# API prefixes for materialized JetStream views

| Metadata | Value |
|---|---|
| Date | 2021-11-18 |
| Author | @mhanel |
| Status | Partially Implemented |
| Tags | jetstream, client, kv, objectstore |

## Context

This document describes a design on how to support API prefixes for materialized JS views.

API prefixes allow the client library to disambiguate access to independent JetStreams that run either in different domains or different accounts. By specifying the prefix in the API, a client program can essentially pick which one it wants to communicate with. This mechanism needs to be supported for materialized JS views as well.

## Overview

Each JetStream only listens to default API subjects with the prefix `$JS.API`. Thus, when the client uses `$JS.API`, it communicates with the JetStream, local to its account and domain.

To avoid traffic going to some other JetStream the following mechanisms are in place:

1. Account: Since the API has to be imported with an altered prefix, the request will not cross account boundaries. In a JetStream enabled account, an import without a prefix set will result in an error as JetStream is imported as well and we error on import overlaps.
2. Domain: On leaf node connections, the nats server adds in denies for `$JS.API.>`.

When the client library sets an API prefix, all API subjects, the client publishes and subscribes to start with that instead of `$JS.API`. As a result the API traffic will not end up in the local JetStream as that alway only ever subscribes to `$JS.API....`

As messages and subscriptions cross boundaries the following happens:

1. Accounts: When crossing the import, the API prefix is stripped and replaced with `$JS.API`.
2. Domain: When crossing the leaf node connection between domains, an automatically inserted mapping strips the API prefix and replaces it with `$JS.API`.

This JetStream disambiguation mechanism needs to be added to materialized views such as KV and object store as well. Specifically, we need to tag along on the same API prefix. Setting different values to reach the same JetStream is a non starter.

## Design

The first token of any API or materialized view is considered the default API prefix and will mean have local semantics. Thus, for the concrete views we treat the tokens `$KV` and `$OBJ` as the respective default API prefixes. For publishes and subscribes the API just replaces the first token with the specified (non default) prefix.

To share API access across accounts this is sufficient as account export/import takes care of the rest. To access an API in the same account but different domain, the `nats-server` maintaining a leaf node connection will add in the appropriate mappings from domain specific API to local API and deny local API traffic.

## KV Example

Assume the API prefix for JetStream has been set to `JS.from-acc1`. For JetStream specific API calls, the local API prefix `$JS.API` has to be replaced with `JS.from-acc1`.

Because the JetStream specified API prefix differs from `$JS.API`, the KV API uses the same prefix as is specififed for JetStream API.

For the KV API we prefix `$KV` with `JS.from-acc1`, resulting in `JS.from-acc1.$KV`. Thus, in order to put `key` in the bin `bin`, we send to `JS.from-acc1.$KV.bin.key`

When crossing the account boundaries, this is then translated back to `$KV.bin.key`. Thus, the underlying stream still needs to be created with a subscription to `$KV.bin.key`.

Domains are just a special case of API prefixes and will work the same way. The API prefix `$JS.<domain-name>.API` will lead to `$JS.<domain-name>.API.$KV.bin.key`. As the leaf node connection into the domain is crossed, the inserted mapping will changes the subject back to `$KV.bin.key`.

## Consequences

The proposed change is backwards compatible with materialized views already in use.

I suggested prefixing so we can have one prefix across different APIs. To avoid accidental API overlaps going forward, the implication for JetStream is to NOT start the first token after the API prefix with `$`.

Specifically, JetStream will never expose any functionality under `$JS.API.$KV.>`. The version with an API prefix would look as follows `JS.from-acc1.$KV`, which will clash with the same subject, used by kv. This is a side effect of JetStream having a two token API prefix and the materialized views using a single token.

This problem can be avoided by unifying the API name spaces to alway be two tokens with the second token being `API`, resulting in `$JS.API`, `$KV.API` and `$OBJ.API`. This however will not be backwards compatible.

## Testing

Here is a server config to test your changes.

- The JetStream prefix to use is `fromA`
- The inbox prefix to use is `forI`

```
jetstream: enabled
accounts: {
```

```
    A: {
        users: [ {user: a, password: a} ]
        jetstream: enabled
        exports: [
            {service: '$JS.API.>' }
            {service: '$KV.>'}
            {stream: 'forI.>' }
        ]
    },
    I: {
        users: [ {user: i, password: i} ]
        imports: [
            {service: {account: A, subject: '$JS.API.>'}, to: 'fromA.>' }
            {service: {account: A, subject: '$KV.>'}, to: 'fromA.$KV.>' }
            {stream: { subject: 'forI.>', account: 'A' } }
        ]
    }
}
```

Test JetStream connected to account `I` talking to JetStream in account `A`: `nats account info -s "nats://i:i@localhost:4222" --js-api-prefix fromA`

KV publishes and subscribes need to support the prefix as well. Absent an actual implementation this is simulated with pub/sub. Your implementation needs to be able to connect to account I and access map ojbjects in Account A.

```
nats -s "nats://a:a@localhost:4222" sub '$KV.map.>'  &
sleep 1
nats -s "nats://i:i@localhost:4222" pub 'fromA.$KV.map.put' "hello world"
```