

Wildcard Subscriptions

There is no special code to subscribe with a [wildcard subject](#). Wildcards are a normal part of the subject name. However, it is a common technique to use the subject provided with the incoming message to determine what to do with the message.

For example, you can subscribe using `*` and then act based on the actual subject.

Go

```
nc, err := nats.Connect("demo.nats.io")
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

// Use a WaitGroup to wait for 2 messages to arrive
wg := sync.WaitGroup{}
wg.Add(2)

// Subscribe
if _, err := nc.Subscribe("time.*.east", func(m *nats.Msg) {
    log.Printf("%s: %s", m.Subject, m.Data)
    wg.Done()
}); err != nil {
    log.Fatal(err)
}

// Wait for the 2 messages to come in
wg.Wait()
```

Java

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

// Use a latch to wait for 2 messages to arrive
CountDownLatch latch = new CountDownLatch(2);
```

```
// Create a dispatcher and inline message handler
Dispatcher d = nc.createDispatcher((msg) -> {
    String subject = msg.getSubject();
    String str = new String(msg.getData(), StandardCharsets.UTF_8);
    System.out.println(subject + ": " + str);
    latch.countDown();
});

// Subscribe
d.subscribe("time.*.east");

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

// Close the connection
nc.close();
```

JavaScript

```
nc.subscribe("time.us.*", (_err, msg) => {
    // converting timezones correctly in node requires a library
    // this doesn't take into account *many* things.
    let time;
    switch (msg.subject) {
        case "time.us.east":
            time = new Date().toLocaleTimeString("en-us", {
                timeZone: "America/New_York",
            });
            break;
        case "time.us.central":
            time = new Date().toLocaleTimeString("en-us", {
                timeZone: "America/Chicago",
            });
            break;
        case "time.us.mountain":
            time = new Date().toLocaleTimeString("en-us", {
                timeZone: "America/Denver",
            });
            break;
        case "time.us.west":
```

```

        time = new Date().toLocaleTimeString("en-us", {
            timeZone: "America/Los_Angeles",
        });
        break;
    default:
        time = "I don't know what you are talking about Willis";
    }
    t.log(subject, time);
});

```

Python

```

nc = NATS()

await nc.connect(servers=["nats://demo.nats.io:4222"])

# Use queue to wait for 2 messages to arrive
queue = asyncio.Queue()
async def cb(msg):
    await queue.put_nowait(msg)

await nc.subscribe("time.*.east", cb=cb)

# Send 2 messages and wait for them to come in
await nc.publish("time.A.east", b'A')
await nc.publish("time.B.east", b'B')

msg_A = await queue.get()
msg_B = await queue.get()

print("Msg A:", msg_A)
print("Msg B:", msg_B)

```

C#

```

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

await using var client = new NatsClient();

```

```

var count = 0;
await foreach (var msg in client.SubscribeAsync<string>("time.*.east"))
{
    Console.WriteLine($"Received {++count}: {msg.Subject}: {msg.Data}");

    if (count == 2)
    {
        break;
    }
}

Console.WriteLine("Done");

// Output:
// Received 1: time.us.east: 2024-10-21T22:11:24 America/New_York (-04)
// Received 2: time.eu.east: 2024-10-22T04:11:24 Europe/Warsaw (+02)
// Done

```

Ruby

```

require 'nats/client'
require 'fiber'

NATS.start(servers:["nats://127.0.0.1:4222"]) do |nc|
  Fiber.new do
    f = Fiber.current

    nc.subscribe("time.*.east") do |msg, reply|
      f.resume Time.now
    end

    nc.publish("time.A.east", "A")
    nc.publish("time.B.east", "B")

    # Use the response
    msg_A = Fiber.yield
    puts "Msg A: #{msg_A}"

    msg_B = Fiber.yield
    puts "Msg B: #{msg_B}"
  end
end

```

```
end.resume
end
```

C

```
static void
onMsg(natsConnection *conn, natsSubscription *sub, natsMsg *msg, void *closure)
{
    printf("Received msg: %s - %.*s\n",
           natsMsg_GetSubject(msg),
           natsMsg_GetDataLength(msg),
           natsMsg_GetData(msg));

    // Need to destroy the message!
    natsMsg_Destroy(msg);
}

(...)

natsConnection      *conn = NULL;
natsSubscription     *sub  = NULL;
natsStatus           s;

s = natsConnection_ConnectTo(&conn, NATS_DEFAULT_URL);
if (s == NATS_OK)
    s = natsConnection_Subscribe(&sub, conn, "time.*.east", onMsg, NULL);

(...)

// Destroy objects that were created
natsSubscription_Destroy(sub);
natsConnection_Destroy(conn);
```

or do something similar with `>`:

Go

```

nc, err := nats.Connect("demo.nats.io")
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

// Use a WaitGroup to wait for 4 messages to arrive
wg := sync.WaitGroup{}
wg.Add(4)

// Subscribe
if _, err := nc.Subscribe("time.>", func(m *nats.Msg) {
    log.Printf("%s: %s", m.Subject, m.Data)
    wg.Done()
}); err != nil {
    log.Fatal(err)
}

// Wait for the 4 messages to come in
wg.Wait()

// Close the connection
nc.Close()

```

Java

```

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

// Use a latch to wait for 4 messages to arrive
CountDownLatch latch = new CountDownLatch(4);

// Create a dispatcher and inline message handler
Dispatcher d = nc.createDispatcher((msg) -> {
    String subject = msg.getSubject();
    String str = new String(msg.getData(), StandardCharsets.UTF_8);
    System.out.println(subject + ": " + str);
    latch.countDown();
});

// Subscribe

```

```
d.subscribe("time.>");

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

// Close the connection
nc.close();
```

JavaScript

```
let nc = NATS.connect({
  url: "nats://demo.nats.io:4222"
});

nc.subscribe('time.>', (msg, reply, subject) => {
  // converting timezones correctly in node requires a library
  // this doesn't take into account *many* things.
  let time = "";
  switch (subject) {
    case 'time.us.east':
      time = new Date().toLocaleTimeString("en-us", {timeZone: "America/New_York"});
      break;
    case 'time.us.central':
      time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Chicago"});
      break;
    case 'time.us.mountain':
      time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Denver"});
      break;
    case 'time.us.west':
      time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Los_Angeles"});
      break;
    default:
      time = "I don't know what you are talking about Willis";
  }
  t.log(subject, time);
});
```

Python

```

nc = NATS()

await nc.connect(servers=["nats://demo.nats.io:4222"])

# Use queue to wait for 4 messages to arrive
queue = asyncio.Queue()
async def cb(msg):
    await queue.put(msg)

await nc.subscribe("time.>", cb=cb)

# Send 2 messages and wait for them to come in
await nc.publish("time.A.east", b'A')
await nc.publish("time.B.east", b'B')
await nc.publish("time.C.west", b'C')
await nc.publish("time.D.west", b'D')

for i in range(0, 4):
    msg = await queue.get()
    print("Msg:", msg)

await nc.close()

```

C#

```

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

await using var client = new NatsClient();

var count = 0;
await foreach (var msg in client.SubscribeAsync<string>("time.>"))
{
    Console.WriteLine($"Received {++count}: {msg.Subject}: {msg.Data}");

    if (count == 4)
    {
        break;
    }
}

```

```
Console.WriteLine("Done");
```

```
// Output:
```

```
// Received 1: time.us.east: 2024-10-21T22:11:24 America/New_York (-04)
```

```
// Received 2: time.us.east.atlanta: 2024-10-21T22:11:24 America/New_York (-04)
```

```
// Received 3: time.eu.east: 2024-10-22T04:11:24 Europe/Warsaw (+02)
```

```
// Received 4: time.eu.east.warsaw: 2024-10-22T04:11:24 Europe/Warsaw (+02)
```

```
// Done
```

Ruby

```
require 'nats/client'
```

```
require 'fiber'
```

```
NATS.start(servers:["nats://127.0.0.1:4222"]) do |nc|
```

```
  Fiber.new do
```

```
    f = Fiber.current
```

```
    nc.subscribe("time.>") do |msg, reply|
```

```
      f.resume Time.now.to_f
```

```
    end
```

```
    nc.publish("time.A.east", "A")
```

```
    nc.publish("time.B.east", "B")
```

```
    nc.publish("time.C.west", "C")
```

```
    nc.publish("time.D.west", "D")
```

```
    # Use the response
```

```
    4.times do
```

```
      msg = Fiber.yield
```

```
      puts "Msg: #{msg}"
```

```
    end
```

```
  end.resume
```

```
end
```

TypeScript

```
await nc.subscribe('time.>', (err, msg) => {
```

```
  // converting timezones correctly in node requires a library
```

```

// this doesn't take into account *many* things.
let time = "";
switch (msg.subject) {
  case 'time.us.east':
    time = new Date().toLocaleTimeString("en-us", {timeZone: "America/New_York"});
    break;
  case 'time.us.central':
    time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Chicago"});
    break;
  case 'time.us.mountain':
    time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Denver"});
    break;
  case 'time.us.west':
    time = new Date().toLocaleTimeString("en-us", {timeZone: "America/Los_Angeles"});
    break;
  default:
    time = "I don't know what you are talking about Willis";
}
t.log(msg.subject, time);
});

```

C

```

static void
onMsg(natsConnection *conn, natsSubscription *sub, natsMsg *msg, void *closure)
{
    printf("Received msg: %s - %.*s\n",
           natsMsg_GetSubject(msg),
           natsMsg_GetDataLength(msg),
           natsMsg_GetData(msg));

    // Need to destroy the message!
    natsMsg_Destroy(msg);
}

(...)

natsConnection      *conn = NULL;
natsSubscription    *sub  = NULL;

```

```

natsStatus          s;

s = natsConnection_ConnectTo(&conn, NATS_DEFAULT_URL);
if (s == NATS_OK)
    s = natsConnection_Subscribe(&sub, conn, "time.>", onMsg, NULL);

(...)

// Destroy objects that were created
natsSubscription_Destroy(sub);
natsConnection_Destroy(conn);

```

The following example can be used to test these two subscribers. The `*` subscriber should receive at most 2 messages, while the `>` subscriber receives 4. More importantly the `time.*.east` subscriber won't receive on `time.us.east.atlanta` because that won't match.

Go

```

nc, err := nats.Connect("demo.nats.io")
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

zoneID, err := time.LoadLocation("America/New_York")
if err != nil {
    log.Fatal(err)
}
now := time.Now()
zoneDateTime := now.In(zoneID)
formatted := zoneDateTime.String()

nc.Publish("time.us.east", []byte(formatted))
nc.Publish("time.us.east.atlanta", []byte(formatted))

zoneID, err = time.LoadLocation("Europe/Warsaw")
if err != nil {
    log.Fatal(err)
}
zoneDateTime = now.In(zoneID)

```

```
formatted = zoneDateTime.String()

nc.Publish("time.eu.east", []byte(formatted))
nc.Publish("time.eu.east.warsaw", []byte(formatted))
```

Java

```
Connection nc = Nats.connect("nats://demo.nats.io:4222");
ZoneId zoneId = ZoneId.of("America/New_York");
ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(Instant.now(), zoneId);
String formatted = zonedDateTime.format(DateTimeFormatter.ISO_ZONED_DATE_TIME);

nc.publish("time.us.east", formatted.getBytes(StandardCharsets.UTF_8));
nc.publish("time.us.east.atlanta", formatted.getBytes(StandardCharsets.UTF_8));

zoneId = ZoneId.of("Europe/Warsaw");
zonedDateTime = ZonedDateTime.ofInstant(Instant.now(), zoneId);
formatted = zonedDateTime.format(DateTimeFormatter.ISO_ZONED_DATE_TIME);
nc.publish("time.eu.east", formatted.getBytes(StandardCharsets.UTF_8));
nc.publish("time.eu.east.warsaw", formatted.getBytes(StandardCharsets.UTF_8));

nc.flush(Duration.ZERO);
nc.close();
```

JavaScript

```
nc.publish('time.us.east');
nc.publish('time.us.central');
nc.publish('time.us.mountain');
nc.publish('time.us.west');
```

Python

```
nc = NATS()

await nc.connect(servers=["nats://demo.nats.io:4222"])

await nc.publish("time.us.east", b'...')
await nc.publish("time.us.east.atlanta", b'...')

await nc.publish("time.eu.east", b'...')
```

```
await nc.publish("time.eu.east.warsaw", b'...')
```

```
await nc.close()
```

C#

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

await using var client = new NatsClient();

Instant now = SystemClock.Instance.GetCurrentInstant();

{
    DateTimeZone zone = DateTimeZoneProviders.Tzdb["America/New_York"];
    string formatted = now.InZone(zone).ToString();
    await client.PublishAsync("time.us.east", formatted);
    await client.PublishAsync("time.us.east.atlanta", formatted);
}

{
    DateTimeZone zone = DateTimeZoneProviders.Tzdb["Europe/Warsaw"];
    string formatted = now.InZone(zone).ToString();
    await client.PublishAsync("time.eu.east", formatted);
    await client.PublishAsync("time.eu.east.warsaw", formatted);
}
```

Ruby

```
NATS.start do |nc|
  nc.publish("time.us.east", '...')
  nc.publish("time.us.east.atlanta", '...')

  nc.publish("time.eu.east", '...')
  nc.publish("time.eu.east.warsaw", '...')

  nc.drain
end
```

TypeScript

```
nc.publish('time.us.east');  
nc.publish('time.us.central');  
nc.publish('time.us.mountain');  
nc.publish('time.us.west');
```

#2

cocobook-sync 2026-10-28 11:12:43 UTC

cocobook-sync 2026-10-28 11:17:53 UTC