provide tools to help receive structured data, like JSON. The core traffic to the NATS server will always be opaque byte arrays. The server does not process message payloads in any form. For libraries that don't provide helpers, you can always encode and decode data before sending the associated bytes to the NATS client.

For example, to receive JSON you could do:

Go

```
nc, err := nats.Connect("demo.nats.io",
    nats.ErrorHandler(func(nc *nats.Conn, s *nats.Subscription, err error) {
        if s != nil {
            log.Printf("Async error in %q/%q: %v", s.Subject, s.Queue, err)
        } else {
            log.Printf("Async error outside subscription: %v", err)
        }
    }))
if err != nil {
    log.Fatal(err)
}
defer nc.Close()
ec, err := nats.NewEncodedConn(nc, nats.JSON_ENCODER)
if err != nil {
    log.Fatal(err)
}
defer ec.Close()

// Define the object
type stock struct {
    Symbol string
    Price  int
}

wg := sync.WaitGroup{}
wg.Add(1)
```

```

// Subscribe
// Decoding errors will be passed to the function supplied via
// nats.ErrorHandler above, and the callback supplied here will
// not be invoked.
if _, err := ec.Subscribe("updates", func(s *stock) {
    log.Printf("Stock: %s - Price: %v", s.Symbol, s.Price)
    wg.Done()
}); err != nil {
    log.Fatal(err)
}

// Wait for a message to come in
wg.Wait()

```

Java

```

class StockForJsonSub {
    public String symbol;
    public float price;

    public String toString() {
        return symbol + " is at " + price;
    }
}

public class SubscribeJSON {
    public static void main(String[] args) {

        try {
            Connection nc = Nats.connect("nats://demo.nats.io:4222");

            // Use a latch to wait for 10 messages to arrive
            CountDownLatch latch = new CountDownLatch(10);

            // Create a dispatcher and inline message handler
            Dispatcher d = nc.createDispatcher((msg) -> {
                Gson gson = new Gson();

                String json = new String(msg.getData(), StandardCharsets.UTF_8);
                StockForJsonSub stk = gson.fromJson(json, StockForJsonSub.class);
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

        // Use the object
        System.out.println(stk);

        latch.countDown();
    });

    // Subscribe
    d.subscribe("updates");

    // Wait for a message to come in
    latch.await();

    // Close the connection
    nc.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

JavaScript

```

const sub = nc.subscribe(subj, {
  callback: (_err, msg) => {
    t.log(`${msg.json()}`);
  },
  max: 1,
});

```

Python

```

import asyncio
import json
from nats.aio.client import Client as NATS
from nats.aio.errors import ErrTimeout

async def run(loop):
    nc = NATS()

```

```

await nc.connect(servers=["nats://127.0.0.1:4222"], loop=loop)

async def message_handler(msg):
    data = json.loads(msg.data.decode())
    print(data)

sid = await nc.subscribe("updates", cb=message_handler)
await nc.flush()

await nc.auto_unsubscribe(sid, 2)
await nc.publish("updates", json.dumps({"symbol": "GOOG", "price": 1200 }).encode())
await asyncio.sleep(1, loop=loop)
await nc.close()

```

C#

```

// dotnet add package NATS.Net
using NATS.Net;

// NATS .NET has a built-in serializer that does the 'unsurprising' thing
// for most types. Most primitive types are serialized as expected.
// For any other type, JSON serialization is used. You can also provide
// your own serializers by implementing the INatsSerializer and
// INasSerializerRegistry interfaces. See also for more information:
// https://nats-io.github.io/nats.net/documentation/advanced/serialization.html
await using var nc = new NatsClient();

CancellationTokenSource cts = new();

// Subscribe for int, string, bytes, json
List<Task> tasks =
[
    Task.Run(async () =>
    {
        await foreach (var msg in nc.SubscribeAsync<int>("x.int", cancellationTok
cts.Token))
        {
            Console.WriteLine($"Received int: {msg.Data}");
        }
    }),

```

```

Task.Run(async () =>
{
    await foreach (var msg in nc.SubscribeAsync<string>("x.string", cancellationToken:
cts.Token))
    {
        Console.WriteLine($"Received string: {msg.Data}");
    }
}),

Task.Run(async () =>
{
    await foreach (var msg in nc.SubscribeAsync<byte[]>("x.bytes", cancellationToken:
cts.Token))
    {
        if (msg.Data != null)
        {
            Console.Write($"Received bytes: ");
            foreach (var b in msg.Data)
            {
                Console.Write("0x{0:X2} ", b);
            }
            Console.WriteLine();
        }
    }
}),

Task.Run(async () =>
{
    await foreach (var msg in nc.SubscribeAsync<MyData>("x.json", cancellationToken:
cts.Token))
    {
        Console.WriteLine($"Received data: {msg.Data}");
    }
}),
];

// Give the subscriber tasks some time to subscribe
await Task.Delay(1000);

```

```

await nc.PublishAsync<int>("x.int", 100);
await nc.PublishAsync<string>("x.string", "Hello, World!");
await nc.PublishAsync<byte[]>("x.bytes", new byte[] { 0x41, 0x42, 0x43 });
await nc.PublishAsync<MyData>("x.json", new MyData(30, "bar"));

await cts.CancelAsync();

await Task.WhenAll(tasks);

public record MyData(int Id, string Name);

// Output:
// Received int: 100
// Received bytes: 0x41 0x42 0x43
// Received string: Hello, World!
// Received data: MyData { Id = 30, Name = bar }

// See also for more information:
// https://nats-io.github.io/nats.net/documentation/advanced/serialization.html

```

Ruby

```

require 'nats/client'
require 'json'

NATS.start(servers:["nats://127.0.0.1:4222"]) do |nc|
  nc.subscribe("updates") do |msg|
    m = JSON.parse(msg)

    # {"symbol"=>"GOOG", "price"=>12}
    p m
  end
end

```

C

```
// Structured data is not configurable in C NATS Client.
```

```

[] #1
[] cocobook-sync [] 2026-10-28 11:12:47 UTC
[] cocobook-sync [] 2026-10-28 11:12:47 UTC

```