S&P Global
Market Intelligence

N6 Producer API Compact Mode

User Guide

27 June 2025

Contents

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 several channels like Files on SFTP, an API, BigQuery, Snowflake and XpressFeed. 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, BigQuery, Snowflake, XpressFeed or other means.

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.

The Producer API is available in two modes, the *Compact* mode and the *Extended* mode. All new datasets are onboarded to the *Compact* mode, which has the essential features of the API and is designed for maximum compatibility with downstream systems. The *Extended* mode has additional features, at the cost of compatibility with fewer downstream systems.

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.

By e-mail: support@ihsmarkit.com

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

1 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

1.1 3.1. Creating a new Producer API account

New accounts for the Producer API are created as part of a Business Engagement and Dictionary request, please refer to Data Dictionary Request Form.

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.

1.2 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@spglobal.com

1.3 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.

2 Using the Data Delivery Producer API

2.1 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 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.ihsmarkit.com/apikey
- QA URL - https://feed-qa.ihsmarkit.com/apikey
- PROD URL - https://feed.ihsmarkit.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.

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)})
```

2.2 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.

2.3 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.
- A dataset can contain a mix of intraday and batch data.

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: 1. Start the batch run: `/{namespace}/BatchRun/{batchRunId}/Events` Set `eventType=Start`. The `dictionaryVersion` and `batchId` form parameters are also required. 2. Submit data: `/{namespace}/{Entity}` This step can be repeated as many times as required. **Note:**Include the `batchRunId` form parameter, in addition to the `data`, `dictionaryVersion` and `requestId` fields. 3. Complete the batch run: `/{namespace}/BatchRun/{batchRunId}/Events` Set `eventType=End`. The `dictionaryVersion` and `batchId` form parameters are also required.

2.4 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.
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.

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

Content-Type: multipart/form-data
```

Form Parameter	Required	Description
data	Yes	Dictionary data to be published. The format is defined by the dataFormat key.
dictionaryVersion	Yes	The version of the dictionary used to format the data
dataFormat		Format of the data. Supported formats: JSON, JSON_GZIP. The default is JSON.
messageType		**Important:**Deprecated. This parameter is ignored by the REST Inbound servers. Type of the message (INSERT, UPDATE, MERGE). The default is MERGE.
batchRunId		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.
requestId	Yes	Request Id for tracking purposes. The requestId must be unique. In the case of batch updates, the requestId must be unique within the batch, while within intraday updates the requestId must be globally unique.
mode		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 *Raising Batch Run Events* below. Set BACKFILL when publishing historical intraday updates. BACKFILL requests are processed with lower priority than NORMAL requests.

2.5 8. Deleting data

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.

```
POST: /{   namespace}/{entity}/delete-by-key
```

Content-Type: multipart/form-data

Note:Note: A HTTP POST method is used, **not** a HTTP DELETE.

Required parameters:

- `data`- a JSON array of objects, with each object containing the entire timeseries key to delete. For example, if an entity to be deleted has an asOf field as the timeseries key, then the delete data would look similar to the following:[{"id":"id1"; "asOf":"2024-05-01"},{"id":"id1"; "asOf":"2024-05-02"},{"id":"id1"; "asOf":"2024-05-03"}]
- `dataFormat`- the format of the data parameter, either JSON or JSON_GZIP.

Note:Note: The `data` parameter must specify the exact timeseries keys to delete. it is not possible to provide a range.

Note:Note: If an entity does not have a timeseries, then only provide the "id" field in each object.

Note:Note: If an entity has a multifield timeseries key, all the fields must be provided in each object in the `data`array.

2.6 9. 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. **Note:**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.

Form Parameter	Send on START	Send on END	Send on CANCEL	Required	Description
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. ● Set BACKFILL when publishing or republishing historical batch runs. **Important:Important**: RESTATEMENT and ADJUSTMENT can be used to republish previously published batch runs. However, only use RESTATEMENT and ADJUSTMENT modes if advised to do so by data.delivery@spglobal.com In general, BACKFILL is the strongly preferred mode for publishing historical data. ● Set RESTATEMENT when an entire 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. *Note: For N6 Files, RESTATEMENT triggers file delivery.* ● Set ADJUSTMENT when making partial updates to a previously published batch run. Set the `incrementFrom` field on the BatchRun close event. *Note: For N6 Files, ADJUSTMENT does not trigger file delivery.* **Important:Important**: 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.
incrementFrom	Yes	Yes			Use in combination with mode ADJUSTMENT to specify the previous batchRunId being modified.

2.7 10. 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 data.delivery@spglobal.com

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.
	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.

2.8 11. 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.

3 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.

3.1 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.

3.2 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 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.