

---

**S&P Global**

Market Intelligence

# Data Delivery Producer API

**Submitting data to Data Delivery**

**06 December 2023**

**Copyright © 2023 by S&P Global Market Intelligence, a division of S&P Global Inc. All rights reserved.**

These materials, including any software, data, processing technology, index data, ratings, credit-related analysis, research, model, software or other application or output described herein, or any part thereof (collectively the “Property”) constitute the proprietary and confidential information of S&P Global Market Intelligence or its affiliates (each and together “S&P Global”) and/or its third-party provider licensors. S&P Global on behalf of itself and its third-party licensors reserves all rights in and to the Property. These materials have been prepared solely for information purposes based upon information generally available to the public and from sources believed to be reliable.

Any copying, reproduction, reverse-engineering, modification, distribution, transmission or disclosure of the Property, in any form or by any means, is strictly prohibited without the prior written consent of S&P Global. The Property shall not be used for any unauthorized or unlawful purposes. S&P Global Market Intelligence’s opinions, statements, estimates, projections, quotes and credit-related and other analyses are statements of opinion as of the date they are expressed and not statements of fact or recommendations to purchase, hold, or sell any securities or to make any investment decisions, and do not address the suitability of any security, and there is no obligation on S&P Global Market Intelligence to update the foregoing or any other element of the Property. S&P Global Market Intelligence may provide index data. Direct investment in an index is not possible. Exposure to an asset class represented by an index is available through investable instruments based on that index. The Property and its composition and content are subject to change without notice.

THE PROPERTY IS PROVIDED ON AN “AS IS” BASIS. NEITHER S&P GLOBAL NOR ANY THIRD PARTY PROVIDERS (TOGETHER, “S&P GLOBAL PARTIES”) MAKE ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, FREEDOM FROM BUGS, SOFTWARE ERRORS OR DEFECTS, THAT THE PROPERTY’S FUNCTIONING WILL BE UNINTERRUPTED OR THAT THE PROPERTY WILL OPERATE IN ANY SOFTWARE OR HARDWARE CONFIGURATION, NOR ANY WARRANTIES, EXPRESS OR IMPLIED, AS TO ITS ACCURACY, AVAILABILITY, COMPLETENESS OR TIMELINESS, OR TO THE RESULTS TO BE OBTAINED FROM THE USE OF THE PROPERTY. S&P GLOBAL PARTIES SHALL NOT IN ANY WAY BE LIABLE TO ANY RECIPIENT FOR ANY INACCURACIES, ERRORS OR OMISSIONS REGARDLESS OF THE CAUSE. Without limiting the foregoing, S&P Global Parties shall have no liability whatsoever to any recipient, whether in contract, in tort (including negligence), under warranty, under statute or otherwise, in respect of any loss or damage suffered by any recipient as a result of or in connection with the Property, or any course of action determined, by it or any third party, whether or not based on or relating to the Property. In no event shall S&P Global be liable to any party for any direct, indirect, incidental, exemplary, compensatory, punitive, special or consequential damages, costs, expenses, legal fees or losses (including without limitation lost income or lost profits and opportunity costs or losses caused by negligence) in connection with any use of the Property even if advised of the possibility of such damages. The Property should not be relied on and is not a substitute for the skill, judgment and experience of the user, its management, employees, advisors and/or clients when making investment and other business decisions.

The inclusion of a link to an external website by S&P Global should not be understood to be an endorsement of that website or the website’s owners (or their products/services). S&P Global is not responsible for either the content or output of external websites. S&P Global keeps certain activities of its divisions separate from each other in order to preserve the independence and objectivity of their respective activities. As a result, certain divisions of S&P Global may have information that is not available to other S&P Global divisions. S&P Global has established policies and procedures to maintain the confidentiality of certain nonpublic information received in connection with each analytical process. S&P Global may receive compensation for its ratings and certain analyses, normally from issuers or underwriters of securities or from obligors. S&P Global reserves the right to disseminate its opinions and analyses. S&P Global Ratings’ public ratings and analyses are made available on its sites, [www.spglobal.com/ratings](http://www.spglobal.com/ratings) (free of charge) and [www.capitaliq.com](http://www.capitaliq.com) (subscription), and may be distributed through other means, including via S&P Global publications and third party redistributors.

The S&P Global logo is a registered trademark of S&P Global, and the trademarks of S&P Global used within this document or materials are protected by international laws. Any other names may be trademarks of their respective owners.

# Contents

<b>3. Getting started: try it out!</b> .....	<b>6</b>
3.1. Creating a new Producer API account.....	6
3.2. Accessing information on rejected data and batch status.....	6
3.3. Notifications about service downtime.....	7
<b>Using the Data Delivery Producer API</b> .....	<b>8</b>
4. API Keys.....	8
5. Entity as a resource.....	9
6. Intraday and batch updates.....	9
7. Post Entity.....	10
8. Deleting data.....	12
8.1 Deleting one or more instances.....	12
8.2 Deleting time series points for one or more instances.....	12
8.3 Deleting with entitlements.....	12
8.4 Deleting as part of a batch run.....	13
9. Purging data.....	13
10. Bulk updates.....	14
11. Starting, ending and cancelling a batch run.....	14
12. Binary attachments.....	16
13. HTTP response codes.....	17
14. Error response.....	18
<b>Data Dictionary</b> .....	<b>19</b>
How dictionaries drive N6 .....	19
Default data dictionary JSON format.....	19

# Preface

## 1. Introduction

The Data Delivery platform provides for the reliable and prompt distribution of data to a large number of clients. Data Delivery distributes the data through three main channels: through Websites, through Files and through an API. Data Delivery also distributes data into other platforms such as Reuters and Bloomberg.

The Producer API is the primary means to get data into Data Delivery for onward distribution. The Producer API provides a simple programming model to submit, update or delete data on the Data Delivery platform. The data can be grouped into batches if appropriate for that data set. Once the data is on the platform it can then be distributed to downstream clients through Files, API, Web or other means.

In addition to the Producer API described here, Data Delivery can also ingest data from files through its D6 service. For example, if a data set is already available as a scheduled file, D6 can consume that file and add the data to Data Delivery without the need for direct interaction with the Producer API. For more information on the D6 service, contact the Data Team at [data.delivery@ihsmarkit.com](mailto:data.delivery@ihsmarkit.com). Internally, D6 calls the Producer API on behalf of the data provider.

The Producer API is accessed via the Internet on the following URLs:

PROD: <https://feed.ihsmarkit.com>

QA: <https://feed-qa.ihsmarkit.com>

DEV: <https://feed-dev.ihsmarkit.com>

**Note:** internal URLs ending "datadelivery.info" accessible via the Intranet, also exist, but these are for the internal use of Data Delivery only.

## 2. Data ownership

The N6 Producer API can be used to create, update and delete data on the N6 platform. The data product retains ownership of the data on N6 and must use the Producer API to create, update or delete data as needed on the N6 platform.

## Conventions

The following typographical conventions are used in this guide:

- `Monospace` is used for code extracts, file names and any text that must be entered into form fields.
- **Bold** is used for user interface labels and buttons, and for emphasis in blocks of code.
- **Colored text** is used for links to other parts of this guide and for links to Web resources.
- *Italic* is used within text for emphasis, and to indicate words used literally.
- Italic monospace is used in sections of code to indicate variable placeholders. For example, the following command uses *optionName* as a variable argument to the ant command:

```
ant -DoptionName ...
```

**Note:** Paragraphs marked with *Note*, *Important*, *Caution*, and *Tip* discuss significant conditions. Take note of this information before continuing.

## Contact us

At S&P Global Market Intelligence, we understand the importance of accurate, deep and insightful information. We integrate financial and industry data, research and news into tools that help track performance, generate alpha, identify investment ideas, perform valuations and assess credit risk. Investment professionals, government agencies, corporations and universities around the world use this essential intelligence to make business and financial decisions with conviction.

S&P Global Market Intelligence is a division of S&P Global (NYSE: SPGI), the world's foremost provider of credit ratings, benchmarks and analytics in the global capital and commodity markets, offering ESG solutions, deep data and insights on critical business factors. S&P Global has been providing essential intelligence that unlocks opportunity, fosters growth and accelerates progress for more than 160 years. For more information, visit [www.spglobal.com/marketintelligence](http://www.spglobal.com/marketintelligence).

By e-mail: [support@ihsmarkit.com](mailto:support@ihsmarkit.com)

On the Web: <https://www.spglobal.com/en/enterprise/about/contact-us.html>

## 3. Getting started: try it out!

The **demo** namespace provides a simple Price type. This namespace is available in our DEV environment as a playground area for anyone interested in trying out the Producer API to send or receive data from the N6 platform.

You can use the following shared account to publish or request demo data:

Username: `markit/resellers/API_OPS/accounts/demo.dv`

Password: `Example@N6`

Download the *Producer API Sample* to get started with the API:

<https://confluence.ihsmarkit.com/download/attachments/163273367/demo.zip?version=3&modificationDate=1570453073000&api=v2>

The sample is a set of HTML forms that submit demo data in the format required by the API.

The code to submit data to the demo namespace is also available in Python;

<https://confluence.ihsmarkit.com/download/attachments/163273367/feed.zip?version=2&modificationDate=1586869826000&api=v2>

Further samples for the Consumer API are available here:

<https://confluence.ihsmarkit.com/display/MDP/N6+Python+Samples>

### 3.1. Creating a new Producer API account

New accounts for the Producer API are created by the Data Team as part of dictionary creation: [mk-datadeliverydataoperations@ihsmarkit.com](mailto:mk-datadeliverydataoperations@ihsmarkit.com)

Note that on QA and PROD, you must provide your source IP addresses for whitelisting as part of securing the Producer API accounts. The "ihsmarkit.com" URLs are only accessible via the Internet, which means the IP addresses you must provide cannot be `10.*.*.*` addresses as these are not internet-routable. Instead we require your NATted IP addresses.

### 3.2. Accessing information on rejected data and batch status

Information about rejected Producer API requests and the status of batch runs can be found here: <https://feed.dev.datadelivery.info/namespaces>

<https://feed.qa.datadelivery.info/namespaces>

<https://feed.prod.datadelivery.info/namespaces> To gain access to these URLs, contact [data.delivery@ihsmarkit.com](mailto:data.delivery@ihsmarkit.com).

### 3.3. Notifications about service downtime

DEV environments can be down at any time without any notification.

Notifications about QA Main environment downtime are sent to **MK-DD API Service Maintenance Notifications**. Raise a Service Now request to be added to this list. Agnes Clarke is approver for additions to the mailing list.

Any downtime of the PROD and UAT environments are handled by the formal client communication procedure owned by Client Services.

# Using the Data Delivery Producer API

## 4. API Keys

All Producer API services require a valid session represented by an API key. Use the Authentication Service to obtain an API key.

The API key returned by the service must be used in all subsequent API calls. The API key remains valid until one of the following events:

- No request using the API key is made for over an hour.
- The total number of currently open sessions exceeds the session limit configured for your account. A new session created in excess of the session limit invalidates the oldest currently open session.
- Sessions can be invalidated after a maximum period of time by the Producer API server. In such cases, invoke the authentication service with your credentials to obtain a new key.

The Authentication Service is invoked by a POST request. The required `username` and `password` parameters should be passed in the message body with the `Content-Type` header set to `application/x-www-form-urlencoded`.

**Note:** Applications should not rely on an API key remaining valid and should instead work on the assumption that an API key can be invalidated at any time. When an API key is invalidated, the correct response is to create a new API key and retry any failed requests with the new key.

**Note:** *Note:* We recommend sending `username` and `password` in the message body only. For enhanced security, future versions of the Producer API service may disable submission of `username` and `password` in the URL query string.

- DEV URL - <https://feed-dev.ihsmarket.com/apikey>
- QA URL - <https://feed-qa.ihsmarket.com/apikey>
- PROD URL - <https://feed.ihsmarket.com/apikey>
- Method: POST
- Request parameters:
  - `username` - User name or contact email of the API user. (mandatory)
  - `password` - Password. (mandatory)

Response status codes:

- 200 OK - Request completed successfully.
- 400 BAD\_REQUEST - The request is not valid, such as an invalid username or password.
- 403 FORBIDDEN - Not entitled for Producer API service.
- 5xx INTERNAL\_SERVER\_ERROR - The service encountered an error. Retry the request. If the error persists, contact [support@markit.com](mailto:support@markit.com).



To close a session, invoke the Authentication Service logout URL by POST to invalidate the API key:

- URL: `${url}/apikey/logout`
- Method: `POST`
- Request parameters:
  - `apikey` - API Key to be invalidated.
  - `format` - Format of the response message. XML, HTML and JSON are supported values.

### Adding the API key to Producer API requests

The API key must be included with all requests to the Producer API, in the form of an Authorization header:

```
Authorization: Bearer apikey
```

For example, the following Python POST request includes the required header:

```
requests.post(url, data=postData, headers={'Authorization': 'Bearer {}'.format(apikey)})
```

## 5. Entity as a resource

The URL for an entity instance consists of a path with the namespace, entity name and entity ID:

```
/{namespace}/{Entity}/{id}
```

For example: `/cbspd/Instrument/979491`

Namespaces, entity names and ID formats are defined in the data dictionary for the data set.

HTTP PUT, POST and DELETE methods on this URL can be used to submit or remove instances of the entity. Entity data is exchanged with the server in a JSON document. The JSON document format is defined by the [Default data dictionary JSON format](#).

**Note:** The Producer API servers do not support the GET method. To view submitted data, it is necessary to invoke the GET method for a given resource on the Consumer API servers.

## 6. Intraday and batch updates

Data can be sent to the iProducer API server in two forms, either as batch data or as intraday updates.

- Batch data is data that is published in batches, namely a set of updates linked to a particular condition like an "N1600" batch. A data dictionary for batch data always includes Batch and BatchRun entities:
  - Batch defines the batch schedule. For example, there would be a Batch record for the "N1600" batch.
  - BatchRun defines the actual instance of a batch. For example, there would be an instance of BatchRun for "N1600" on each day it runs.
- Intraday data is not organized into batches. Instead data is submitted as and when it becomes available.

There are some small differences between the form parameters for intraday and batch POST and PUT requests, but the main difference is in the sequence of URLs called during the update process:

Update type	Description
Intraday updates	Simply submit data with the following POST requests: /{namespace}/{Entity}
Batch updates	At least three distinct POST requests are required: <ol style="list-style-type: none"> <li>1. Start the batch run: /{namespace}/BatchRun/{batchRunId}/Events Set eventType=Start. The dictionaryVersion and batchId form parameters are also required.</li> <li>2. Submit data: /{namespace}/{Entity} This step can be repeated as many times as required. <b>Note:</b> Include the batchRunId form parameter, in addition to the data, dictionaryVersion and requestId fields.</li> <li>3. Complete the batch run: /{namespace}/BatchRun/{batchRunId}/Events Set eventType=End. The dictionaryVersion and batchId form parameters are also required.</li> </ol>

## 7. Post Entity

Use the HTTP method POST to send a JSON array of entity instances to the Producer API server.

In the case of an intraday or 'live' dataset, if the instances pass validation and other checks, the records are published and made available on the Consumer API servers and in files.

In the case of batched namespaces, the POST requests for a batch run must be preceded by a START batch run event and followed by an END batch run event.

**Important:** The following limits apply to requests:

Limit	Value	Explanation
Maximum compressed request payload	5MB	The total size of the payload of a message may not exceed 5MB after compression. MongoDB imposes a limit of 16MB, but 12MB is set as the limit to account for different compression ratios.
Maximum entity record size	10MB	None of the entity records in a message may exceed 10MB (in uncompressed JSON form).
Maximum number of records per request		There is no hard limit for number of records but it is recommended to keep individual messages to 1000 records or less. Higher throughput would be achieved by sending three parallel requests of 1000 records each, rather than one message with 3000 records. Do not exceed 5,000 records per request in any case.
Max parallel HTTPS requests	20	The limit is counted per namespace.

For the special case of submitting a single record to the Producer API server, you can use the PUT method (or alternatively simply POST an array of one record). When using PUT, it is required that the entity ID in the path must match the entity ID in the JSON document.

PUT: `/{namespace}/{Entity}/{id}` (deprecated)

POST: `/{namespace}/{Entity}`

Content-Type: `multipart/form-data`

Form Parameter	Required	Description
<code>data</code>	Yes	Dictionary data to be published. The format is defined by the <code>dataFormat</code> key.
<code>dictionaryVersion</code>	Yes	The version of the dictionary used to format the data
<code>dataFormat</code>		Format of the data. Supported formats: JSON, JSON_GZIP. The default is JSON.
<code>messageType</code>		<b>Important:</b> Deprecated. This parameter is ignored by the REST Inbound servers. Type of the message (INSERT, UPDATE, MERGE). The default is MERGE.
<code>batchRunId</code>		The identifier of the batchRun in case this request is part of a batch run. This key should not be set in case of intraday data.
<code>requestId</code>	Yes	Request Id for tracking purposes. The <code>requestId</code> must be unique. In the case of batch updates, the <code>requestId</code> must be unique within the batch, while within intraday updates the <code>requestId</code> must be globally unique.
<code>mode</code>		Publication mode. Supported values for intraday data are: NORMAL, BACKFILL. The default is NORMAL. Data sent in batch runs can be sent in additional modes: RESTATEMENT and ADJUSTMENT. For more information, see <i>Raising Batch Run Events</i> below. Set BACKFILL when publishing historical intraday updates. BACKFILL requests are processed with lower priority than NORMAL requests. When mode is set to BACKFILL multiple batch runs for the same batch can be published in parallel. If mode is NORMAL, only one batch run per batch can be published at any time (but even when mode is NORMAL batch runs for <b>different batches</b> can be published in parallel).
<code>updateLastValueSnapshot</code>		<b>Important:</b> Only set this parameter if requested to do so by the data delivery team. If set, last value snapshot will be updated. Default it true. This parameter only applies to intraday updates.
<code>publishToMessageBus</code>		<b>Important:</b> Only set this parameter if requested to do so by the data delivery team. If set, messages will be forwarded on the message bus. Default it true. This parameter only applies to intraday updates.

## 8. Deleting data

**Important:** Namespaces are configured with a `waitForMergeDone` property. If this is set to `false`, the DELETE is executed asynchronously. This can lead to issues in cases where data is first deleted then reinserted. Synchronous behavior can be forced on DELETE requests by setting an additional parameter `waitForMergeDone=true`. Setting this parameter will cause slower responses from the Producer API.

**Important:** A related namespace configuration is `allowDeletesInBackground`: this configuration causes asynchronous deletes to be discarded if they are followed by a fresh inserts for the same id value. However, where `allowDeletesInBackground` is set, the namespace does not support delete operations with a time range.

To configure `waitForMergeDone` Or `allowDeletesInBackground` on your namespace, contact API Operations.

### 8.1 Deleting one or more instances

**Note:** For performance reasons, DELETE is not enabled for namespaces that do not routinely require this feature. Contact Data Delivery API Operations to enable this feature for your namespace.

The following URL deletes the entities with the given IDs from the N6 /latest and /timeseries services - the records are not available in snapshots or in the timeseries database. The data remains available in the /stream API (if streaming is configured for this namespace).

```
DELETE: /{namespace}/{Entity}?id={1},{2}...
```

For example, `/cbspd/Instrument?id=979491,979492...`

### 8.2 Deleting time series points for one or more instances

The following URL deletes the entities with the given IDs from the N6 platform for a given time range only - the entities are still available in snapshots and in the timeseries database, if instances of the entities occur outside of the given time range. If the time range encompasses the record currently in the /latest collection, then the /latest collection is updated with the most recent instance that has not been deleted. References to the IDs will remain in the streaming API (if it is configured for this namespace).

The `from` and `to` parameters of the delete request are applied to the timeseries key of the specified entity.

```
DELETE: /{namespace}/{Entity}/?id={1},{2}&from={ISO8601}&to={ISO8601}
```

For example, `/cbspd/Instrument?`

```
id=979491,979492&from=2015-01-16T23:55:55+02:00Z&to=2015-01-16T23:58:55+02:00Z
```

### 8.3 Deleting with entitlements

Delete events are visible to Consumer API clients regardless of entitlements. This exposes the IDs of the deleted records to all clients. In cases where this is not desirable, it is possible to delete records with

entitlements applied. However, to apply the entitlements, the Deleted records must be republished as part of the delete operation.

POST: `/{namespace}/{entity}/delete`

Content-Type: multipart/form-data

**Note:** To delete with entitlements, a HTTP POST method is used, **not** a HTTP DELETE.

Required parameters:

- `data` - the complete instances to delete, with all fields required for entitlements. Mandatory fields are not enforced on Delete data, but if entitlement fields change in the future, any deletes without new entitlement fields will not be served to clients.
- `dataFormat` - the format of the data parameter, either JSON or JSON\_GZIP.

Optional parameters:

- `requestId` - if no request ID is provided, the Producer API generates one and returns to this to the client in the response. The client should log the request ID for error tracing.
- `from` and `to` - to apply a time range to the delete operation. If `from` and `to` are omitted, the given records are deleted entirely from the `/latest` and `/timeseries` collections. If `from` and `to` are not omitted, `/timeseries` records falling in the time range are deleted, and the `/latest` collection is updated with the most recent record if available. If `from` or `to` are omitted, that limit on the time range is considered unbounded.

## 8.4 Deleting as part of a batch run

Add parameter `batchRunId=BATCH_RUN_ID` to include the DELETE message in an open batch run.

- The batch run START event for that batch run ID must already have been sent, and the END and CANCEL events have not yet been sent.
- When the batch run END event is sent, the DELETE messages that are part of the batch run are applied first, and then the UPDATE messages are applied.
- The delete message can delete any data in the given namespace.

**Note:** Only HTTP DELETE-based delete operations can be included in a batch run. *Deleting with entitlements* (see 8.3) is not supported for batch runs.

**Note:** Setting the parameter `clearBatchRunsWithSameAsOf` to true will cause a batch run to delete records sent on previous batch runs (for that specific batch) with the same `asOf`.

## 9. Purging data

A DELETE request (see section 10) removes data from the `/latest` and `/timeseries` collections of the Consumer API, but the given data remains in the `/stream` API. To completely remove data from all services including `/stream`, use the Purge API:

POST: `/{namespace}/{Entity}/purge`

**Note:** Purge requests use HTTP POST not HTTP DELETE.

Apply the following parameters:

- `requestId` - optional
- `from` - optional parameter (omitting `from` is the same as setting `from` to the beginning of time)
- `to` - optional parameter (omitting `to` is the same as setting `to` to the end of time)
- `fieldName=values` (where `fieldName` could be any field on the entity such as `id`, `isin` and so on).

**Note:** `from` and `to` are applied to the stream timestamp ("at" field in the Consumer /stream API).

## 10. Bulk updates

It is possible to update the contents of the /latest and /timeseries collections in bulk operations.

**Note:** The content of the /batchrun and /stream services is unchanged after a bulk update.

**Note:** Bulk updates are not enabled on the Producer API except on a case-by-case basis. Contact API Operations at [data.delivery@ihsmarkit.com](mailto:data.delivery@ihsmarkit.com) to discuss if you need this feature enabled. Configuration on the Producer API side is required to enable the feature, including temporarily enabling the feature and defining indexes to support the operation.

Bulk update requests have the form:

```
POST: /{namespace}/{Entity}/bulk_updates
```

Bulk update message bodies have fixed structure, being a JSON document with fields for the *filter* (to identify the data to modify), and the *update* (listing the fields to update in matching documents). The document also specifies whether to update /latest (snapshot) data and or /timeseries data.

```
[ { "filter": { "isin": ["XS1185955306"] }, "update": { "issuerShortName": "Japan2", "midPrice": 125.9 }, "applyOnSnapshot": true, "applyOnTimeseries": true }, { "filter": { "isin": [ "8DWHRMAM30YE" ] }, "update": { "issuerShortName": "Test3 Issue", "markitTicker": "markitTicker2", "midPrice": 124.9 }, "applyOnSnapshot": true, "applyOnTimeseries": true } ]
```

**Note:**

- Multiple filter conditions can be specified; a target document must match all of these conditions (logical AND).
- It is not possible currently to update fields in an array of structs.

## 11. Starting, ending and cancelling a batch run

```
POST: /{namespace}/BatchRun/{batchRunId}/Events
```

For example, /cbspd/BatchRun/S0800-2015-10-01-1443658658916/Events.

Parameters are sent as `application/x-www-form-urlencoded`.

Form Parameter	Send on START	Send on END	Send on CANCEL	Required	Description
dictionaryVersion	Yes	Yes	Yes	Yes	The version of the dictionary used to format the data
eventType	Yes	Yes	Yes	Yes	The status of the BatchRun: Start, End, Cancel.
batchId	Yes	Yes	Yes	Yes	The Batch for which this BatchRun is an instance.
asOf	Yes			Yes	The date and time when the BatchRun is calculated. <b>Note:</b> Required for Start event; omit this field for the End or Cancel event.
runDate	Yes				DateTime at which the batch run is scheduled to be sent. This corresponds to the scheduling defined in the Batch entity.
recordCount		Yes			The total number of records over all entities sent in the BatchRun. The recordCount will only be populated for Completed BatchRuns. It will be empty when a BatchRun starts.
version	Yes				The version of the BatchRun. The first run of a BatchRun will have a version of 1. The version will be incremented with 1 for every rerun.
mode	Yes				Publication mode. Supported values: NORMAL, BACKFILL, RESTATEMENT, ADJUSTMENT. The default is NORMAL. <ul style="list-style-type: none"> <li>Set BACKFILL when publishing historical batch runs.</li> <li>Set RESTATEMENT when an existing batch needs to be republished. RESTATEMENT is identical to NORMAL except that the mode=RESTATEMENT makes explicit that a batch run is being rewritten. Set the incrementFrom field on the BatchRun close event.<i>Note: For N6 Files, RESTATEMENT triggers file delivery.</i></li> <li>Set ADJUSTMENT when making partial updates to a previously published batch run. Set the incrementFrom field on the BatchRun close event.<i>Note: For N6 Files, ADJUSTMENT does not trigger file delivery.</i></li> </ul> <b>Important:Important:</b> If multiple NORMAL batch runs are started for the same batchId, the newer runs will cancel the older runs. It is not possible to send NORMAL batch runs for the same batchId in parallel. This rule does not apply to BACKFILL, RESTATEMENT and ADJUSTMENT batch runs.
clearBatchRunsWithSameAsOf	Yes	Yes			Set to true so that a batch run removes all data from previous batch runs sharing the same asOf. This setting is useful for situations where batch run has fewer IDs than a previous run, as setting clearBatchRunsWithSameAsOf=true will cause the orphaned IDs from the previous batch runs to be removed.
incrementFrom	Yes	Yes			Use in combination with mode ADJUSTMENT to specify the previous batchRunId being modified.
updateLastValueSnapshot					<b>Important:</b> Only set this parameter if requested to do so by the data delivery team.

Form Parameter	Send on START	Send on END	Send on CANCEL	Required	Description
					If set, last value snapshot will be updated. Default it true. For historic updates, this value can be set to false.
publishToMessageBus					<b>Important:</b> Only set this parameter if requested to do so by the data delivery team. If set, the batch run will not be published on the N6 internal message bus.

## 12. Binary attachments

Unstructured data (such as PDFs) can be attached to structured dictionary content using the N6 Attachment Service.

A sample attachment can be downloaded from this URL:

<https://api-dev.ihsmarkit.com/demo/Price/latest?id=ID000223399&format=HTML&limit=3>

The Python code to upload the sample is as follows. Note the sample sets a mimetype on the multipart attachment upload - this mimetype will also be set when the attachment is downloaded.

```
import requests
import json
import time
import math
import urllib.parse
import http.client

r = requests.post("https://feed-dev.ihsmarkit.com/apikey", data={'username': 'markit/resellers/API_OPS/
accounts/demo.dv', 'password': 'Example@N6'})
print("POST /apikey", r.status_code, r.reason)
apikey = r.text
print(apikey)

t = math.floor(time.time() * 1000)
requestId = 'requestId' + str(t)
data = '''[
  {
    "id":"ID000223399",
    "asOfDateTime":"2021-04-01T09:50:14.047Z",
    "attachmentId":"producerApiSample2",
    "attachmentName":"producerApiSample.zip"
  }
]'''
r = requests.post(" https://feed-dev.ihsmarkit.com/demo/Price", files=dict(foo='bar'),
  headers={'authorization', 'Bearer {}'.format(apikey)},
  data={'dictionaryVersion': 1,
        'dataFormat': 'JSON',
        'requestId': requestId,
        'data': data
      })
jsonvar = json.loads(r.text)
print(r.status_code, r.text)

t = math.floor(time.time() * 1000)
requestId = 'requestId' + str(t) + 'attachment'
```



```

path="https://feed-dev.ihsmarket.com/demo/Price/attachmentId"
zipPath='C:\\dev\\python\\producer_api_sample\\feed_with_attachment.zip'
mime='application/zip'
fileName='producerApiSample.zip'
files = {'producerApiSample2': (fileName, open(zipPath, 'rb'), mime)}
r = requests.post(path, files=files, data={'requestId': requestId}, headers={'Authorization': 'Bearer
{}'.format(apikey)})
print(r.status_code, r.text)

```

The dictionary fields used by this sample (Price.attachmentId and Price.attachmentName) are defined here: <https://dictionarymanager.dev.datadelivery.info/dictionaries/demo/published/Price>

name	Price				
description	The price of an instrument.				
tags					
timeseriesKeyExtension					
fields					
	name	tags	type	description	version
	id		string(30)*	Unique identifier for the price.	1
	asOfDateTime	timeSeries	dateTime*	The date and time of the price.	1
	attachmentName	attachmentName	string(200)	Name of attachment.	1
	attachmentId	attachment	string(50)	Id of an attachment.	1
	cusip	matchIdentifier	string(9)	Unique CUSIP identifier for the instrument to which the price data applies.	1
	askPrice		decimal	The ask price of the instrument.	1
	bidPrice		decimal	The bid price of the instrument.	1
	billingCode		string(20)	The field indicates the billing code of the security. It is used to determine the correct billing for the valuation of the security.	1
	midPrice		decimal	The mid price of the instrument.	1
	type		string(20)	The asset class that the instrument belongs to Agency Pools.	1

10 items

## 13. HTTP response codes

The categories of HTTP responses can be handled as follows:

HTTP return code	Description
200	The message has been successfully published to data delivery. In case of the inbound API, this means the data has been stored in the database and published on the bus
4xx	The request could not be understood by the server due to malformed syntax or incorrect parameters. The client should not repeat the request without modifications.
5xx	The request failed due to an internal server error and should be retried. In case of repeated failure, contact <a href="mailto:data.delivery@markit.com">data.delivery@markit.com</a> .

The following specific HTTP response codes may be returned:

Return Code	Error Code	Message template
200		
400	INVALID_API_KEY	The API key \$input is not valid.
	INVALID_PARAMETER	Invalid parameter \$input.
	INVALID_PARAMETER_VALUE	Parameter \$parameter has an invalid value \$input. \$details.
	MISSING_PARAMETER	Parameter \$parameter not provided.

Return Code	Error Code	Message template
	INVALID_JSON_DATA	Data field is not valid JSON. \$details.
	INVALID_DICTIONARY_DATA	Data field does not comply to the dictionary. \$details.
	BATCH_RUN_IS_NOT_OPEN	Batch run \$input was not started.
	BATCH_RUN_ALREADY_CANCELLED	Batch run \$input was already cancelled.
	BATCH_RUN_ALREADY_CLOSED	Batch run \$input was already closed.
	BATCH_ALREADY_RUNNING	Batch run \$batchrun is already running for batch \$batch.
404	NAMESPACE_DOES_NOT_EXIST	Namespace \$input does not exist. Available namespaces: \$namespaces.
	TYPE_DOES_NOT_EXIST	Type \$input does not exist in namespace \$namespace. Available types: \$types.
408	REQUEST_TIMEOUT	The server timed out in trying to read the request
429	TOO_MANY_REQUESTS	Too many parallel requests. Maximum: \$maxRequests.
500	INTERNAL_FAILURE	Internal failure, please contact support for details.
503	SERVICE_UNAVAILABLE	Service is temporary unavailable. Please try to access it later.

## 14. Error response

In case of an error, a JSON response is provided for diagnostic purposes:

Property	Description
requestId	The requestId of the failing message.
errorMessage	Message describing the error
fragment	Fragment of the input causing the problem (if available)
index	Index of the element causing the problem (if available)
errorCode	A code classifying the type of error.

# Data Dictionary

The N6 platform revolves around the data dictionary concept. The data dictionary is used to describe a data set: what high level entities does a data set contain, what fields does each entity have and how do these entities relate to each other. As a very simple example, a dictionary could consist of the entities *Instrument* and *Price*. The instrument could contain fields like *isin*, *cusip* and *maturityDate* and the price could contain a *askPrice*, *bidPrice* & *midPrice*. Additionally, the price would then contain a field called *Instrument*, defining for which Instrument this Price was and linking back to the underlying instrument.

To distinguish between data sets which may define *Instrument* and *Price* differently, each dictionary is given a namespace (a short character code) to uniquely identify which data set an entity belongs to.

## How dictionaries drive N6

The N6 platform is designed to utilize the information described in the data dictionary. Using the data dictionary, we generate our data store model on the fly when uploading a dictionary to N6, allowing data to be captured immediately after upload has been completed without requiring code or database changes (as long as a dictionary doesn't introduce new concepts of course that are not currently covered). Similarly, after the dictionary has been uploaded the customer facing N6 API dynamically exposes the data as described in the dictionary, generating customer on boarding documentation directly of the dictionary as well (hence sample value and proper description are important). Finally, our file platform will expose the fields and entities as described in the dictionary in the file configuration console dynamically based on the dictionary upon upload, allowing file configurators to setup templates and customer facing files without development needs.

## Default data dictionary JSON format

This section describes the structure of the default data dictionary JSON format, such as is used by Data Delivery Consumer API.

### Objects in a data dictionary

- Namespace - all data dictionaries have a unique namespace, like "bondref" or "csbpd".
- Entities - data fields are organized into entities, such as Instrument and Price. Entities always have an "id" field that uniquely identifies each instance of the entity (i.e. a primary key).
- Structs - structs are like entities but they do not have a primary key.
- Fields - essentially structs and entities are just lists of data fields.

### Data dictionary JSON structure

A data dictionary JSON document consists of a JSON array of entity instances. The type of the entity and the namespace of the dictionary are specified by the URL in the REST API.

**Note:** The dictionary namespace and the entity name do not occur in the JSON document.

**Note:** As the REST API is organized by namespace and entity (`/namespace/EntityName`), JSON documents do not contain a mix of namespaces nor a mix of different entity types at the top level of the document.

Each instance in the document is a JSON object, which is a map of String field names and values. For example, if *Instrument* has fields *isin* (String) and *id* (string), a resulting JSON document with two instances would appear similar to the following:

### (1) Example 1

```
[
  {
    "isin": "US1234567890",
    "id": "1234567"
  },
  {
    "isin": "CA0987654321",
    "id": "7654321"
  }
]
```

**Note:** A JSON value MUST be an object, array, number, or string, or one of the following three literal names: `false`, `null`, or `true`.

### References to other entities

If an entity has a field that references another entity, then the key fields of the referenced entity are nested inside the element for that field. For example, if we add a *benchmark* field to *Instrument* that references another instrument, and *id* is the primary key of *Instrument*, then the XML would look like this:

### (2) Example 2

#### Structs

If a field references a struct, all the fields of the struct are nested inside the referencing field.

#### Fields with cardinality greater than one

Fields that can be lists of values (i.e. the dictionary defines them with cardinality greater than one) are represented by placing the values in a JSON array. For example, *instrumentRelation* is a list of two structs:

### (4) Example 4

#### Optional fields

If a field is optional and it has no value, then omit the element from the document. For example, if *benchmark* and *instrumentRelation* in (4) Example 4 are optional and have no value, the document would look similar to the following:

### (5) Example 5

#### Dates and dateTimes

Dates and dateTimes should be sent as Strings in ISO8601 format. For example, a date should be sent as 1999-08-23. A dateTime should be sent as 2015-01-16T23:55:55+02:00. Always include a timezone for date times, as specified in the ISO standard.

**Note:** More specifically, dates and dateTimes must be sent in the subset of ISO8601 used by the W3C XML Schema Recommendation.

It is also possible to send Date-Time fields as Unix timestamps (in millis). These are evaluated in UTC time.

