KIP-821: Connect Transforms support for nested structures

- Status
- Motivation
- Proposed Changes
  - Nested notation
    - Rules
    - Examples
  - Affected SMTs
  - Non-affected SMTs
- Public Interfaces
  - New configuration flags
  - Affected SMTs
    - Cast
    - ExtractField
    - HeaderFrom
    - MaskField
    - ReplaceField
    - TimestampConverter
    - ValueToKey
    - InsertField
    - HoistField
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives
  - Keep ExtractField as it is and use it multiple times until reaching nested fields
  - Use dots as the only separator and escape with backslashes when collides
  - Use custom separators for edge cases
  - Use JSONPath notation to access nested elements
  - Use named styles instead of syntax versions
  - Use configuration flag per SMT instead of per-field configuration
  - Use Double-dots to escape dots included on field names
- Potential Improvements (out of scope)
  - Support Array access
  - Support Deep-Scan

Status

Current state: Voting

Discussion thread: here

JIRA: here

> ⚠ *Unable to render Jira issues macro, execution error.*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Single Message Transforms (SMT), introduced with KIP-66, have greatly improved Connector's usability by enabling the transformation of input/output data formats and contents without the need for additional streaming applications.

However, these benefits have been limited by SMT's limitation to only act on fields available at the root of the data structure.

Here are some tickets/comments related to this limitation:

- https://issues.apache.org/jira/browse/KAFKA-7624
- https://issues.apache.org/jira/browse/KAFKA-10640
- https://github.com/apache/kafka/blob/0c707b1fccd0b21a3ead765d61f376f338c69bd0/connect/transforms/src/main/java/org/apache/kafka/connect/transforms/Cast.java#L58-L59

```
    // TODO: Currently we only support top-level field casting. Ideally we could use a dotted notation
in the spec to
    // allow casting nested fields.
```

This KIP aims to include support for nested structures on the existing SMTs.

Proposed Changes

Nested notation

Dotted notation tends to be the most intuitive way to describe paths to nested fields in a record structure and will cover most of the scenarios. e.g. `jq` already uses it[1].

However, field names in JSON could include dots (e.g. `{'nested.field': {'value':42}}`).

Therefore, the nested notation must support escaping dots that could be valid field names.

Instead of escaping dots with backslashes — which in JSON configurations leads to unfriendly configurations — it's proposed to follow a similar approach as the JSONata[2] where backticks are used to define field names with dots, e.g. `nested.field`

[1] https://stedolan.github.io/jq/manual/#Basicfilters

[2] https://docs.jsonata.org/simple#examples

> Field references containing whitespace or reserved tokens can be enclosed in backticks

Rules

- 1. If **field names do not contain dots (.)**, then **only use dots** to represent nested field paths.
- 2. If field names **contain dots**, then:
  - **wrap** the field name with a **backtick pair (`...`)** by
    - adding an **opening backtick at the beginning of the field name** (beginning of a path, or after a dot)
    - adding a **closing backtick at the end of the field name** (end of the path, or before the next dot)
  - If a field is wrapped and doesn't contain dots, is processed the same way: field name within the wrapping backticks is used
- 3. If a **field name includes backticks**, then:
  - If a backtick is followed by a dot in the field name, then the backtick should be escaped with a **backslash** to signal that the backtick is part of the name and not closing a backtick pair.
    - Backslashes (\) do not need to be escaped. If the backslash happens to be part of the field name and before a backtick is to be escaped, then **add another backslash**.
  - else, backticks do not require escape
- 4. If wrapping backtick pairs are incomplete, the Connect configuration must fail fast to avoid getting ambiguous paths deployed.

Examples

Scenario	Nested struct	Path

1. Normal (no dots or backticks on field names)	`foo:`
	`  bar:`
	`  baz: val`
		`OK: foo.bar.baz`

2. Field names including dots	`foo:` `  bar.baz: val`	`OK: foo.`bar.baz``
2.1 Using backticks within a field name without dots	`foo:` `  bar:` `  baz: val`	`OK: foo.bar.baz` `ERROR: foo.`bar.baz: no pair` `ERROR: foo.bar`.baz: no pair`
3. Field names including backticks	`foo:` `  ba`r:` ` baz: val`	`OK: foo.ba`r.baz` `OK: foo.`ba`r`.baz`
3.1. Field names including backticks at the wrapping position	`foo:` `  bar`.`baz: val`	`OK: foo.`bar\`.\`baz` `ERROR: foo.`bar\`.`baz`: no pair` `ERROR: foo.`bar`.`baz`: valid but different path (see 2.1)`

3.2. Field names including dots and backticks between a backtick pair	`foo:` `  b`ar.baz: val`	`OK: foo.`b`ar.baz``
3.3. Field names including backslash and backticks at the wrapping position	`foo:` `  bar\`.`baz: val`	`OK: foo.`bar\\`.\`baz``
3.4. Field names wrapped by backticks	`foo:` `  `bar`:` `  baz: val`	`OK: foo.`\`bar\``.baz` `ERROR: foo.`bar`.baz: valid but different path (see 2.1)`

Affected SMTs

These SMTs will include support for nested structures:

- `Cast`
- `ExtractField`
- `HeaderFrom`
- `MaskField`
- `ReplaceField`
- `TimestampConverter`
- `ValueToKey`
- `InsertField`
- `HoistField`

Non-affected SMTs

These SMTs do not require nested structure support:

- `DropHeaders`: Drop one or multiple headers.
- `Filter`: Drops the whole message based on a predicate.
- `InsertHeader`: Insert a specific message to the header.
- `RegexRouter`: Acts on the topic name.
- `SetSchemaMetadata`: Acts on root schema.
- `TimestampRouter`: Acts on timestamp.
- `Flatten`: Acts on the whole key or message.

Public Interfaces

New configuration flags

Name	Type	Default	Importance	Documentation
`field.syntax.version`	`STRING`	`V1`	`HIGH`	Permitted values: `V1` , `V2` . Defines the version of the syntax to access fields. If set to "V1", then the field paths are limited to access the elements at the root level of the struct or map. If set to "V2", the syntax will support accessing nested elements. To access nested elements, dotted notation is used. If dots are already included in the field name, then backtick pairs can be used to wrap field names containing dots. e.g. to access elements from a struct/map named "foo.bar", the following format can be used to access its elements: "`foo.bar`.baz". This configuration will affect all the field paths used by the transform.

This flag will be added conditionally to some SMTs, as described below.

Affected SMTs

Cast

Changes:

- Extend `spec` to support nested notation.

Examples:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	  "parent": {
	  "child": {
	  "k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.Cast$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.spec": "k1:string,parent.child.k2:int64"
		}</pre>
			<pre>{
			  "k1": "123",
			  "parent": {
			"child": {
			  "k2": 123
			  }
			  }
			}</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	  "parent.child": {
	  "k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.Cast$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.spec": "k1:string,`parent.child`.k2:int64"
		}</pre>
			<pre>{
			  "k1": "123",
			  "parent.child": {
			  "k2": 123
			  }
			}</pre>

ExtractField

Changes:

- Extend `field` to support nested notation.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	  "parent": {
	  "child": {
	  "k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ExtractField$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.field": "parent.child.k2"
		}</pre>
			<pre>"123"</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	  "parent.child": {
	  "k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ExtractField$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.field": "`parent.child`.k2"
		}</pre>
			<pre>"123"</pre>

3. Nested field, an object returned.	<pre>{
	  "k1": 123,
	  "parent": {
	  "child":
	{
	  "k2":
	"123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.
		transforms.ExtractField$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.field": "parent.child"
		}</pre>
			<pre>{ "k2":
			"123" }</pre>

HeaderFrom

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1":
	123,
	
	"parent":
	{
	
	"child": {
	
	"k2":
	"123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.
		HeaderFrom$Value",
		"transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "k1,parent.child.k2",
		"transforms.smt1.headers": "k1,k2"
		}</pre>
			<pre>head
			ers:
			-
			k1=1
			23
			-
			k2="
			123"</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1":
	123,
	  "parent.
	child": {
	  "k2":
	"123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.
		HeaderFrom$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "k1,`parent.child`.k2",
		"transforms.smt1.headers": "k1,k2"
		}</pre>
			<pre>head
			ers:
			-
			k1=1
			23
			-
			k2="
			123"</pre>

MaskField

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	
	"parent": {
	
	"child": {
	
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.MaskField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "parent.child.k2"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent": {
			
			"child": {
			
			"k2": ""
			  }
			  }
			}</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	
	"parent.child": {
	
	"k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.MaskField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "`parent.child`.k2"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent.child": {
			
			"k2": ""
			  }
			}</pre>

ReplaceField

Changes:

- Extend the `include` and `exclude` lists

Example:

scenario	input	SMT	output

1. Nested field. Drop field	<pre>{
	  "k1": 123,
	
	"parent": {
	
	"child": {
	
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ReplaceField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.exclude": "parent.child.k2"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent": {
			
			"child": {
			  }
			  }
			}</pre>

2. Nested field. Drop struct	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.
		ReplaceField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.exclude": "parent.child"
		}</pre>
			<pre>{
			  "k1": 123,
			"parent": {
			  }
			}</pre>

3. Nested field. Include field	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123",
	"k3": "234"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.
		ReplaceField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.include": "parent.child.k2"
		}</pre>
			<pre>{
			"parent": {
			"child": {
			"k2": "123"
			  }
			}</pre>

4. Nested field. Include struct	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123",
	"k3": "234"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.
		ReplaceField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.include": "parent.child"
		}</pre>
			<pre>{
			"parent": {
			"child": {
			"k2": "123",
			"k3": "234"
			  }
			}</pre>

5. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	"parent.child": {
	"k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ReplaceField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.renames": "`parent.child`.k2:field2"
		}</pre>
			<pre>{
			  "k1": 123,
			  "parent.child": {
			  "field2": "123"
			  }
			}</pre>

TimestampConverter

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	  "parent": {
	  "child": {
	  "k2": 1556204536000
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.field": "parent.child.k2",
		"transforms.smt1.format": "yyyy-MM-dd",
		"transforms.smt1.target.type": "string"
		}</pre>
			<pre>{
			  "k1": 123,
			  "parent": {
			  "child": {
			  "k2": "2014-04-25" }
			  }
			}</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	  "parent.child": {
	  "k2": 1556204536000
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.field": "`parent.child`.k2",
		"transforms.smt1.format": "yyyy-MM-dd",
		"transforms.smt1.target.type": "string"
		}</pre>
			<pre>{
			  "k1": 123,
			  "parent.child": { "k2": "2014-04-25" }
			}</pre>

ValueToKey

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "parent.child.k2"
		}</pre>
			<pre>"123"</pre>

2. Nested struct to Key.	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "parent.child"
		}</pre>
			<pre>{
			  "k2": "123"
			}</pre>

3. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	  "parent.child": {
	  "k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.fields": "`parent.child`.k2"
		}</pre>
			<pre>"123"</pre>

InsertField

Changes:

- Extend `*.field` to support nested notation.

New configurations (additional to `field.syntax.version` described above):

Name	Type	Default	Importance	Documentation
`field.on.missing.parent`	STRING	create	`MEDIUM`	Permitted values: `create`, `ignore`. Defines how to react when the field to act on does not have a parent and "field.style" is "nested". If set to "create", then the SMT will create the parent struct/map when it does not exist. If set to "ignore", then it will SMT have no effect.
`field.on.existing.field`	STRING	overwrite	`MEDIUM`	Permitted values: `overwrite`, `ignore`. Defines how to react when the field to act on already exists. If set to "overwrite", then the SMT will be applied to the existing field. If set to "ignore", then it will SMT have no effect.

Example:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	
	"parent": {
	
	"child": {
	
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "parent.child.k3"
		"transforms.smt1.static.value": "v3"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent": {
			
			"child": {
			
			"k2": "123",
			
			"k3": "v3"
			  }
			  }
			}</pre>

2. Nested field, when field names include dots	<pre>{
	  "k1": 123,
	
	"parent.child": {
	
	"k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "`parent.child`.k3"
		"transforms.smt1.static.value": "v3"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent.child": {
			
			"k2": "123",
			
			"k3": "v3"
			  }
			}</pre>

3. Nested field with the parent missing	<pre>{
	  "k1": 123,
	
	"parent": {
	
	"child": {
	
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "parent.other.k3"
		"transforms.smt1.static.value": "v3"
		}</pre>
			<pre>{
			  "k1": 123,
			
			"parent": {
			
			"child": {
			
			"k2": "123"
			  },
			
			"other": {
			
			"k3": "v3"
			  }
			  }
			}</pre>

4. Nested field with the parent missing, and ignore is set	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "parent.other.k3"
		"transforms.smt1.static.value": "v3",
		"transforms.smt1.field.on.missing.parent": "ignore"
		}</pre>
			<pre>{
			  "k1": 123,
			"parent": {
			"child": {
			"k2": "123"
			  }
			  }
			}</pre>

5. Nested field with the parent missing	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "parent.child.k2"
		"transforms.smt1.static.value": "456"
		}</pre>
			<pre>{
			  "k1": 123,
			"parent": {
			"child": {
			"k2": "456"
			  }
			  }
			}</pre>

6. Nested field with the parent missing, and ignore is set	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.InsertField$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.static.field": "parent.child.k2"
		"transforms.smt1.static.value": "456",
		"transforms.smt1.field.on.existing.field": "ignore"
		}</pre>
			<pre>{
			  "k1": 123,
			"parent": {
			"child": {
			"k2": "123"
			  }
			  }
			}</pre>

HoistField

Changes:

- Add a `hoisted` config to point to a specific path to hoist.

New configurations:

Name	Type	Default	Importance	Documentation
`hoisted`	`STRING`	<empty>	`MEDIUM`	Path to the element to be hoisted. If empty, the root struct/map is hoisted.

Examples:

scenario	input	SMT	output

1. Nested field.	<pre>{
	  "k1": 123,
	"parent": {
	"child": {
	"k2": "123"
	  }
	}
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.HoistFIeld$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.hoisted": "parent.child.k2",
		"transforms.smt1.field": "other"
		}</pre>
			<pre>{
			  "k1": 123,
			"parent": {
			"child": {
			"other": {
			"k2": "123"
			}
			  }
			}</pre>

2. Nested struct, when field names include dots	<pre>{
	  "k1": 123,
	"parent.child": {
	"k2": "123"
	  }
	}</pre>
		<pre>{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.transforms.HoistFIeld$Value", "transforms.smt1.field.syntax.version": "v2",
		"transforms.smt1.hoisted": "`parent.child`",
		"transforms.smt1.field": "other"
		}</pre>
			<pre>{
			  "k1": 123,
			"other": {
			"parent.child": {
			"k2": "123"
			  }
			}</pre>

Compatibility, Deprecation, and Migration Plan

Existing SMT configurations will not be affected by these changes as the default `field.style` is `plain` and users will need to opt-in the the new notation.

Rejected Alternatives

Keep `ExtractField` as it is and use it multiple times until reaching nested fields

This KIP proposes simplifying this configuration by replacing multiple invocations with only one nested one.

Use dots as the only separator and escape with backslashes when collides

Trying to keep only one separator, one of the alternatives is to use dots to separate; if it collides with the existing field names use backslashes "\" to represent dots that are part of the name e.g. "this.field" (which would refer to the nested field "field" under the top-level "this" field), and "this\.field" (which would refer to the field named "this.field").

However, backslashes are also used by JSON. This could lead to unfriendly configurations like "this\\\\.is\\\\.not\\\\.very\\\\.readable"

Use custom separators for edge cases

Using double dots to escape separators is another alternative to try sticking to using only dots as a field separator.

Comparing:

With double dots	With separator

`{`
`  "transforms": "cast",`
`  "transforms.cast.field.syntax.version":`
`"v2",`
`  "transforms.cast.type": "..."`
`  "transforms.cast.spec": "address..personal.country:`
`string"`
`}`
	`{`
	`  "transforms": "cast",`
	`  "transforms.cast.field.syntax.version": "v2",`
	`  "transforms.cast.field.separator": "/",`
	`  "transforms.cast.type": "..."`
	`  "transforms.cast.spec": "address.personal/country:`
	`string",`
	`}`

Even if using custom separators represents a more explicit configuration, there is always the possibility that all the separators are already included as part of the field name, leading to issues and requests for changes.

To avoid this, this KIP proposes using the approach to precede dots with another to escape itself.

Use JSONPath notation to access nested elements

JSONPath[1] was a proposed alternative to the nested notation. A drafted version of the KIP with examples using the proposed notation is outlined here: [DRAFT] KIP-821: Connect Transforms support for nested structures (JsonPath-based draft)

The following limitations were found:

- The JSONPath spec is too extensive for the use cases included in this KIP.
- A sub-set of JSONPath was proposed, but the custom spec ends up being more complex than the notation proposed here.
  - A sub-set will imply not using existing dependencies. However, adding an existing dependency would also reduce the chance of the KIP being accepted as the risk for external vulnerabilities will increase.
  - The sub-set will require users to learn JSONPath, and then what's covered and what's not by the custom implementation.

Given these cons, the KIP prefers the dotted notation.

[1] https://github.com/json-path/JsonPath

Use named styles instead of syntax versions

Was considered to use a configuration to name the styles to target fields:

- `field.style` with valid values: "plain", and "nested".

Even though this configuration is self-describing, it limits the semantics of the values.

Instead, the KIP is considering a versioned configuration "field.syntax.version" to avoid affecting current behavior and make it easier to extend by including compatible changes on the same version.

Use configuration flag per SMT instead of per-field configuration

Instead of adding a configuration under each field config, e.g. `include.syntax.version` , the KIP proposed to have a single configuration per SMT, to affect all the input fields.

Use Double-dots to escape dots included on field names

Double dot is often used in JSON Path as a descendant selector, see https://www.ietf.org/id/draft-ietf-jsonpath-base-05.html

This may confuse users. To avoid this, the backtick approach is proposed in this KIP.

Potential Improvements (out of scope)

Support Array access

Adding notation for arrays (e.g. [], or array.<offset>) to access array elements and apply SMTs to fields within the array.

This has to consider fields that could be including `[`, `]` , or numbers as part of their names and how to escape them.

Support Deep-Scan

Supported by JsonPath, could allow applying SMTs to multiple fields with the same name at different locations of the structure.

At the moment is not clear how to escape the character used for deep-scan. e.g. if using * .