InfinityQS®
Quality Re-imagined

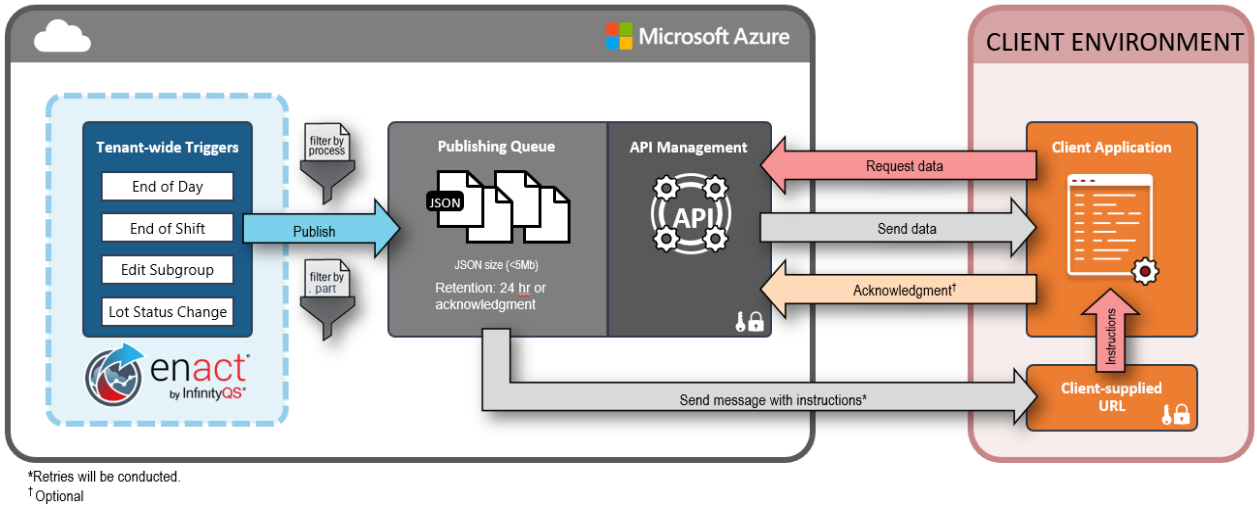
Enact KPI Publishing Integration

Contents

Overview

This document explains various aspects of Enact's Key Performance Indicator (KPI) publishing functionality, such as how users are notified of aggregated results and how to retrieve those results from Enact. This document also provides details of the content exchanged between Enact and external systems.

The Enact KPI publishing functionality uses a webhook to a Receiver Uniform Resource Locator (URL) to provide a message that aggregated results are ready to be retrieved. A user-specified application then uses these messages to call the published KPIs from Enact and write them to the desired system—which is most often a Business Intelligence (BI) dashboard. The following diagram shows the overall flow of the KPI publishing feature:



Enact KPIs are published for individual data streams composed of a part, process, and feature combination, along with a time period for the associated data stream. These aggregated KPIs are published for either daily or shift time intervals for each data stream. For example, the data stream of Net Weights for a 300g bag of potato chips from Line 1 could have KPIs published for a specific date (e.g. November 30, 2020) and a specific shift (e.g. Shift 1 beginning on November 30, 2020).

Enact KPIs are published at four different time intervals, and the messages delivered indicate what kind(s) of published results are available:

- **Daily**: Results are published after the end of each process day (after midnight based on the process time zone).
- **Shift**: Results are published 15 minutes after the end of the shift (based on the shift schedule for the individual process).
- **Lot**: Results are published after the status of a lot is changed.
- **Edits**: Edits are processed every hour, and can result in a new summary for a given date and shift. For example, a data edit performed on November 30, 2020 for data entered at 9:00 am on November 23, 2020 would result in a new daily aggregated record for November 23, 2020 and a new aggregated record for Shift 1 beginning on November 23, 2020.

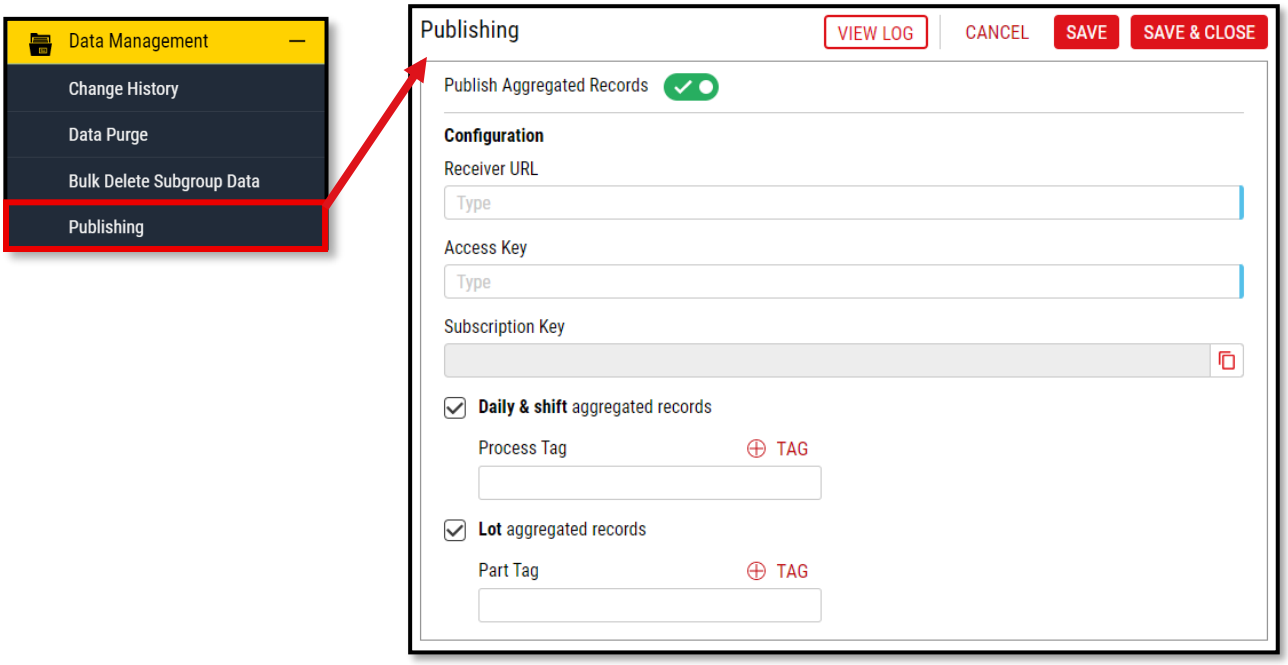
At the time of each aggregation, all data streams are combined into a single record that can be parsed. This strategy reduces the number of records that need to be retrieved.

Configuring Enact

Webhooks are one of a few ways in which web applications communicate with each other. Webhooks send real-time data from one application to another whenever a given event occurs. In Enact, this communication happens over the web through the "Receiver URL," which is provided by the receiving application.

To retrieve aggregated KPIs from Enact, an application must be created on the client side, whose Receiver URL will be provided in Enact to start getting the event messages once the KPIs for aggregated data are available to be consumed.

Note: An application capable of using webhooks should be created and accessible before configuring Enact.



- **Receiver URL**: User-specified URL to receive messages indicating KPIs have been published and are ready to be retrieved.
    - The Receiver URL must be Hypertext Transfer Protocol Secure (HTTPS)
    - This is a required field
- **Access Key**: User-specified key for the URL to verify the identity and origin of Enact messages.
    - Max length is 128 characters
    - Only the following characters are allowed: Alphanumeric [0-9 a-z A-Z], special characters $-_.!*'()
    - This is a required field
- **Subscription Key**: This is a key provided by Enact to be used by the client's application in all API calls to Enact.
- **Process Tag**: This is used to specify which processes will have KPIs published.
    - This is a required field if "**Daily & shift** aggregated records" is checked.
- **Part Tag:** This is used to specify which lots will have KPIs pubslihed.
    - This is a required field if "**Lot** aggregated records" is checked.

Once you click "Save," Enact validates the "Receiver URL" through a Validation Event Message.

Validation Event Message

Before the actual event messages are published on the webhook, there will be a registration of the Receiver URL to perform a handshake or validation step. The event contains the "Access Key" as a query string parameter in the HTTP Request Uniform Resource Identifier (URI).

A validation event message is published using HTTP OPTIONS method of API, and the schema of this validation event message is in the following format:

```
{
  "header": {
    "host": "Webhook URL",
    "max-forwards": "10",
    "webhook-request-origin": "eventgrid.azure.net",
    "webhook-request-callback": "https://rp-eastus-
eventgrid.azure.net/eventsubscriptions/validate?id=sddfw0sdsd-sddsw-rwwq-d32"
  }
}
```

The client's application listening at the Receiver URL must then acknowledge the receipt of the event by updating the return headers and returning a status code of 200—to ensure that the event message has been accepted and processed.

The following header fields are to be included in the validation request. For KPI publishing, Enact will put a validation request with "*Webhook-Request-Origin*" and "*Webhook-Request-Callback*" fields; both of these can be used for completing the validation. **However, the recommended option is to respond to the "*Webhook-Request-Origin*" with "*Webhook-Allowed-Origin.*"**

- **Webhook-Request-Origin:** This header must be included in the validation request; it contains a DNS expression that identifies the sending source, for example *Webhook-Request-Origin: enact.infinityqs.com*. To successfully complete the handshake, the target should reply to the request by returning the "*Webhook-Allowed-Origin*" header with the same value that came in the request "*Webhook-Request-Origin.*" This is the recommended method to successfully validate KPI subscribers of Enact.
- **Webhook-Request-Callback:** This optional header allows the delivery target to grant permission asynchronously, via a simple HTTPS callback. This option is generally chosen when the application does not allow custom code to be written—and validation can be done manually. If this method is used for validation, the request to callback URL must be sent within 30 seconds after the message is received.

If the validation is successful, the following message is displayed: *Aggregate publishing configuration saved successfully*. If the subscription is not successful, the following message is displayed: *Connection to the URL failed*.

Samples of applications in JavaScript and C# .Net application with authentication and validation handshake implementation can be found in **Appendix A** and **Appendix B**.

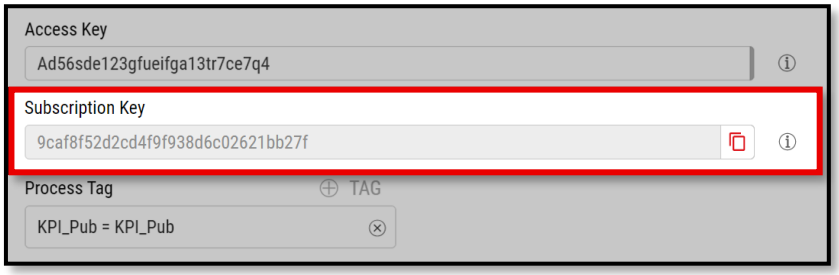
Retrieving Published KPIs from Enact

Once published KPIs are available from Enact, Enact sends a message to the Receiver URL. The body of the event message displays as shown below; the full event message can be viewed in **Appendix C**.


```
"body": {
  "subject": "kpipublisher/dfrte34/3",
  "id": "9aeb0fdf-c01e-0131-0922-9eb54906e209",
  "type": "DailyAggregationCompleted",
  "time": "2020-07-07T10:38:39.4589254Z",
  "source": "enact.infinityqs.com",
  "dataschema": "#",
  "specversion": "1.0",
  "data": {
    "aggregationTime": "2020-07-07T10:32:39.4589254Z",
    "expirationTime": "2020-07-08T10:38:39.4589254Z",
    "api": "https://api.infinityqs.com/enact/{kpi}/dcd4d685-8590-4565-be5b-
aed8399cb2c2.json"
  }
```


Upon receiving the message at the Receiver URL, the external application uses the API from the message to retrieve the published KPIs from Enact. To call the API, make a simple HTTP Get call with a header containing the *Subscription Key*, which is required to authenticate the request.

The subscription key must be passed in the header of the HTTP GET request, such as OCP-APIM-Subscription-Key: xxxx-xxxxx-xxxx-xxxxx. The value of subscription key is available in the Enact application.



Once the API GET request is placed, the API responds with the KPI information in form of a JSON. Part of the JSON is shown below; the full schema can be viewed in **Appendix D**.


```
"headers": {
  "Content-Length": 2557,
  "Request-Context": "appId=cid-v1:2cc94b9f-64fb-4388-a8f5-5a2afc3c0859"
  "Date": "Thu, 09 Jul 2020 07:29:10 GMT"
},
"body": {
  "aggregationEvent": "DailyAggregationCompleted", // Type of event, in our case it would have
different values like "DailyAggregationCompleted" or "ShiftAggregationCompleted" or
"DailyReAggregationCompleted" or "ShiftReAggregationCompleted".
  "acknowledgementPath": "https://enactapi.infinityqs.com/enact/kpi/24d03268-f13d-410b-9e6d-
f83869a28c0c?operationId=3a4c7ef83352431aa760dd7048c0f189&api-version=v1", //API url to acknowledge
aggregation data is received, will be called after receiving data.
```


To avoid abuse of these Data APIs, the APIs are throttled to accept a maximum of five (5) requests for sample event in a period of a minute.

As these API requests are HTTP requests over the internet, the client application should consider resiliency and should handle the transient errors.

Acknowledgement

You can find the API URL to acknowledge the request under *acknowledgementPath*. Calling this acknowledgement API confirms that KPIs have been retrieved—and that the data can be removed from Enact.

- Calling the acknowledgement API:
  - Make a HTTP POST request with subscription key in header.
  - The Subscription Key should be same as the KPI API request.

Although acknowledging the successful data retrieval for KPIs is optional, it is recommended. In the event that the acknowledgement API is not called by the client, aggregated data KPIs will persist for 24 hours in Enact—and can be retrieved with the same API URL.

Message Delivery & Retries

Most event message publishers provide reliable delivery. To guarantee delivery, they confirm that an event message will be delivered at least once, and they provide a retry mechanism out of the box for event messages. The event publisher continues trying to deliver event messages until the target application does not respond with a response status code, such as 200.

A publisher might send the same message more than once. For example, the message publisher may fail after sending a message, and another instance may replace the original instance—resending the same message again. There could also be other scenarios in which an event is delivered more than once to a target application. In these cases, the application should be able to make that same call repeatedly while producing the same results. Using a message identifier to check if the message has been processed already is helpful.

Understanding the JSON File

The body of the JSON file published by Enact is in the following format:

- **Aggregation Event**: *aggregationEvent* displays the type of event the KPIs are published for (*DailyAggregationCompleted*, *ShiftAggregationCompleted, DailyReAggregationCompleted*, or *ShiftReAggregationCompleted*)
- **Acknowledgement Path**: *acknowledgementPath* displays the API URL to acknowledge data retrieval
- **Metadata**: Where Enact data is contained
    - *metaData* displays every process, part, feature, shift, and tag related to the retrieval
- **Data** is where statistics are displayed for each process
    - *subgroupData* displays each part/process/feature data stream specific to a process
        - *netContentStatistics* are grouped together
        - *processCapabilityIndexes* are grouped together
    - *SamplingComplianceEvents* displays sampling compliance data for the process

References

https://github.com/cloudevents/spec/blob/v1.0/spec.md#required-attributes

List of Appendices

Appendix A. JavaScript Code Example of the Validation

```
module.exports = function (context, req) {
    context.log('JavaScript HTTP trigger function processed a request.');

    if (request.query["AccessKey"] == "AccessKeyOfWebHookProvidedInEnact") {
        //  Allow requests only if required Access Key is passed in request

        // Validation handshake request will be of HTTP OPTIONS Type
        if (req.method == "OPTIONS") {
            // If the request is for subscription validation, send back the validation code

            context.log('Validate request received');
            context.res = { status: 200 };
            context.res.headers.append('Webhook-Allowed-Origin', req.headers["WebHook-Request-Origin"]);
// Respond with Allowed Origin to be set as Request origin, to make handshake successful.
            context.res.headers.append('WebHook-Allowed-Rate', "*"); // Respond with Allowed rate to be
set with some number in case requests need to restricted.
        }
        else {
            var message = req.body;

            // The request is not for subscription validation, so it's for an event.
            // CloudEvents schema delivers one event at a time.
            // Event Message can be processed here
            var event = JSON.parse(message);
            context.log('Source: ' + event.source);
            context.log('Time: ' + event.eventTime);
            context.log('Data: ' + JSON.stringify(event.data));
        }
    }

    context.done();
};
```

Appendix B. C# Code Example of the Validation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading.Tasks;
using System.Net;
using System.Text;
using System.Net.Http;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using Microsoft.AspNetCore.Mvc;
using viewer.Models;

namespace viewer.Controllers
{
    [Route("api/[controller]")]
    public class EnactKPIController : Controller
    {
        #region Data Members

        private bool EventTypeNotification
            => HttpContext.Request.Headers["aeg-event-type"].FirstOrDefault() ==
               "Notification";

        private const string AccessKey = "AccessKeyOfWebHookProvidedInEnact";

        #endregion

        #region Public Methods
        [HttpOptions]
        public async Task<IActionResult> Options()
        {
            if (HttpContext.Request.Query["AccessKey"].Equals(AccessKey))
            {
                // HTTP OPTIONS Request for endpoint Validation handshake for CloudEvent Schema
                using (var reader = new StreamReader(Request.Body, Encoding.UTF8))
                {
                    var webhookRequestOrigin = HttpContext.Request.Headers["WebHook-Request-
Origin"].FirstOrDefault();
                    var webhookRequestCallback = HttpContext.Request.Headers["WebHook-Request-Callback"];
                    HttpContext.Response.Headers.Add("WebHook-Allowed-Origin", webhookRequestOrigin);
                }

                return Ok();
            }

            return Unauthorized();
        }

        [HttpPost]
        public async Task<IActionResult> Post()
        {
            if (HttpContext.Request.Query["AccessKey"].Equals(AccessKey))
            {
                using (var reader = new StreamReader(Request.Body, Encoding.UTF8))
                {
                    var jsonContent = await reader.ReadToEndAsync();

                    if (EventTypeNotification)
                    {
                        // Check to see if this is passed in using
                        // the CloudEvents schema
                        if (IsCloudEvent(jsonContent))
                        {
                            return await HandleCloudEvent(jsonContent);
                        }
                    }
```

```
                return BadRequest();
            }
        }

        return Unauthorized();
    }

    #endregion

    #region Private Methods

    private async Task<IActionResult> HandleCloudEvent(string jsonContent)
    {
        var details = JsonConvert.DeserializeObject<CloudEvent<dynamic>>(jsonContent);
        var eventData = JObject.Parse(jsonContent);

        // Process the event message
        await ProcessData(
            details.Id,
            details.Type,
            details.Subject,
            details.Time,
            eventData.ToString()
        );

        return Ok();
    }

    private static bool IsCloudEvent(string jsonContent)
    {
        // Cloud events are sent one at a time, while Grid events
        // are sent in an array.
        try
        {
            // Attempt to read one JSON object.
            var eventData = JObject.Parse(jsonContent);

            // Check for the spec version property.
            var version = eventData["specversion"].Value<string>();
            if (!string.IsNullOrEmpty(version)) return true;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        return false;
    }

    #endregion
}
}
```

Appendix C. Event Message Schema

```
{
  "header": {
    "content-type": "application/cloudevents+json; charset=utf-8",
    "accept-encoding": "gzip, deflate",
    "aeg-subscription-name": "NODESUBSCRIPTIONENACTINTEGRATION",
    "aeg-delivery-count": "0",
    "aeg-data-version": "1.0",
    "aeg-metadata-version": "1",
    "aeg-event-type": "Notification",
    "content-length": "611",
    "host": "294391040d8b.ngrok.io",
    "x-forwarded-proto": "https",
    "x-forwarded-for": "52.154.68.23"
  },
  "body": {
    "subject": "kpipublisher/dfrte34/3", // Subject of event, specifying the information about
event, in our case it will combination of subscription name and some key.
    "id": "9aeb0fdf-c01e-0131-0922-9eb54906e209", // Unique identifier of an event, generated by
publisher
    "type": "DailyAggregationCompleted", // Type of event, in our case it would have different
values like "DailyAggregationCompleted" or "ShiftAggregationCompleted" or
"DailyReAggregationCompleted" or "ShiftReAggregationCompleted".
    "time": "2020-07-07T10:38:39.4589254Z", // Date time of event generation in UTC
    "source": "enact.infinityqs.com", // Source of the event
    "dataschema": "#", // Schema specification of data object.
    "specversion": "1.0", // Specification version of cloud event
    "data": {
      "aggregationTime": "2020-07-07T10:32:39.4589254Z", // UTC Date Time of aggregation run, for
which this event was raised.
      "expirationTime": "2020-07-08T10:38:39.4589254Z", // UTC Date Time at which data for this
aggregation event will expire and will no longer be accessible.
      "api": "https://api.infinityqs.com/enact/{kpi}/dcd4d685-8590-4565-be5b-aed8399cb2c2.json" //
API url to get data for the aggregation event, which will be called on receiving this event
    }
  }
}
```

Appendix D. JSON Data Response Schema of Daily, Shift, or Subgroup Edit aggregation


```
{
  "headers": {
    "Content-Length": 2557,
    "Request-Context": "appId=cid-v1:2cc94b9f-64fb-4388-a8f5-5a2afc3c0859"
    "Date": "Thu, 09 Jul 2020 07:29:10 GMT"
  },
  "body": {
    "aggregationEvent": "DailyAggregationCompleted", // Type of event, in our case it would have
different values like "DailyAggregationCompleted" or "ShiftAggregationCompleted" or
"DailyReAggregationCompleted" or "ShiftReAggregationCompleted".
    "acknowledgementPath": "https://enactapi.infinityqs.com/enact/kpi/24d03268-f13d-410b-9e6d-
f83869a28c0c?operationId=3a4c7ef83352431aa760dd7048c0f189&api-version=v1", //API url to acknowledge
aggregation data is received, will be called after receiving data.
    "metadata": {
      "processes": [
        { // Details of processes for which data has been published, Process name with hierarchy, process
identifier, tag identifiers associated with processes.
          "id": 1,
          "name": "Company|Division|Region|Site|Department|Process",
          "tagValues": [ 1, 3]
        }
      ],
      "parts": [
        { // Details of parts for which data has been published, Part name with revision appended (if
applicable), part identifier, tag identifiers associated with parts.
          "id": 1,
          "name": "Part 1",
          "tagValues": []
        }
      ],
      "features": [
        { // Details of features for which data has been published, Feature name, Feature identifier, tag
identifiers associated with features.
          "id": 1,
          "name": "Density",
          "featureType": "Variable",
          "tagValues": [2]
        }
      ],
      "shifts": [
        { // Details of shifts for which data has been published, Shift name, Shift identifier,
applicable only when event type is ShiftAggregationCompleted or ShiftReAggregationCompleted.
          "id": 1,
          "name": "Morning Shift"
        }
      ],
      "tags": [
        { // Details of Tags associated with Part, Process or Feature for which aggregated data is
published, name of tag group and corresponding tag values along with identifier.
          "id": 1,
          "name": "color",
          "values": [
            {
              "id": 1,
              "name": "Red"
            },
            {
              "id": 2,
              "name": "Green"
            }
          ]
        },
```

```
        {
          "id": 2,
          "name": "time",
          "values": [
            {
              "id": 3,
              "name": "Morning"
            },
            {
              "id": 4,
              "name": "Evening"
            }
          ]
        }
      ]
    },
    "data": [
      {
        "processId": 1, // Process identifier of aggregated data
        "subgroupData": [
          {
            "summaryDate": "2020-07-09T12:59:00Z", // Date when data was collected for the process
               "partId": 1, // Part Identifier of aggregated data
            "featureId": 1, // Feature identifier of aggregated data
            "shiftId": 1, // Shift identifier of aggregated data (applicable only in case shift
aggregation events)
            "subgroupCount": 1, // Number of subgroups aggregated
            "pieceFeatureCount": 3, // Number of pieces aggregated
            "defectCount": 0, // Number of defects
            "defectiveCount": 6, // Number of defectives
            "oosCount": 8, // Number of pieces out of specification limits
            "mean": 7, // Mean of piece values
            "sdLongTerm": 1, // Long Term standard deviation
            "sdShortTerm": 1, // Short Term standard deviation
            "cqsTotal": 5, // Composite quality score, derived from OOS, Defects and Defectives
            "cqsPpm": 5, // Composite quality score, per million

            "netContentStatistics": {
              "mavUpperCount": 0, // Number of piece values above MAV upper NCC limit
              "mavLowerCount": 0, // Number of piece values below MAV lower NCC limit
              "t1UpperCount": 0, // Number of piece values above T1 upper NCC limit
              "t1LowerCount": 0, // Number of piece values below T1 lower NCC limit
              "t2UpperCount": 0, // Number of piece values above T2 upper NCC limit
              "t2LowerCount": 0, // Number of piece values below T2 lower NCC limit
              "devLscMav": 2, // Deviation of LSC MAV (Process Mean - LSC)
                 "devLsct1t2" 0 // Deviation of LSC T1 T2(Process Mean - LSC)
            },

            "processCapabilityIndices": {
              "pp": 1.39769554620138, // Process performance index
              "ppk": 1.06135758420235, // Process performance index
              "ppm": 0, // Process performance index
              "cp": 1.00436581543935, // Process capability index
              "cpk": 0.76267773652657, // Process capability index
              "cpm": 0 // Process capability index
            }
          }
        ],

        "samplingComplianceEvents": [
          {
            "processId": 1, // Process identifier of aggregated data
            "shiftId": 1, // Shift identifier of aggregated data (applicable only in case shift
aggregation events)
```

```
            "summaryDate": "2020-07-09T12:59:00Z", // Date when data was collected for the process
               "subgroupEvents": {
              "due": 2, // Number of subgroup data collections expected
              "on-time": 3, // Number of subgroup data collections succeeded
              "late": 1, // Number of subgroup data collections that were late
              "miss": 0 // Number of subgroup data collection that were missed
            },

            "checklistEvents": {
              "due": 2, // Number of checklist data collections expected
              "on-time": 3, // Number of checklist data collections succeeded
              "late": 1, // Number of checklist data collections that were late
              "miss": 0 // Number of checklist data collection that were missed
            }
          }
        ]
      }
    ]
  }
}
```

Appendix E. JSON Data Response Schema of Lot aggregation


```
{
  "headers": {
    "Content-Length": 2557,
    "Request-Context": "appId=cid-v1:2cc94b9f-64fb-4388-a8f5-5a2afc3c0859"
    "Date": "Thu, 16 June 2021 07:29:10 GMT"
  },
  "body": {
    "aggregationEvent": "LotAggregationCompleted", // Type of event, in our case it would have different
values like "LotAggregationCopmleted" or "LotReAggregationCompleted".
    "acknowledgementPath": "https://enactapi.infinityqs.com/enact/kpi/24d03268-f13d-410b-9e6d-
f83869a28c0c?operationId=3a4c7ef83352431aa760dd7048c0f189&api-version=v1", //API url to acknowledge
aggregation data is received, will be called after receiving data.
    "metadata": {
      "parts": [
        { // Details of parts for which data has been published, Part name with revision appended (if
applicable), part identifier, tag identifiers associated with parts.
          "id": 1,
          "name": "Part 1",
          "tagValues": [
            9
          ]
        }
      ],
      "features": [
        { // Details of features for which data has been published, Feature name, Feature identifier, tag
identifiers associated with features.
          "id": 1,
          "name": "Density",
          "featureType": "Variable",
          "tagValues": [2]
        }
      ],
      "tags": [
        { // Details of Tags associated with Part or Feature for which aggregated data is published, name
of tag group and corresponding tag values along with identifier.
          "id": 1,
          "name": "color",
          "values": [
            {
              "id": 1,
              "name": "Red"
            },
            {
              "id": 2,
              "name": "Green"
            }
          ]
        },
        {
          "id": 2,
          "name": "Publishing Group Tag",
          "values": [
            {
              "id": 9,
              "name": "Lot Publishing Tag"
            }
          ]
        }
      ],
      "lots": [
        { // Details of lots for which data has been published, Lot name, Lot identifier, tag identifiers
associated with lots.
```

```
          "id": 1,
          "name": "Lot A",
          "tagValues": []
        }
      ]
    },
    "data": [
      {
        "lotId": 1, // Lot identifier of aggregated data
        "partRevisionId": 1, // Part identifier of aggregated data
        "lotStartTime": "2021-06-16T19:01:43Z", // Lot start time for the aggregated data
        "lotEndTime": "2021-06-16T19:33:22Z", // Lot end time for the aggregated data
        "subgroupData": [
          {
            "featureId": 1, // Feature identifier of aggregated data
            "pieceFeatureCount": 3, // Number of pieces aggregated
            "defectCount": 0, // Number of defects
            "defectiveCount": 2, // Number of defectives
            "oosCount": 4, // Number of pieces out of specification limits
            "mean": 7, // Mean of piece values
            "sdLongTerm": 1, // Long Term standard deviation
            "capabilityIndexes": {
              "pp": 1.69769554620138, // Process performance index
              "ppk": 1.56135758420235, // Process performance index
            }
          }
        ]
      }
    ]
  }
}
```