

Managing Streams and consumers

Streams and durable consumers can be defined administratively outside the application (typically using the NATS CLI Tool) in which case the application only needs to know about the well-known names of the durable consumers it wants to use. But you can also manage streams and consumers programmatically.

Common stream management operations are:

- Add a stream. Adding a stream is an idempotent function, which means that if a stream does not exist, it will be created, and if a stream already exists, then the add operation will succeed only if the existing stream matches exactly the attributes specified in the 'add' call.
- Delete a stream.
- Purge a stream (delete all the messages stored in the stream)
- Get or remove a specific message from a stream by sequence number
- Add or update (or delete) a consumer
- Get info and statistics on streams/consumers/account. Get/remove/get information on individual messages stored in a stream.

Go

```
func ExampleJetStreamManager() {
    nc, _ := nats.Connect("localhost")

    js, _ := nc.JetStream()

    // Create a stream
    js.AddStream(&nats.StreamConfig{
        Name: "example-stream",
        Subjects: []string{"example-subject"},
        MaxBytes: 1024,
    })

    // Update a stream
    js.UpdateStream(&nats.StreamConfig{
        Name: "example-stream",
```

```

    []MaxBytes: 2048,
    []})

    []// Create a durable consumer
    []js.AddConsumer("example-stream", &nats.ConsumerConfig{
    []    []Durable: "example-consumer-name",
    []})

    []// Get information about all streams (with Context JSOpt)
    []ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    []defer cancel()
    []for info := range js.StreamsInfo(nats.Context(ctx)) {
    []    []fmt.Println("stream name: ", info.Config.Name)
    []}

    []// Get information about all consumers (with MaxWait JSOpt)
    []for info := range js.ConsumersInfo("example-stream", nats.MaxWait(10*time.Second)) {
    []    []fmt.Println("consumer name: ", info.Name)
    []}

    []// Delete a consumer
    []js.DeleteConsumer("example-stream", "example-consumer-name")

    []// Delete a stream
    []js.DeleteStream("example-stream")
    []}

```

Java

```

try (Connection nc = Nats.connect("localhost")) {
    JetStreamManagement jsm = nc.jetStreamManagement();

    // Create a stream
    StreamInfo si = jsm.addStream(StreamConfiguration.builder()
        .name("example-stream")
        .subjects("example-subject")
        .maxBytes(1024)
        .build());
    StreamConfiguration config = si.getConfiguration();
    System.out.println("stream name: " + config.getName() + ", max_bytes: " +

```

```

config.getMaxBytes());

// Update a stream
si = jsr.updateStream(StreamConfiguration.builder()
    .name("example-stream")
    .maxBytes(2048)
    .build());
config = si.getConfiguration();
System.out.println("stream name: " + config.getName() + ", max_bytes: " +
config.getMaxBytes());

// Create a durable consumer
jsr.createConsumer("example-stream", ConsumerConfiguration.builder()
    .durable("example-consumer-name")
    .build());

// Get information about all streams
List<StreamInfo> streams = jsr.getStreams();
for (StreamInfo info : streams) {
    System.out.println("stream name: " + info.getConfiguration().getName());
}

// Get information about all consumers
List<ConsumerInfo> consumers = jsr.getConsumers("example-stream");
for (ConsumerInfo ci : consumers) {
    System.out.println("consumer name: " + ci.getName());
}

// Delete a consumer
jsr.deleteConsumer("example-stream", "example-consumer-name");

// Delete a stream
jsr.deleteStream("example-stream");
}

```

JavaScript

```

import { AckPolicy, connect, Empty } from "../../src/mod.ts";

const nc = await connect();

```

```
const jsm = await nc.jetstreamManager();

// list all the streams, the `next()` function
// retrieves a paged result.
const streams = await jsm.streams.list().next();
streams.forEach((si) => {
    console.log(si);
});

// add a stream
const stream = "mystream";
const subj = `mystream.*`;
await jsm.streams.add({ name: stream, subjects: [subj] });

// publish a reg nats message directly to the stream
for (let i = 0; i < 10; i++) {
    nc.publish(`${subj}.a`, Empty);
}

// find a stream that stores a specific subject:
const name = await jsm.streams.find("mystream.A");

// retrieve info about the stream by its name
const si = await jsm.streams.info(name);

// update a stream configuration
si.config.subjects?.push("a.b");
await jsm.streams.update(name, si.config);

// get a particular stored message in the stream by sequence
// this is not associated with a consumer
const sm = await jsm.streams.getMessage(stream, { seq: 1 });
console.log(sm.seq);

// delete the 5th message in the stream, securely erasing it
await jsm.streams.deleteMessage(stream, 5);

// purge all messages in the stream, the stream itself
// remains.
await jsm.streams.purge(stream);
```

```

// purge all messages with a specific subject (filter can be a wildcard)
await jsm.streams.purge(stream, { filter: "a.b" });

// purge messages with a specific subject keeping some messages
await jsm.streams.purge(stream, { filter: "a.c", keep: 5 });

// purge all messages with upto (not including seq)
await jsm.streams.purge(stream, { seq: 100 });

// purge all messages with upto sequence that have a matching subject
await jsm.streams.purge(stream, { filter: "a.d", seq: 100 });

// list all consumers for a stream:
const consumers = await jsm.consumers.list(stream).next();
consumers.forEach((ci) => {
    console.log(ci);
});

// add a new durable pull consumer
await jsm.consumers.add(stream, {
    durable_name: "me",
    ack_policy: AckPolicy.Explicit,
});

// retrieve a consumer's configuration
const ci = await jsm.consumers.info(stream, "me");
console.log(ci);

// delete a particular consumer
await jsm.consumers.delete(stream, "me");

```

Python

```

import asyncio

import nats
from nats.errors import TimeoutError

async def main():

```

```

nc = await nats.connect("localhost")

# Create JetStream context.
js = nc.jetstream()

# Persist messages on 'foo's subject.
await js.add_stream(name="sample-stream", subjects=["foo"])

await nc.close()

if __name__ == '__main__':
    asyncio.run(main())

```

C#

```

// dotnet add package NATS.Net
using NATS.Net;
using NATS.Client.JetStream;
using NATS.Client.JetStream.Models;

await using var client = new NatsClient();

INatsJSContext js = client.CreateJetStreamContext();

// Create a stream
var streamConfig = new StreamConfig(name: "example-stream", subjects: ["example-subject"])
{
    MaxBytes = 1024,
};
await js.CreateStreamAsync(streamConfig);

// Update the stream
var streamConfigUpdated = streamConfig with { MaxBytes = 2048 };
await js.UpdateStreamAsync(streamConfigUpdated);

// Create a durable consumer
await js.CreateConsumerAsync("example-stream", new ConsumerConfig("example-consumer-name"));

// Get information about all streams
await foreach (var stream in js.ListStreamsAsync())

```

```

{
    Console.WriteLine($"stream name: {stream.Info.Config.Name}");
}

// Get information about all consumers in a stream
await foreach (var consumer in js.ListConsumersAsync("example-stream"))
{
    Console.WriteLine($"consumer name: {consumer.Info.Config.Name}");
}

// Delete a consumer
await js.DeleteConsumerAsync("example-stream", "example-consumer-name");

// Delete a stream
await js.DeleteStreamAsync("example-stream");

// Output:
// stream name: example-stream
// consumer name: example-consumer-name

```

C

```

#include "examples.h"

static const char *usage = ""\
"-stream      stream name (default is 'foo')\n" \
"-txt         text to send (default is 'hello')\n" \
"-count       number of messages to send\n" \
"-sync        publish synchronously (default is async)\n";

static void
_jsPubErr(jsCtx *js, jsPubAckErr *pae, void *closure)
{
    int *errors = (int*) closure;

    printf("Error: %u - Code: %u - Text: %s\n", pae->Err, pae->ErrCode, pae->ErrText);
    printf("Original message: %.*s\n", natsMsg_GetDataLength(pae->Msg), natsMsg_GetData(pae->Msg));

    *errors = (*errors + 1);
}

```

```

// If we wanted to resend the original message, we would do something like that:
//
// js_PublishMsgAsync(js, &(pae->Msg), NULL);
//
// Note that we use `&(pae->Msg)` so that the library set it to NULL if it takes
// ownership, and the library will not destroy the message when this callback returns.

// No need to destroy anything, everything is handled by the library.
}

int main(int argc, char **argv)
{
    natsConnection      *conn  = NULL;
    natsStatistics      *stats = NULL;
    natsOptions          *opts  = NULL;
    jsCtx               *js     = NULL;
    jsOptions            jsOpts;
    jsErrCode            jerr    = 0;
    natsStatus           s;
    int                  dataLen=0;
    volatile int         errors = 0;
    bool                 delStream = false;

    opts = parseArgs(argc, argv, usage);
    dataLen = (int) strlen(payload);

    s = natsConnection_Connect(&conn, opts);

    if (s == NATS_OK)
        s = jsOptions_Init(&jsOpts);

    if (s == NATS_OK)
    {
        if (async)
        {
            jsOpts.PublishAsync.ErrHandler          = _jsPubErr;
            jsOpts.PublishAsync.ErrHandlerClosure    = (void*) &errors;
        }
        s = natsConnection_JetStream(&js, conn, &jsOpts);
    }
}

```

```

if (s == NATS_OK)
{
    jsStreamInfo    *si = NULL;

    // First check if the stream already exists.
    s = js_GetStreamInfo(&si, js, stream, NULL, &jerr);
    if (s == NATS_NOT_FOUND)
    {
        jsStreamConfig  cfg;

        // Since we are the one creating this stream, we can delete at the end.
        delStream = true;

        // Initialize the configuration structure.
        jsStreamConfig_Init(&cfg);
        cfg.Name = stream;
        // Set the subject
        cfg.Subjects = (const char*[1]){subj};
        cfg.SubjectsLen = 1;
        // Make it a memory stream.
        cfg.Storage = js_MemoryStorage;
        // Add the stream,
        s = js_AddStream(&si, js, &cfg, NULL, &jerr);
    }
    if (s == NATS_OK)
    {
        printf("Stream %s has %" PRIu64 " messages (%" PRIu64 " bytes)\n",
            si->Config->Name, si->State.Msgs, si->State.Bytes);

        // Need to destroy the returned stream object.
        jsStreamInfo_Destroy(si);
    }
}

if (s == NATS_OK)
    s = natsStatistics_Create(&stats);

if (s == NATS_OK)
{
    printf("\nSending %" PRIu64 " messages to subject '%s'\n", total, stream);
    start = nats_Now();

```

```

}

for (count = 0; (s == NATS_OK) && (count < total); count++)
{
    if (async)
        s = js_PublishAsync(js, subj, (const void*) payload, dataLen, NULL);
    else
    {
        jsPubAck *pa = NULL;

        s = js_Publish(&pa, js, subj, (const void*) payload, dataLen, NULL, &jerr);
        if (s == NATS_OK)
        {
            if (pa->Duplicate)
                printf("Got a duplicate message! Sequence=%" PRIu64 "\n", pa->Sequence);

            jsPubAck_Destroy(pa);
        }
    }
}

if ((s == NATS_OK) && async)
{
    jsPubOptions    jsPubOpts;

    jsPubOptions_Init(&jsPubOpts);
    // Let's set it to 30 seconds, if getting "Timeout" errors,
    // this may need to be increased based on the number of messages
    // being sent.
    jsPubOpts.MaxWait = 30000;
    s = js_PublishAsyncComplete(js, &jsPubOpts);
    if (s == NATS_TIMEOUT)
    {
        // Let's get the list of pending messages. We could resend,
        // etc, but for now, just destroy them.
        natsMsgList list;

        js_PublishAsyncGetPendingList(&list, js);
        natsMsgList_Destroy(&list);
    }
}
}

```

```

if (s == NATS_OK)
{
    jsStreamInfo *si = NULL;

    elapsed = nats_Now() - start;
    printStats(STATS_OUT, conn, NULL, stats);
    printPerf("Sent");

    if (errors != 0)
        printf("There were %d asynchronous errors\n", errors);

    // Let's report some stats after the run
    s = js_GetStreamInfo(&si, js, stream, NULL, &jerr);
    if (s == NATS_OK)
    {
        printf("\nStream %s has %" PRIu64 " messages (%" PRIu64 " bytes)\n",
            si->Config->Name, si->State.Msgs, si->State.Bytes);

        jsStreamInfo_Destroy(si);
    }
}

if (delStream && (js != NULL))
{
    printf("\nDeleting stream %s: ", stream);
    s = js_DeleteStream(js, stream, NULL, &jerr);
    if (s == NATS_OK)
        printf("OK!");
    printf("\n");
}

if (s != NATS_OK)
{
    printf("Error: %u - %s - jerr=%u\n", s, natsStatus_GetText(s), jerr);
    nats_PrintLastErrorStack(stderr);
}

// Destroy all our objects to avoid report of memory leak
jsCtx_Destroy(js);
natsStatistics_Destroy(stats);
natsConnection_Destroy(conn);
natsOptions_Destroy(opts);

```

```
// To silence reports of memory still in used with valgrind
nats_Close();

return 0;
}
```

```
### #1
[] cocobook-sync [] 2026-10-28 11:13:00 UTC
[] cocobook-sync [] 2026-10-28 11:13:00 UTC
```