

Including a Reply Subject

The optional reply-to field when publishing a message can be used on the receiving side to respond. The reply-to subject is often called an *inbox*, and most libraries may provide a method for generating unique inbox subjects. Most libraries also provide for the request-reply pattern with a single call. For example to send a request to the subject `time`, with no content for the messages, you might:

Go

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

// Create a unique subject name for replies.
uniqueReplyTo := nats.NewInbox()

// Listen for a single response
sub, err := nc.SubscribeSync(uniqueReplyTo)
if err != nil {
    log.Fatal(err)
}

// Send the request.
// If processing is synchronous, use Request() which returns the response message.
if err := nc.PublishRequest("time", uniqueReplyTo, nil); err != nil {
    log.Fatal(err)
}

// Read the reply
msg, err := sub.NextMsg(time.Second)
if err != nil {
    log.Fatal(err)
}

// Use the response
```

```
log.Printf("Reply: %s", msg.Data)
```

Java

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

// set up a listener for "time" requests
Dispatcher d = nc.createDispatcher(msg -> {
    System.out.println("Received time request");
    nc.publish(msg.getReplyTo(), (" " + System.currentTimeMillis()).getBytes());
});
d.subscribe("time");

// make a subject for replies and subscribe to that
String replyToThis = NUID.nextGlobal();
Subscription sub = nc.subscribe(replyToThis);

// publish to the "time" subject with reply-to subject that was set up
nc.publish("time", replyToThis, null);

// wait for a response
Message msg = sub.nextMessage(1000);

// look at the response
long time = Long.parseLong(new String(msg.getData()));
System.out.println(new Date(time));

nc.close();
```

JavaScript

```
// set up a subscription to process the request
const sc = StringCodec();
nc.subscribe("time", {
    callback: (_err, msg) => {
        msg.respond(sc.encode(new Date().toLocaleTimeString()));
    },
});

// create a subscription subject that the responding send replies to
```

```
const inbox = createInbox();
const sub = nc.subscribe(inbox, {
  max: 1,
  callback: (_err, msg) => {
    t.log(`the time is ${sc.decode(msg.data)}`);
  },
});

nc.publish("time", Empty, { reply: inbox });
```

Python

```
nc = NATS()

future = asyncio.Future()

async def sub(msg):
    nonlocal future
    future.set_result(msg)

await nc.connect(servers=["nats://demo.nats.io:4222"])
await nc.subscribe("time", cb=sub)

unique_reply_to = nc.new_inbox()
await nc.publish("time", b'', unique_reply_to)

# Use the response
msg = await asyncio.wait_for(future, 1)
print("Reply:", msg)
```

C#

```
// dotnet add package NATS.Net
using NATS.Net;
using NATS.Client.Core;

await using var client = new NatsClient();

await client.ConnectAsync();
```

```

// Create a new inbox for the subscription subject
string inbox = client.Connection.NewInbox();

// Use core API to subscribe to have a more fine-grained control over
// the subscriptions. We use <string> as the type, but we are not
// really interested in the message payload.
await using INatsSub<string> timeSub
    = await client.Connection.SubscribeCoreAsync<string>("time");

Task responderTask = Task.Run(async () =>
{
    await foreach (var msg in timeSub.Msgs.ReadAllAsync())
    {
        // The default serializer uses StandardFormat with Utf8Formatter
        // when formatting DateTimeOffset types.
        await msg.ReplyAsync<DateTimeOffset>(DateTimeOffset.UtcNow);
    }
});

// Subscribe to the inbox with the expected type of the response
await using INatsSub<DateTimeOffset> inboxSub
    = await client.Connection.SubscribeCoreAsync<DateTimeOffset>(inbox);

// The default serializer uses UTF-8 encoding for strings
await client.PublishAsync(subject: "time", replyTo: inbox);

// Read the response from subscription message channel reader
NatsMsg<DateTimeOffset> reply = await inboxSub.Msgs.ReadAsync();

// Print the current time in RFC1123 format taking advantage of the
// DateTimeOffset's formatting capabilities.
Console.WriteLine($"The current date and time is: {reply.Data:R}");

await inboxSub.UnsubscribeAsync();
await timeSub.UnsubscribeAsync();

// make sure the responder task is completed cleanly
await responderTask;

// Output:

```

```
// The current date and time is: Tue, 22 Oct 2024 12:21:09 GMT
```

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 msg
    end

    nc.publish("time", 'example', NATS.create_inbox)

    # Use the response
    msg = Fiber.yield
    puts "Reply: #{msg}"

    end.resume
  end
end
```

C

```
natsConnection      *conn      = NULL;
natsStatus           s          = NATS_OK;

s = natsConnection_ConnectTo(&conn, NATS_DEFAULT_URL);
// Publish a message and provide a reply subject
if (s == NATS_OK)
    s = natsConnection_PublishRequestString(conn, "request", "reply", "this is the request");

(...)

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

📄 #1

📄 cocobook-sync 📄 2026-10-28 11:12:49 UTC

📄 cocobook-sync 📄 2026-10-28 11:12:49 UTC