

# KIP-683: Add recursive support to Connect Cast and ReplaceField transforms, and support for casting complex types to either a native or JSON string

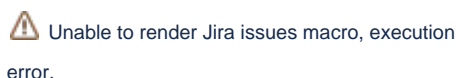
- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
  - [Cast](#)
  - [ReplaceField](#)
- [Proposed Changes](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

## Status

**Current state:** *Under Discussion*

**Discussion thread:** [here](#)

JIRA:

A screenshot of a JIRA error message. It features a yellow warning triangle icon on the left, followed by the text "Unable to render Jira issues macro, execution error." in a standard sans-serif font.

**Pull Request:** [#9493](#)

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

## Motivation

Common patterns for transforming "real-world" messages that are anything but the most simple schemas will often include a combination of using the `Cast`, `ReplaceField`, and `Flatten` transforms. However, none of these support any kind of transformation on `Map` or `Array` fields and `Flatten` does not even support them at all (the transformation will just fail with an error).

Some of this makes sense – Connect is considered a single message transporter (my own words), and single message transformations are supposed to be a 1-in and 1-out kind of a flow. Sometimes in reality, however, we will have payloads attached as children in the messages. I think KSQL is a good tool to approach this problem with on a large scale, but if your Kafka environment is so large or so small that adding KSQL for just one or few exceptions might not make sense.

I have solved some of these problem with customizations to `Cast` and `ReplaceField` transforms and I hope that the community could maybe get some value from the enhancements which I have made.

So based on my experimentation and running different scenarios with customized transforms, the key things that help to give a bit more flexibility with transforming more complicated messages are:

- Ability to `Cast` or perform `ReplaceField` operations (rename, exclude, or include) on fields which are nested as children one or more layers down.
- Ability to `Cast` the entire contents of a field which is a complex type (`Array`, `Map`, `Struct`) to a string so it can be handled downstream by something else (for example: if you are Sinking your messages to a database table, but your message is a header plus a payload of multiple `Structs` in an `Array`, then convert this array to a string and then later parse it in the database). This is a much better outcome than just not being able to handle the message at all (if you want to go to a `JdbcSink`, for example).
- And on top of this, that sometimes it is much easier to work with a "complex field as a string" if the string is in JSON format instead of the native Java object represented as a text string. So to give a setting which says how you want the complex fields to be represented as strings (JSON format vs object).
- Ability to merge a "must contain only exactly one of" different complex payloads into a common field name using `ReplaceField` (for example if each of their structures are identical or mostly identical then it is a lot less work to merge them all together before you work with them downstream).

After this, then it works quite well to chain transforms together in a pattern sort of like this (or other variations you can think of):

- `ReplaceField` to rename multiple differently-named "must contain only exactly one of" payload fields into a single field name even if they are a child of a field in the top level of the message schema.
- `Cast` these complex field types (the "payload") as a string.
- `Flatten` any other `Struct` nesting of primitive and complex types so in the end it is ready to insert into a database table.

# Public Interfaces

## Cast

New configuration parameters will be added to the `Cast` Connect transform:

- `recursive` optional boolean, default = false
- `complex.string.as.json` optional boolean, default = false

The default value is false so any existing connectors should continue to work as they did before. If a user wants to begin using these options, then they can update their connector config files or set a PUT request to the Connect Rest API for whichever connectors they desire to update.

## ReplaceField

New configuration parameters will be added to the `ReplaceField` Connect transform:

- `recursive` optional boolean, default = false

The default value is false so any existing connectors should continue to work as they did before. If a user wants to begin using these options, then they can update their connector config files or set a PUT request to the Connect Rest API for whichever connectors they desire to update.

## Proposed Changes

In order to make this change, a fair amount of the structure and flow of both transforms must be altered. Namely, that instead of just looking for fields based on the configuration or doing a loop through only the top level of the message, both the Schema update and the Value update will now need to call recursive methods which build the schema and value from top to bottom based on the type of fields that are encountered.

Also, to add this ability to cast a complex string as a JSON-formatted string, then we will need to introduce some new dependencies for the transform project. The initial thought is to use an instance of Connect's own `JsonConverter` and `JsonDeserializer` so that the usage and dependencies can be self-contained within Kafka and the code required to handle this can be significantly minimized.

In my own experimentation (and what you can see in PR [#9493](#)) it has included the following for both transforms:

- Adding Struct, Map, and Array as supported input type fields (but that they can only be traversed to find children fields underneath, or Cast in their entirety to a string).
- Removal of value assignment for Schema-less messages from the main `applySchemaless` method, into a new method `buildUpdatedSchemalessValue`, which, when `recursive=true`, recursively calls itself for any child Map or Array fields to traverse through the whole structure searching for fields to Cast or rename/exclude/include (for `ReplaceField`).
- New method `buildUpdatedSchemalessArrayValue` for handling child Arrays in Schema-less messages (same as above).
- New method `buildUpdatedSchema` which works in partnership with `getOrCreateSchema` (which I renamed to `getOrCreateUpdatedSchema` to more fit the naming and ideas which exist in the rest of the flow) which, when `recursive=true`, recursively builds the schema of any complex children as fields of different types are found. Each child schema which is built is also added to the `schemaUpdateCache` after it is built so that it can be fetched again instead of rebuilt in case it appears somewhere else in the schema or when building the value.
- New methods `buildUpdatedStructSchema`, `buildUpdatedMapSchema`, and `buildUpdatedArraySchema` to individually handle the process to build child schemas of each of these types of complex fields.
- Removal of the value assignment for Schema messages from the main `applyWithSchema` method, into a new method `buildUpdatedSchemaValue`, which, when `recursive=true`, recursively calls itself for any child Struct, Map, or Array fields to traverse through the whole structure searching for fields to Cast or rename/exclude/include (for `ReplaceField`).
- New methods `buildUpdatedArrayValue` and `buildUpdatedMapValue` to individually handle the process to build the new values of complex child fields of these types.
- New method `castToJsonString` in the `Cast` transform to handle casting a complex type field to a JSON-formatted string in case `complex.string.as.json=true`.

## Compatibility, Deprecation, and Migration Plan

The only impact should be that new functionality is added, which users must update their configuration in order to use it. Existing functionality should not be impacted and no updated should be needed in order to keep using these transforms the same as before.

## Rejected Alternatives

One possible alternative to the "recursive" idea is support for nested field selection via some kind of dotted or path-like notation in the configuration. This could potentially also be added but at this time the current proposal is to only search out child field names which match what is given in the configuration at no matter which level or how many times they appear within the schema or structure of the message.

Casting complex strings to JSON adds a bit of a variation to what the `Cast` transform is already doing, however the value which is added is quite large and makes the data much easier to use downstream if it can be given in a more standardized format like this. It could be done in other ways (such as traversing the message in a loop and use some of the Jackson classes to build new JSON objects etc) which removes the dependencies on the Kafka Connect JSON project, however I feel that this would duplicate the effort which has already been done there and it is easier to implement and support if we just use what is already available in Kafka.