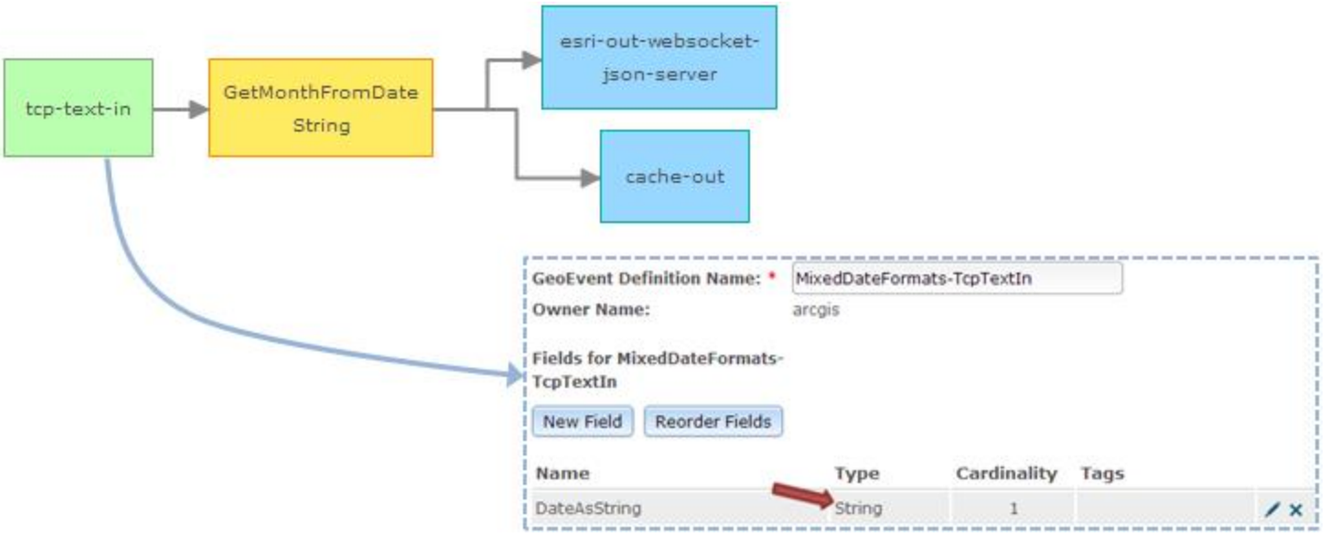
First, design a simple GeoEvent Service which incorporates a *Field Calculator* to isolate the month portion of the received date/time string and append the value, as a String, in a new field named MonthInYear.



It is important that the input, *tcp-text-in* in the illustration, rely on a GeoEvent Definition which specifies that the received date/time be handled as a String.

The illustrated GeoEvent Service could send its event data to any output. I've chosen to send the event data to a cache managed by GeoEvent Processor so I can review the event's structure as JSON – as well as a WebSocket. The WebSocket is only important if you need to give a second inbound connector a chance to process the event data it receives in order to interpret the date/time String as a Date … using an *Expected Date Format* mask. We'll look some more at this in a moment…

In the illustration above, *GetMonthFromDateString* is a *Field Calculator* processor configured with an expression which uses a Java String function *substring( )* to pull a two character substring out of the received *DateAsString* field. The *substring( )* function expects a starting index and an ending index. The index is zero-based, and given the sample input, we want to pull the two characters from positions 5 and 6 from the received string. (The function includes the character at the starting index but not the character at the specified ending index.)

GetMonthFromDateString processor properties

Resulting GeoEvent Definition Name	MixedDateFormats-GetMonthFromString
Expression	substring(DateAsString,5,7)
New Field Name	MonthInYear
New Field Type	String
New Field Tag	
Result Destination	New Field
Existing Field Name	

Sample Input:
"2014-03-06 10:44"

2 0 1 4 - 0 3 - 0 6 1 0 : 4 4
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] …

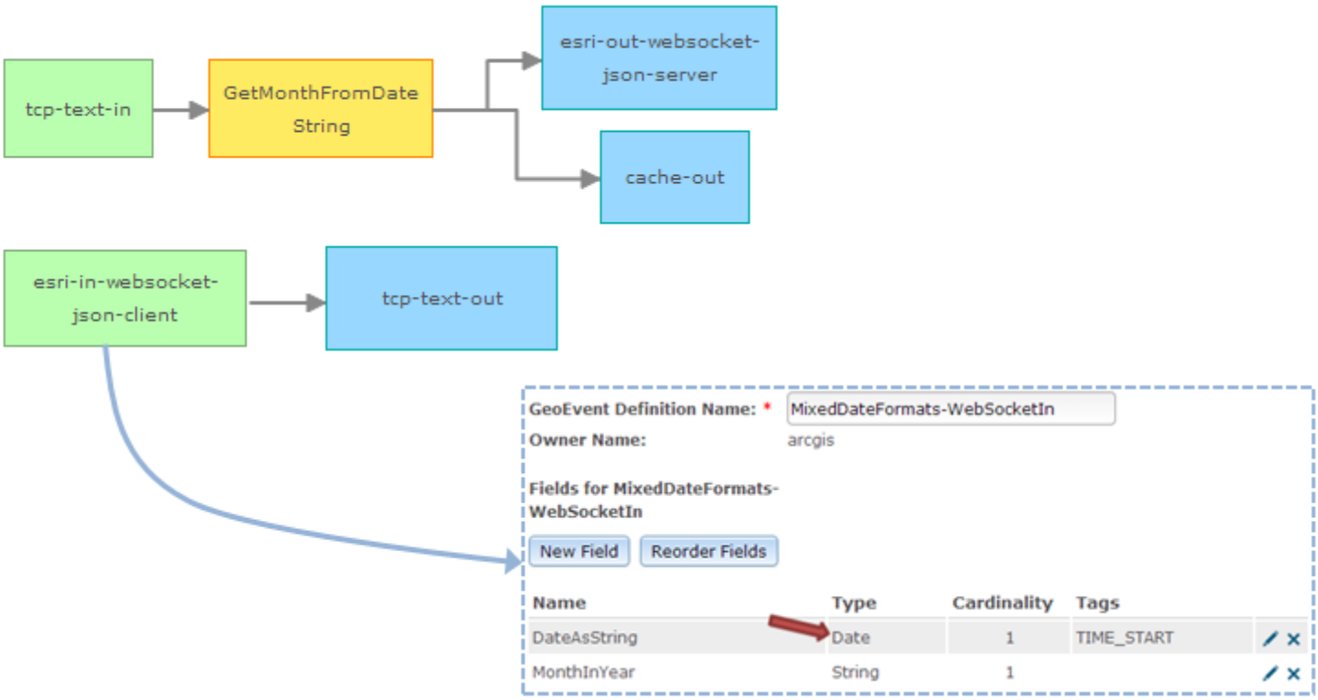
Notice that the *Field Calculator* specifies the name for the new GeoEvent Definition, created because the event schema was modified by adding a new field: *MonthInYear*. Viewed as JSON, what we get out from the GeoEvent Service above looks something like:

```
[
  - {
        DateAsString: "2014-03-06 10:44",
        MonthInYear: "03"
    }
]
```

Notice that, in the JSON illustration, the name of each field has been specified and the values appear in double-quotes (meaning that they are sent out from the GeoEvent Service as literal strings).

The second part of the problem, necessary only if you need a second input to process the date/time String and use an *Expected Date Format* mask to intrepret the string value as a Date, relies on WebSockets. I chose to use a pair of connectors from the "Working with Web Sockets" tutorial: Receive generic JSON via a WebSocket (CLIENT) and Send generic JSON to a WebSocket (SERVER).

The illustration at the top of the page sends generic JSON, illustrated above, to a "server" WebSocket so that a second input "client" WebSocket can re-ingest the event data.



Notice that you can incorporate more than one input into a single GeoEvent Service. Here I am sending JSON from the output *esri-out-websocket-json-server* to the input *esri-in-websocket-json-client*. The WebSocket input is configured to use the event defintion shown, which matches the JSON illustrated earlier with fields *DateAsString* and *MonthInYear*.

An important difference is that – when String values are received for fields which the GeoEvent Definition specifies should be Date values – the WebSocket input will use an *Expected Date Format* mask to attempt to convert the data.

esri-in-websocket-json-client input properties

Remote server web socket URI (Client Mode)	ws://localhost:6180/MixedDateFormats-WebSocket

▾ Advanced

JSON Object Name	
Expected Date Format	yyyy-MM-dd hh:mm
Create GeoEvent Definition	false
GeoEvent Definition Name (Existing)	MixedDateFormats-WebSocketIn
Construct Geometry From Fields	false