# Component Wear Anomalies

# Skinwrapper Machines

**Let's consider the Vega skinwrapper family of packaging machines by OCME**

- They work by wrapping products (bottles) in a plastic film
- ...Which is cut and heated, so that the film shrinks and stabilizes the content

# OCME Vega Shrinker

**A public dataset for a skinwrapper is available from Kaggle**

- The dataset contains a single run-to-failure experiment
- I.e. the machine was left running until its blade became unusable

**This is an example of anomaly due to component wear**

- It's a common type of anomaly
- ...And run-to-failure experiments are a typical way to investigate them

# OCME Vega Shrinker

**A public dataset for a skinwrapper is available from Kaggle**

- The dataset contains a single run-to-failure experiment
- I.e. the machine was left running until its blade became unusable

**This is an example of anomaly due to component wear**

- It's a common type of anomaly
- ...And run-to-failure experiments are a typical way to investigate them

**All problems in this class share a few properties**

- There is a critical anomaly at the end of the experiment
- The behavior becomes more and more distant from normal over time

...Meaning that they are good fit for many of the techniques we have studied

# The Dataset

## Let's have a first look at the dataset

In [3]:
```
data = pd.read_csv(data_file)
data
```

Out[3]:

| | mode | segment | smonth | sday | stime | timestamp | pCut::Motor_Torque | pCut::CTRL_Position_controller::Lag_error | pCut::CTR |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 1 | 4 | 184148 | 0.008000 | 0.199603 | 0.027420 | 62839262 |
| **1** | 1 | 0 | 1 | 4 | 184148 | 0.012000 | 0.281624 | 0.002502 | 62839262 |
| **2** | 1 | 0 | 1 | 4 | 184148 | 0.016000 | 0.349315 | -0.018085 | 62839262 |
| **3** | 1 | 0 | 1 | 4 | 184148 | 0.020000 | 0.444450 | -0.054680 | 62839261 |
| **4** | 1 | 0 | 1 | 4 | 184148 | 0.024000 | 0.480923 | -0.042770 | 62839261 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1062907** | 2 | 518 | 12 | 28 | 185909 | 8.179999 | -0.277697 | -0.023948 | 19492447 |
| **1062908** | 2 | 518 | 12 | 28 | 185909 | 8.183999 | -0.285098 | -0.022138 | 19492450 |
| **1062909** | 2 | 518 | 12 | 28 | 185909 | 8.187999 | -0.155192 | -0.034412 | 19492453 |
| **1062910** | 2 | 518 | 12 | 28 | 185909 | 8.191999 | -0.371426 | 0.031594 | 19492456 |
| **1062911** | 2 | 518 | 12 | 28 | 185909 | 8.195999 | -0.284030 | 0.010178 | 19492459 |

1062912 rows × 14 columns

- The data refers to disjoint measurement windows
- Each segment contains data sampled every 4ms

# The Dataset

## Let's check some statistics

In [4]:
```
data.describe()
```

Out[4]:

| | mode | segment | smonth | sday | stime | timestamp | pCut::Motor_Torque | pCut::CTRL_Pos |
|---|---|---|---|---|---|---|---|---|
| **count** | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 | 1.062912e+06 |
| **mean** | 2.323699e+00 | 2.590000e+02 | 5.271676e+00 | 1.654143e+01 | 1.362122e+05 | 4.102069e+00 | -1.206338e-01 | -5.472746e-05 |
| **std** | 1.649207e+00 | 1.498222e+02 | 3.505212e+00 | 8.490150e+00 | 3.226381e+04 | 2.364827e+00 | 6.078708e-01 | 1.212122e-01 |
| **min** | 1.000000e+00 | 0.000000e+00 | 1.000000e+00 | 1.000000e+00 | 8.115800e+04 | 4.000000e-03 | -6.560303e+00 | -1.888258e+00 |
| **25%** | 1.000000e+00 | 1.290000e+02 | 2.000000e+00 | 9.000000e+00 | 1.113170e+05 | 2.056000e+00 | -3.696310e-01 | -2.201461e-02 |
| **50%** | 2.000000e+00 | 2.590000e+02 | 4.000000e+00 | 1.800000e+01 | 1.348180e+05 | 4.104000e+00 | -1.187128e-01 | 6.456900e-04 |
| **75%** | 3.000000e+00 | 3.890000e+02 | 8.000000e+00 | 2.300000e+01 | 1.618270e+05 | 6.152000e+00 | 2.546913e-01 | 2.380830e-02 |
| **max** | 8.000000e+00 | 5.180000e+02 | 1.200000e+01 | 3.100000e+01 | 2.232490e+05 | 8.199999e+00 | 3.856873e+00 | 2.021531e+00 |

- The data is neither normalized nor standardized

# Missing Values

## Let's check for missing values in the columns related to sensor readings

```
In [5]: data.isnull().any()

Out[5]: mode                                                  False
        segment                                               False
        smonth                                                False
        sday                                                  False
        stime                                                 False
        timestamp                                             False
        pCut::Motor_Torque                                    False
        pCut::CTRL_Position_controller::Lag_error             False
        pCut::CTRL_Position_controller::Actual_position       False
        pCut::CTRL_Position_controller::Actual_speed          False
        pSvolFilm::CTRL_Position_controller::Actual_position  False
        pSvolFilm::CTRL_Position_controller::Actual_speed     False
        pSvolFilm::CTRL_Position_controller::Lag_error        False
        pSpintor::VAX_speed                                   False
        dtype: bool
```

No missing value in the dataset

# Acquisition Windows

## And let's check the length of each segment

In [6]:
```
data.groupby('segment')['mode'].count().describe()
```

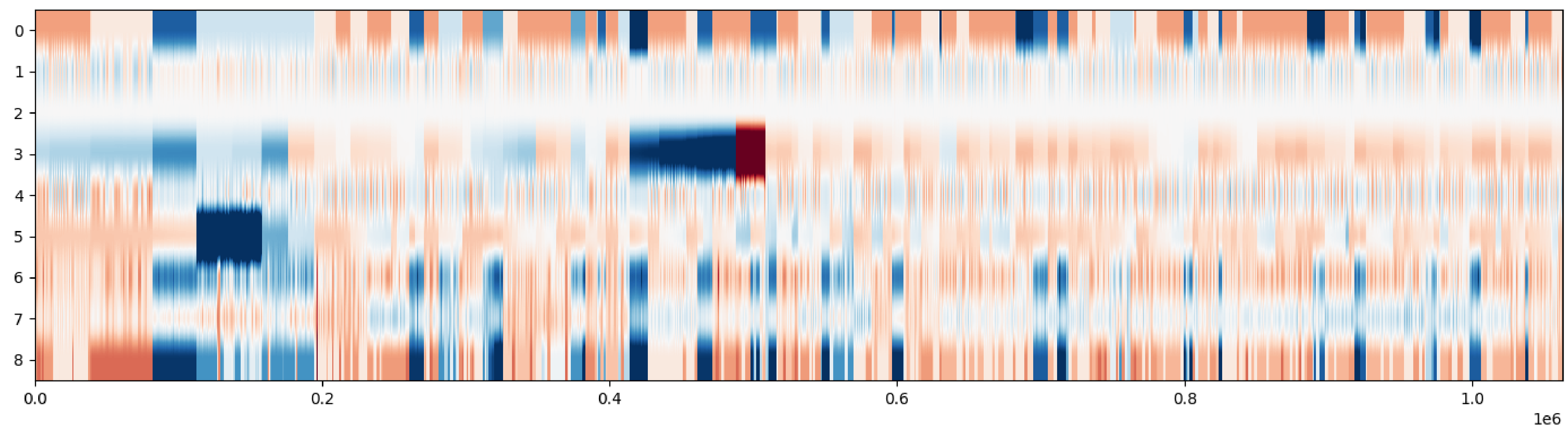
Out[6]:
```
count     519.0
mean     2048.0
std         0.0
min      2048.0
25%      2048.0
50%      2048.0
75%      2048.0
max      2048.0
Name: mode, dtype: float64
```

- There are 519 segments overall
- ...Each with 2048 samples

# Input Columns

## Let's have a look at all non-time related columns

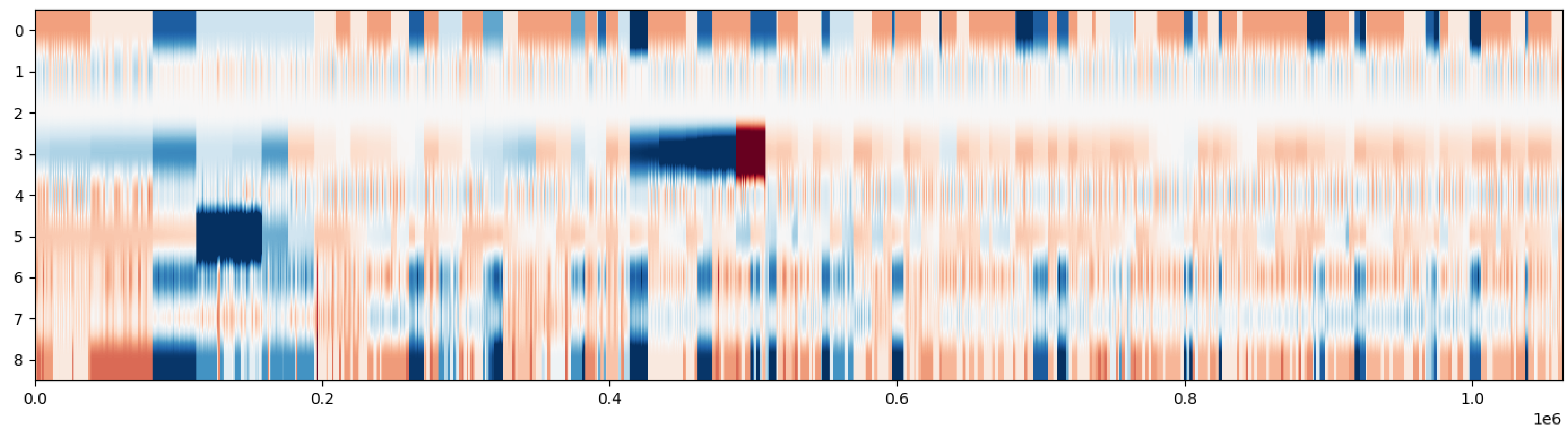
In [7]:
```
feat_in = data.columns[[0, 6, 7, 8, 9, 10, 11, 12, 13]]
data2 = data[feat_in].copy()
util.plot_dataframe((data2 - data2.mean()) / data2.std(), figsize=figsize)
```



# Input Columns

## Let's have a look at all non-time related columns

In [7]:
```
feat_in = data.columns[[0, 6, 7, 8, 9, 10, 11, 12, 13]]
data2 = data[feat_in].copy()
util.plot_dataframe((data2 - data2.mean()) / data2.std(), figsize=figsize)
```

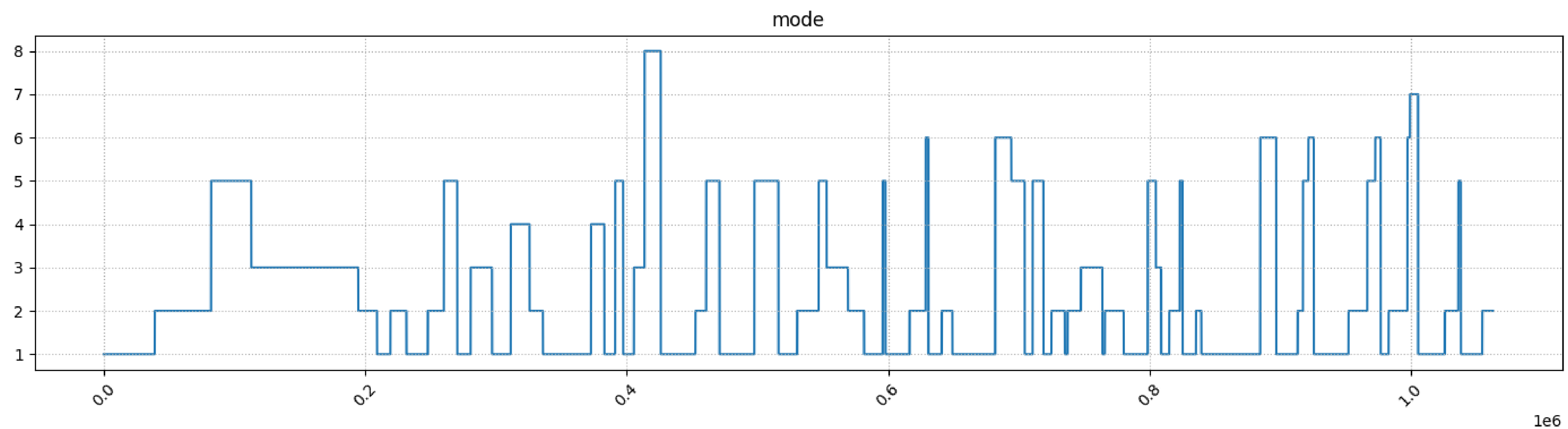



- A few features (row 0, 2, 3, 5, 8) have very "suspicious" behavior

# Input Columns

## Column 0 corresponds to an operating mode

```
In [8]: util.plot_series(data2[data2.columns[0]], figsize=figsize, title=data2.columns[0])
```

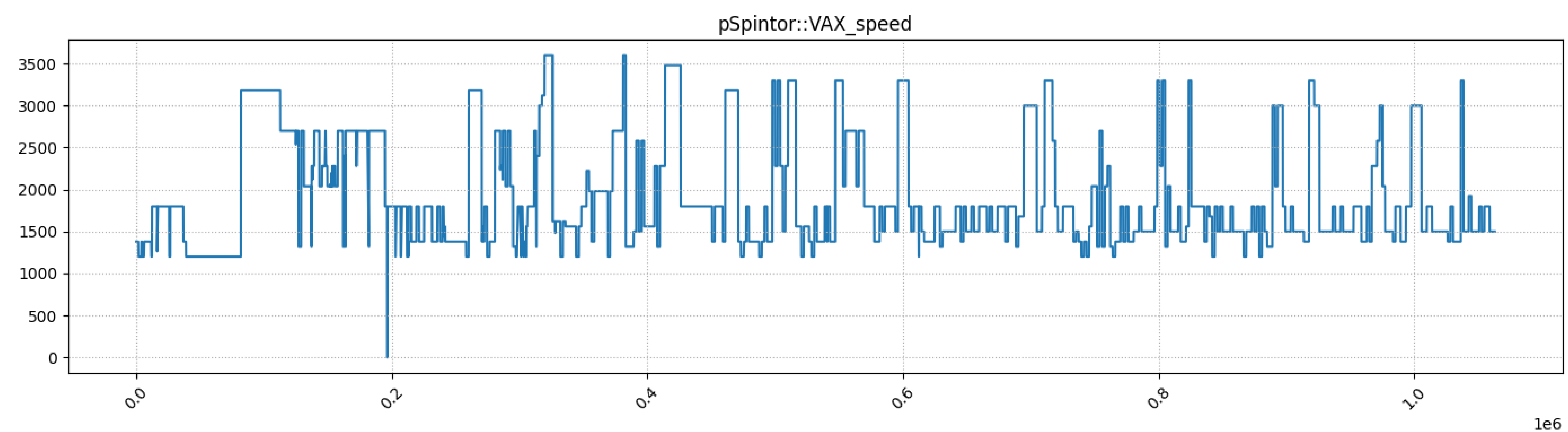



- The mode is a controlled parameter and does not change in the middle of a segment
- Intuitively, the mode has an impact on the machine behavior

# Input Columns

## Column 8 is also a fixed over long periods of time

In [9]:
```
util.plot_series(data2[data2.columns[8]], figsize=figsize, title=data2.columns[8])
```

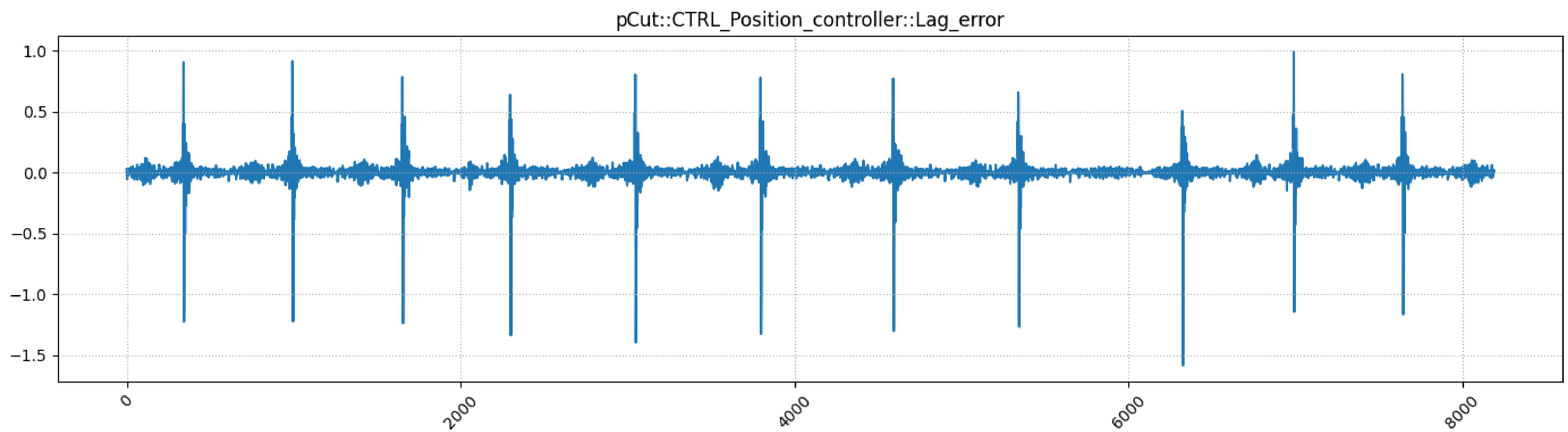



- This is likely a controlled parameter
- Ideally, we would speak with the customer (but in this excercise we can't)

# Input Columns

## Column 2 peaks repeatedly over short time periods

In [10]:
```
util.plot_series(data2[data2.columns[2]].iloc[:2048*4], figsize=figsize, title=data2.columns
```

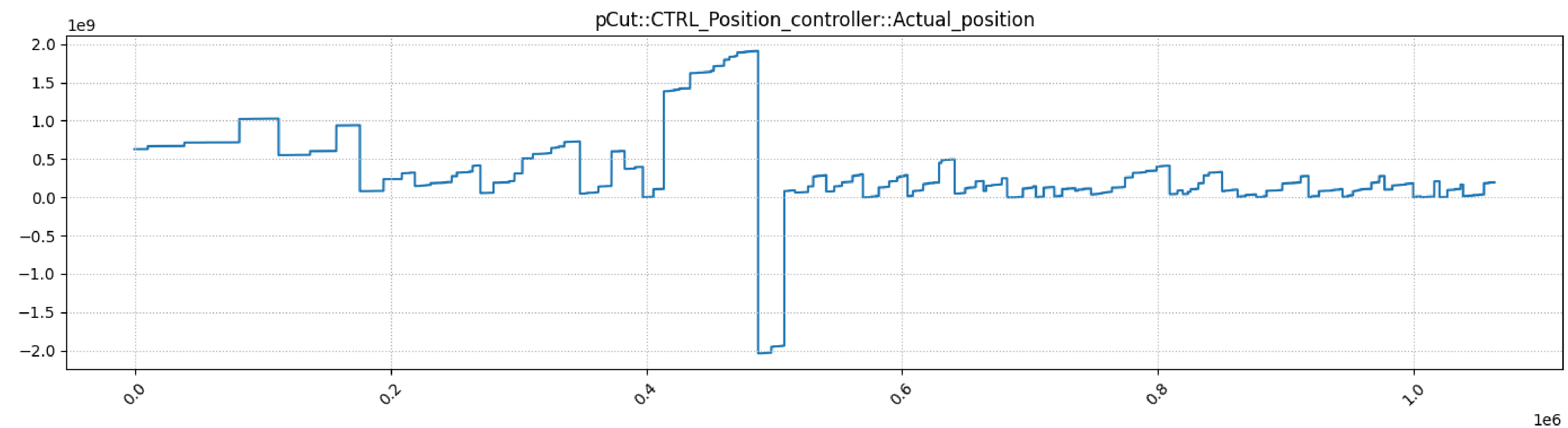



- There is nothing really wrong with this
- ...And it explains the mostly white row in our previous plot

# Input Columns

## Column 3 contains an odd, localized, deviation

```
In [11]: util.plot_series(data2[data2.columns[3]], figsize=figsize, title=data2.columns[3])
```

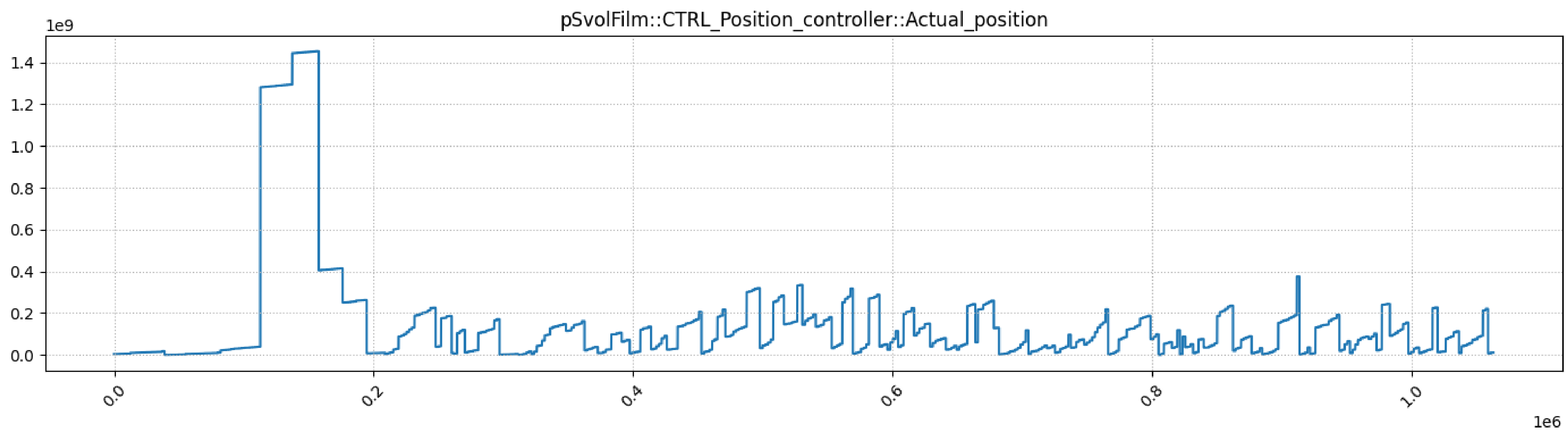



- This is likely the result of manual adjustment
- We'd better keep this column off

# Input Columns

## ...And the same holds for column 5

```
In [12]: util.plot_series(data2[data2.columns[5]], figsize=figsize, title=data2.columns[5])
```




- Again, this is probably the a mistake, or due to human intervention
- We'd better keep this column off

# Data Preparation

# Binning

**This dataset contain high-frequency data (4ms sampling period)**

- In this situation, feeding the raw data to a model does not usually make sense
- So we will use subsampling

**A binning approach typically works as follows:**

We apply a sliding window, but so that its consecutive applications do not overlap

- Each window application is called a bin
- ...From which we extract one or more features
- ...By applying different aggregation functions

**The result is series that contains a smaller number of samples**

...But typically a larger number of features

# Binning

## We will apply binning to all columns not related to time

...Except for the two we chose to discard

```
In [13]: feat_in_r = data2.columns[[0, 1, 2, 4, 6, 7, 8]]
         print(list(feat_in_r))

['mode', 'pCut::Motor_Torque', 'pCut::CTRL_Position_controller::Lag_error', 'pCut::CTRL_Position_controller::Actual_speed', 'pSvolFilm::CTRL_Position_controller::Actual_speed', 'pSvolFilm::CTRL_Position_controller::Lag_error', 'pSpintor::VAX_speed']
```

## First, we define which aggregation function to apply to each field

```
In [14]: aggmap = {a: ['mean', 'std', 'skew'] for a in feat_in_r}
         aggmap['mode'] = 'first'
         aggmap['pSpintor::VAX_speed'] = 'first'
```

- For the features that are fixed over a segment, we pick the first value

# Binning

## Then we build out bins

```
In [15]: binsize = 512 # 2 seconds of measurements
         bins = []
         for sname, sdata in data.groupby('segment'):
             sdata['bin'] = sdata.index // binsize # Build the bin numbers
             tmp = sdata.groupby('bin').agg(aggmap) # Apply the aggregation functions
             bins.append(tmp)
         data_b = pd.concat(bins)
```

- We process each segment individually
- ...So that we are sure that no bin overlaps two segments

**We chose our bin size based on:**

- Having enough data to compensate noise
- Capture regular patterns (e.g. our spiking signal)

# Binning

## Let's inspect the result

In [16]:
```
data_b
```

Out[16]:

| | mode | pCut::Motor_Torque | | | pCut::CTRL_Position_controller::Lag_error | | | pCut::CTRL_Position_controller::Actual_speed | | | pS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | first | mean | std | skew | mean | std | skew | mean | std | skew | m |
| **bin** | | | | | | | | | | | |
| **0** | 1 | -0.125718 | 0.544329 | -2.488922 | -0.000167 | 0.110956 | -3.746651 | 1599.917513 | 3999.045185 | -0.209394 | 37 |
| **1** | 1 | -0.072671 | 0.540906 | -2.635165 | -0.000567 | 0.103435 | -3.336454 | 844.540436 | 3862.165458 | 0.309415 | 36 |
| **2** | 1 | 0.014070 | 0.354287 | 0.152579 | 0.000724 | 0.031961 | 0.032216 | 60.570994 | 2816.788446 | -0.245380 | 32 |
| **3** | 1 | -0.207667 | 0.455206 | -3.803152 | -0.000150 | 0.103125 | -4.496910 | 2670.353585 | 2474.506197 | 0.155697 | 26 |
| **4** | 1 | -0.289142 | 0.434465 | -4.372388 | -0.000275 | 0.106193 | -5.242337 | 2579.037789 | 2626.919219 | -0.502959 | 30 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2071** | 2 | -0.184381 | 0.781750 | -3.758458 | 0.000508 | 0.116385 | -1.459755 | 1819.838542 | 4007.857890 | -0.029032 | 36 |
| **2072** | 2 | -0.207829 | 0.791797 | -3.968010 | -0.000285 | 0.115259 | -1.482473 | 1936.977300 | 3932.533837 | -0.082154 | 36 |
| **2073** | 2 | -0.146367 | 0.846064 | -3.683229 | -0.001013 | 0.116217 | -1.488226 | 1556.298091 | 4088.929784 | 0.137784 | 36 |
| **2074** | 2 | -0.125521 | 0.800829 | -3.798401 | -0.001139 | 0.117804 | -1.245373 | 1148.293884 | 3980.947227 | 0.422832 | 36 |
| **2075** | 2 | 0.032077 | 0.349737 | -0.137235 | -0.000274 | 0.034452 | -0.078601 | 493.331985 | 3127.137947 | 0.204754 | 33 |

2076 rows × 17 columns

- We have much fewer rows, and more columns