

# KIP-682: Connect TimestampConverter support for multiple fields and multiple input formats

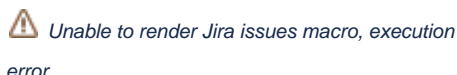
- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Changes](#)
  - [Supporting Multiple Fields](#)
  - [Supporting Multiple String Input Formats](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

## 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 blue, monospaced font.

**Proposed Pull Request:** [#11523](#)

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

## Motivation

The `TimestampConverter` transform only allows to convert one field at a time for each usage of the transform (by use of the `field` configuration parameter). But in a real environment you will often have multiple timestamps on an event (such as Created On, Last Updated On, Approved On, etc), and if you are in a position that one of them need to be converted using `TimestampConverter` then probably more than one (if not all of them) need to be transformed. For large messages which may already be going through multiple other transforms, then the performance goes down quite a bit if you end up chaining more than just a few `TimestampConverter` transforms just to catch all of the different fields.

At the same time, in the case of parsing strings to timestamps, in "real" environments it is not always possible to strictly control timestamp formats if multiple different services are producing messages to the same topic. For example, maybe some have specified a time zone and some have not, some give milliseconds, and some do not, etc. All of these variations could even be "valid" within the ISO 8601 standard but even the slightest difference in format of any event that does not match the exact specified `format` pattern will produce a failure with `TimestampConverter`. So it would be better if it was possible to give an input pattern that allowed for different variations to be parsed from string into a proper Date/Time type.

## Public Interfaces

From the perspective of using this transform in Connect, the following things will be changed:

- Change the configuration parameter `field` to be called `fields` since it will now support multiple comma-separated field names (but can support backward compatibility for some time).
- Add new configuration parameters `format.input` to allow for a pattern format which supports multiple variations to parse a string, and `format.output` to specify the exact string format to output in the case of converting from a Date/Time to a string.
- The configuration parameter `format` could possibly be removed at a later date (but remains for now for backwards compatibility), or could also be used to specify both `format.input` and `format.output` at the same time for more simple scenarios (assuming you just have a single string input format).
- As general housekeeping, the `TimestampConverter` class should also be updated at the same time to include public `ConfigName` and `ConfigDefault` interfaces instead of various public string class attributes for the configuration properties, similar to what has been done in several of the other SMTs (like `ReplaceField` for example).

## Proposed Changes

### Supporting Multiple Fields

For supporting multiple fields, we can change the `field` property from a single string to a type `ConfigDef.Type.LIST` and for clarity and consistency the property should be renamed to `fields` instead. Then when performing the translation, the code can apply the transformation to all fields in the list instead of just the one field specified in the old `field` property.

## Supporting Multiple String Input Formats

For **output** of a Date/Time field to a string, it must continue to be given in a single specific format, not in some kind of pattern. Because of this, we need to separate the format configuration parameter into two: one parameter for output **to** strings with an exact format, and one parameter for input format of strings to be parsed into the `target.type` that can support a pattern of different variations of the string-based date or timestamps.

To support this, there will be two new parameters added: `format.input` and `format.output`.

The existing `format` parameter can also remain in place to allow for configuration which will provide both the input and output formats at the same time, and work exactly as it did before this change. In this scenario, it would not support multiple different input formats (so again, the same as before). But it should not allow to set a mix of both the old and the new format parameters.

In order to support multiple input patterns the suggestion is to make the string to target type parsing use some of the features of `java.time` such as `DateTimeFormatter` instead of relying on the much older and more limited `java.text.SimpleDateFormat`. `java.time` was added in Java 8 which is the oldest version of Java supported by Kafka, so it should not add any new dependency if we wish to introduce its usage in Kafka.

The new `format.input` property will require a regular expression-like string that is compatible with the JDK's `DateTimeFormatter.ofPattern()` method. For example patterns like this would be supported: `"[yyyy-MM-dd[['T']] ]HH:mm:ss[.SSSSSSz][.SSS[XXX][X]]]"`

This example pattern would be able to support and successfully parse if there are multiple different formats in the same field, including:

- 2021-11-22
- 2021-11-22 11:19:45
- 2021-11-22T11:19:45
- 2021-11-22T11:19:45.000Z
- and more...

## Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

The transform configuration parameter `field` will be renamed to `fields` but should be done so in a way that adoption is voluntary until a major version deprecation can occur (e.g. that `field` still works and backward compatibility is maintained).

The transform configuration parameter `format` will continue to function as before (as a `SimpleDateFormat` pattern for both input and output strings) and if users wish to use the `DateTimeFormatter` input format they will need to use new input and output specific parameters `format.output` and `format.input` instead.

- *If we are changing behavior how will we phase out the older behavior?*

Existing configuration parameters and public class strings will be left as they are and continue to function as they do, but will be marked and described as deprecated and can be fully removed if and when it is appropriate in a future major release.

- *If we need special migration tools, describe them here.*

No migration tool should be necessary; if users wish to begin using the new features they will just need to update their connector configurations.

- *When will we remove the existing behavior?*

The deprecated configuration parameter `field` and the public configuration-related class strings will be removed after the next major Kafka release, based on the standard deprecation practice for the Kafka project.

## Rejected Alternatives

One initial thought was to change the entire transform from using `java.util.Date` to instead use `java.time` classes instead. However, after a bit of investigation I quickly found that since Kafka and Connect have a huge list of dependencies on dates and times being a `java.util.Date`, then it quickly became apparent that the easiest thing to do would be to focus on the core problem: parsing strings into a `Date` in a smarter way with the help of something like `DateTimeFormatter`, and then continue returning a `Date` for use by the rest of Connect.