Subject Transform

Metadata	Value
Date	2022-07-17
Author	@jnmoyne, @derekcollison, @tbeets
Status	Implemented
Tags	server

Context and Problem Statement

As part of multiple technical implementations of the NATS Server, there is a need to create a mapping formula, or _ transform_, that can be applied to an input subject, yielding a desired output subject.

Transforms can be used as part of:

- Core NATS Subject mapping at the account level
- JetStream stream definition
- JetStream sourcing
- JetStream RePublish
- Cross-account service and stream mapping
- Shadow subscriptions (e.g. across a leaf)

A subject transform shall be defined by a *Source* filter that defines what input subjects are eligible (via match) to be transformed and a *Destination* subject mapping format that defines the notional subject filter that the *transformed* output subject will match.

Input subject tokens that match Source wildcard(s) "survive" the transformation and can be used as input to 'mapping functions' whose output is used to build the output subject.

Design

Destination, taken together with Source, form a valid subject token transform. The resulting transform is applied to an input subject (that matches Source subject filter) to determine the output subject.

Weighted and cluster-scoped mappings

In the case of Core NATS subject mapping at the account level, you can actually have more than one mapping destination per source. Each of those mappings has a 'weight' (a percentage between 0 and 100%), for a total of 100%. That weight percentage indicate the likeliness of that mapping being used.

Furthermore, (as of 2.10) weighted mappings can be cluster-scoped meaning that you can also create mapping destinations (for a total of 100% per cluster name) that apply (and take precedence) when the server is part of the cluster specified. This allows administrators to define mappings that change depending upon the cluster where the message is initially published from.

For example consider the following mappings:

```
        "foo":[
               {destination:"foo.west", weight: 100%, cluster: "west"},
               {destination:"foo.central", weight: 100%, cluster: "central"},
               {destination:"foo.east", weight: 100%, cluster: "east"},
               {destination:"foo.elsewhere", weight: 100%}
        ],
```

Means that an application publishing a message on subject `foo` will result in the message being published over Core NATS with the subject:

- `foo.west` if the application is connected to any server in the `west` cluster
- `foo.central` if the application is connected to any server in the `central` cluster
- `foo.east` if the application is connected to any server in the `east` cluster

You can also define 100%'s worth of destinations as a catch-all for servers that apply for the other clusters in the Super-Cluster (or if the server is not running in clustered mode). In the example above a message published from cluster `south` would be mapped to `foo.elsewhere`.

Transform rules

A given input subject must match the Source filter (in the usual subscription-interest way) for the transform to be valid.

For a valid input subject:

- Source-matching literal-token positions are ignored, i.e. they are not usable by mapping functions.
- Source-matching wildcard-token positions (if any) are used by mapping functions in the transform Destination format.
- Source-matching `*` wildcard (single token) tokens are passed to Destination mapping functions by wildcard cardinal position number. E.g. using `$x` or `{{wildcard(x)}}` notation, where `x` is a number between 1 (first instance of a `*` wildcard-token in the Source filter) and `x` (x's instance of a `*` wildcard-token in the Source filter).
- Source-matching `>` wildcard (multi token) tokens are mapped to the respective `>` position in the Destination format
- Literal tokens in the Destination format are mapped to the output subject unchanged (position and value)

Using all the wildcard-tokens in the transform's Source

- For transforms that are defined in inter-account imports (streams and services) the destinations *MUST* make use of *ALL* of the wildcard-tokens present in the transform's Source.
- However, starting with version 2.10, for transforms used any other place (i.e. Core NATS account mappings, Subject transforms in streams, stream imports and stream republishing) it is allowed to drop any number of wildcard-tokens.

Mapping Functions

Mapping functions and placed in the transform Destination format as subject tokens. Their format is `{{MappingFunction()}}` and the legacy `$x` notation (equivalent to `{{Wildcard(x)}}`) is still supported for backwards compatibility.

> Note: Mapping functions names are valid in both upper CamelCase and all lower case (i.e. both `{{Wildcard(1)}}` and `{{wildcard(1)}}` are valid)

> For transforms defined in inter-accounts imports (streams and services) the *ONLY* allowed mapping function is `{{Wildcard(x)}}` (or the legacy `$x`)

List of Mapping Functions

Currently (2.9.0) the following mapping functions are available:

- `{{Wildcard(x)}}` outputs the value of the token for wildcard-token index `x` (equivalent to the legacy `$x` notation)
- `{{Partition(x,a,b,c,...)}}` ouputs a partition number between `0` and `x-1` assigned from a deterministic hashing of the value of wildcard-tokens `a`, `b` and `c`
- `{{Split(x,y)}}` splits the value of wildcard-token `x` into multiple tokens on the presence of character `y`
- `{{SplitFromLeft(x,y)}}` splits in two the value of wildcard-token `x` at `y` characters starting from the left
- `{{SplitFromRight(x,y)}}` splits in two the value of wildcard-token `x` at `y` characters starting from the right
- `{{SliceFromLeft(x,y)}}` slices into multiple tokens the value of wildcard-token `x`, every `y` characters starting from the left
- `{{SliceFromRight(x,y)}}` slices into multiple tokens the value of wildcard-token `x`, every `y` characters starting from the right

Example transforms

Input Subject	Source filter	Destination format	Output Subject
`one.two.three`	`">"`	`"uno.>"`	`uno.one.two.three`
`four.five.six`	`">"`	`"eins.>"`	`eins.four.five.six`
`one.two.three`	`">"`	`">"`	`one.two.three`

Input Subject	Source filter	Destination format	Output Subject
four.five.six	">"	"eins.zwei.drei.vier.>"	eins.zwei.drei.vier.four.five.six
one.two.three	"one.>"	"uno.>"	uno.two.three
one.two.three	"one.two.>"	"uno.dos.>" |uno.dos.three`	
one	"one"	"uno"	uno
one.two.three.four.five	"one.*.three.*.five"	"uno.$2.$1"	uno.four.two
one.two.three.four.five	"one.*.three.*.five"	"uno.{{wildcard(2)}}. {{wildcard(1)}}"	uno.four.two
one.two.three.four.five	"*.two.three.>"	"uno.$1.>"	uno.one.four.five
-abc-def--ghij-	"*"	"{{split(1,-)}}"	abc.def.ghi
12345	"*"	"{{splitfromleft(1,3)}}"	123.45
12345	"*"	" {{SplitFromRight(1,3)}}"	12.345
1234567890	"*"	"{{SliceFromLeft(1,3)}}"	123.456.789.0
1234567890	"*"	" {{SliceFromRight(1,3)}}"	1.234.567.890

> Note: The NATS CLI provides a utility, `server mappings` for experimenting with different transforms and input subjects.