[DRAFT] KIP-821: Connect Transforms support for nested structures (JsonPath-based draft)

NOTE: Rejected, see KIP-821: Connect Transforms support for nested structures

Status

Current state: Under Discussion

Discussion thread: here

JIRA: here

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

Motivation

Single Message Transforms (SMT), KIP-66, have greatly improved Connector's usability by enabling processing input/output data without the need for additional streaming applications.

Though, these benefits have been limited by SMTs limited to fields available on the root structure:

- https://issues.apache.org/jira/browse/KAFKA-7624
- https://issues.apache.org/jira/browse/KAFKA-10640

This KIP is aimed to include support for nested structures on the existing SMTs where nested structures are used.

Public Interfaces

From the existing list of SMTs, there are the following to be impacted by this change:

New configuration flags

Name	Type	Default	Importance	Documentation
`field.style`	`STRING`	`plain`	`HIGH`	Permitted values: `plain` , `nested`. Defines how to traverse a record structure to apply a transformation. If set to "`plain`", then the transformations will only apply to the elements located at the root of the message. If set to "nested", then nested elements will be affected by the transformations as well. To access nested elements, dotted notation is used. If dots are already included in the field name, then dots themselves can be used to represent dots part of the field name. e.g. to access elements from a struct/map named "same.field", the following format can be used to access its elements: "same..field.element"

These flags will be added conditionally to some SMTs, described below.

Affected SMTs

Cast

Changes:

- Extend `spec` to support nested notation.

Examples:

scenario	input	smt	output
1. Nested field.	`{ "k1": 123, "parent": { "child": { "k2": "123" } } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.Cast$Value", "transforms.smt1.spec.paths": "$[k1]:string,$[parent][child][k2]:int64" }`	`{ "k1": "123", "parent": { "child": { "k2": 123 } } }`
2. Nested field, when field names include dots	`{ "k1": 123, "parent.child": { "k2": "123" } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.Cast$Value", "transforms.smt1.spec.paths": "$['k1']:string,$['parent.child']['k2']:int64" }`	`{ "k1": "123", "parent.child": { "k2": 123 } }`

ExtractField

Changes:

- Extend `field` to support nested notation.

Example:

scenario	input	smt	output
1. Nested field.	`{ "k1": 123, "parent": { "child": { "k2": "123" } } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.ExtractField$Value", "transforms.smt1.field.path": "$[parent][child][k2]" }`	`"123"`

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$Value",
		"transforms.smt1.field": "$['parent.child'][k2]"
		}</pre>
			<pre>"123"</pre>

HeaderFrom

Changes:

- Extend `fields` to support nested notation.
- As this SMT affects only existing fields, additional configurations will not be required.

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$Value",
		"transforms.smt1.fields.pahts": "$[k1],$[parent][child]
		[k2]",
		"transforms.smt1.headers": "k1,k2"
		}</pre>
			<pre>header
			s:
			-
			k1=123
			- 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$Value",
		"transforms.smt1.fields": "$[k1],$[parent.child][k2]",
		"transforms.smt1.headers": "k1,k2"
		}</pre>
			<pre>header
			s:
			-
			k1=123
			- 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$Value",
		"transforms.smt1.fields.paths": "$[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$Value",
		"transforms.smt1.fields": "$[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$Value",
		"transforms.smt1.exclude.path": "$[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$Value",
		"transforms.smt1.exclude": "$[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$Value",
		"transforms.smt1.include": "$[parent][child][k2]"
		}</pre>
			<pre>{
			  "parent": {
			  "child":
			{
			  "k2":
			"123"
			  }
			  }
			}</pre>

4. Nested field. Include struct	{
	  "k1": 123,
	  "parent": {
	  "child":
	{
	  "k2":
	"123",
	  "k3":
	"234"
	  }
	  }
	}
		{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.
		transforms.ReplaceField$Value",
		"transforms.smt1.include": "$[parent][child]"
		}
			{
			  "parent": {
			  "child":
			{
			  "k2":
			"123",
			  "k3":
			"234"
			  }
			  }
			}

5. Nested field, when field names include dots	{
	  "k1": 123,
	  "parent.
	child": {
	  "k2":
	"123"
	  }
	}
		{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.connect.
		transforms.ReplaceField$Value",
		"transforms.smt1.renames.paths": "$[parent.child][k2]:
		field2"
		}
			{
			  "k1": 123,
			  "parent.
			child": {
			  "field2":
			"123"
			  }
			}

TimestampConverter

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	smt	output

1. Nested field.	{
	  "k1": 123,
	  "parent": {
	  "child": {
	  "k2":
	1556204536000
	  }
	  }
	}
		{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.
		connect.transforms.TimestampConverter$Value",
		"transforms.smt1.field": "$[parent][child]
		[k2]",
		"transforms.smt1.format": "yyyy-MM-dd",
		"transforms.smt1.target.type": "string"
		}
			{
			  "k1": 123,
			  "parent": {
			  "child": {
			  "k2": "2014-04-
			25" }
			  }
			}

2. Nested field, when field names include dots	{
	  "k1": 123,
	  "parent.
	child": {
	  "k2":
	1556204536000
	  }
	  }
	}
		{
		"transforms": "smt1",
		"transforms.smt1.type": "org.apache.kafka.
		connect.transforms.TimestampConverter$Value",
		"transforms.smt1.field.style": "nested",
		"transforms.smt1.field": "$[parent.child]
		[k2]",
		"transforms.smt1.format": "yyyy-MM-dd",
		"transforms.smt1.target.type": "string"
		}
			{
			  "k1": 123,
			  "parent.child":
			{ "k2": "2014-04-
			25" }
			}

ValueToKey

Changes:

- Extend `fields` to support nested notation.

Example:

scenario	input	smt	output
1. Nested field.	`{ "k1": 123, "parent": { "child": { "k2": "123" } } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.fields.paths": "$[parent][child][k2]" }`	`"123"`
2. Nested struct to Key.	`{ "k1": 123, "parent": { "child": { "k2": "123" } } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.fields.paths": "$[parent][child]" }`	`{ "k2": "123" }`
3. Nested field, when field names include dots	`{ "k1": 123, "parent.child": { "k2": "123" } }`	`{ "transforms": "smt1", "transforms.smt1.type": "org.apache.kafka.connect.transforms.ValueToKey", "transforms.smt1.fields.paths": "$[parent.child][k2]" }`	`"123"`

InsertField

Changes:

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

New configurations (additional to `field.style` 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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.static.field.path": "$[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$Value",
		"transforms.smt1.hoisted": "$[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$Value",
		"transforms.smt1.field.style": "nested",
		"transforms.smt1.hoisted": "$[parent.child]",
		"transforms.smt1.field": "other"
		}</pre>
			<pre>{
			  "k1": 123,
			  "other": {
			  "parent.
			child": {
			  "k2":
			"123"
			  }
			  }
			}</pre>

Non-affected SMTs

These SMT 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.

Proposed Changes

Nested notation

Using dots tends to be the most intuitive way to access the nested record structures, e.g. `jq` tooling already uses it[1] and will cover most of the scenarios.

Dots are already allowed as part of element names on JSON (i.e. Schemaless) records (e.g. `{'nested.key': {'val':42}}`). Instead of escaping them with backslashes, which in JSON configurations will lead to unfriendly configurations, it's proposed to follow a similar approach as CSV to escape double quotes by preceding it with the same character (double quotes in this case).

Then, for transform configuration, double dots can be used to escape existing dots that are part of the field name.

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

[2] https://datatracker.ietf.org/doc/html/rfc4180 2.7

> If double-quotes are used to enclose fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote.

Examples:

Compatibility, Deprecation, and Migration Plan

Existing SMT configurations will not be affected by these changes as the default `field.style` is `plain`, which represents the current behavior.

Rejected Alternatives

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

This KIP proposes to simplify 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 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

<pre>{
  "transforms": "cast",
  "transforms.cast.field.style": "nested",
  "transforms.cast.type": "..."
  "transforms.cast.spec": "address..personal.country:
string"
}</pre>
	<pre>{
	  "transforms": "cast",
	  "transforms.cast.field.style": "nested",
	  "transforms.cast.field.separator": "/",
	  "transforms.cast.type": "..."
	  "transforms.cast.spec": "address.personal/country:
	string",
	}</pre>

Even if using custom separators represent 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 request for changes.

To avoid this, this KIP is proposing to use the approach to precede dots with another do escape itself.

Potential KIPs

Future KIPs could extend this support for:

- Recursive notation: name a field and apply it to all fields across the schema matching that name, as proposed by

  Unable to render Jira issues macro, execution error.
- Access to arrays: Adding notation for arrays (e.g. []) to represent access to arrays and applying SMTs to fields within an array.