transferData HTTP/TLS/JSON

Interface Design Description

Abstract
This document describes an HTTP protocol with TLS payload security and JSON payload encoding variant of the transferData service.

Contents

1 Overview

This document describes the transferData service interface, which provides data transfer capabilities for sensor measurements and JSON-encoded payloads within the Arrowhead Framework.

It's implemented using protocol, encoding as stated in below table:

Profile Type	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	TLS	1.3
Encoding	JSON	RFC 8259 [**?**]
Compression	N/A	-
Semantics	SenML	RFC 8428 [**?**]
Ontology	N/A	-

Table 1: Communication and semantics details used for the transferData service interface

This document provides the Interface Design Description IDD to the *transferData – Service Description* document. For further details about how this service is meant to be used, please consult that document.

The rest of this document describes how to realize the transferData service interface in detail. Both in terms of its operations (Section 2) and its information model (Section 3).

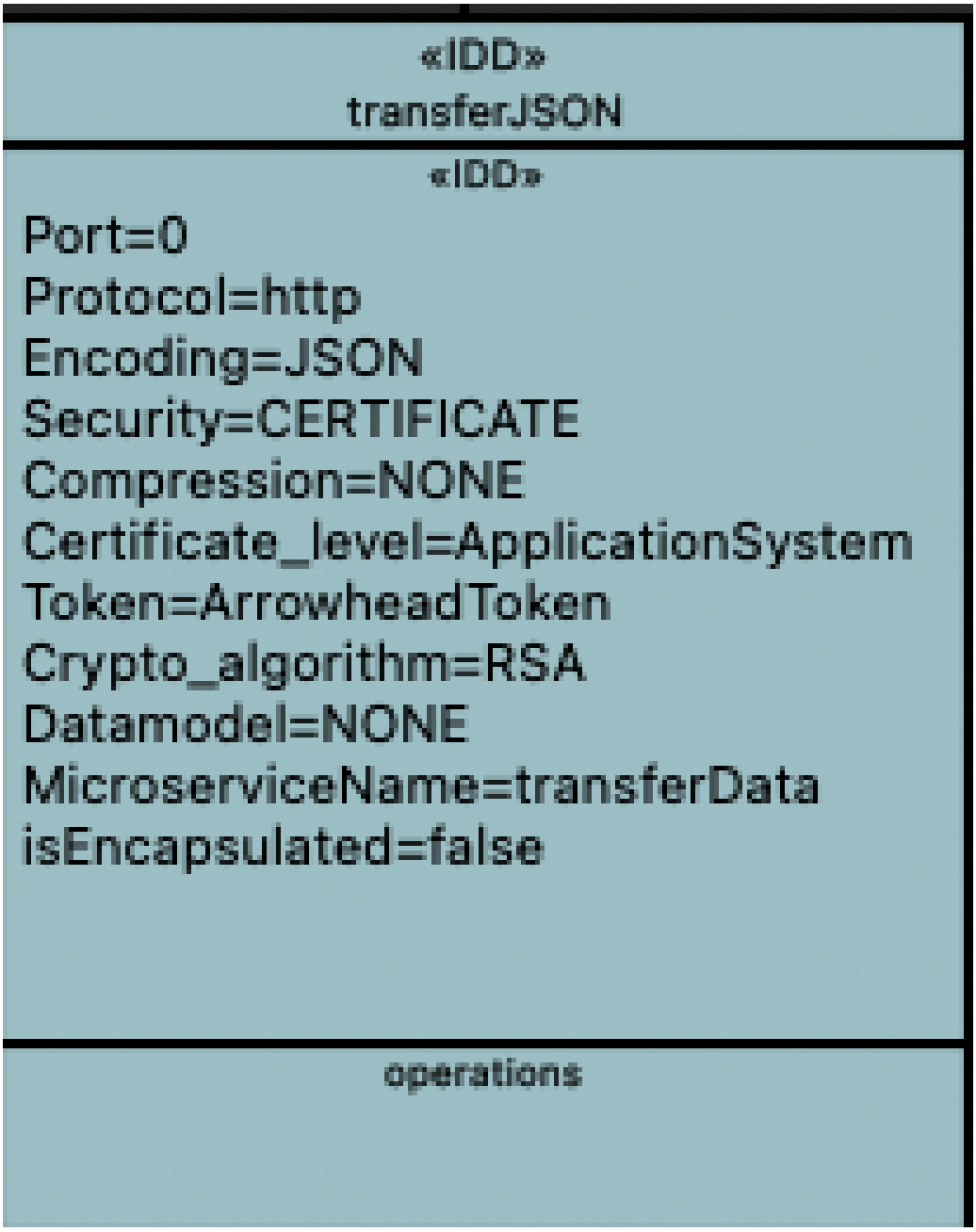


Figure 1: SysML model of the transferData interface, its operations, datamodels and implementation.

2 Service Operations

In particular, each subsection first names the HTTP protocol method and path used to call the function, after which it names an abstract function from the transferData Service Description document, as well as input and output types.

All operations in this section respond with the status code `201 Created` if called successfully. The error codes are, `400 Bad Request` if request is malformed, `401 Unauthorized` if improper client side certificate is provided, `500 Internal Server Error` if the service is unavailable.

2.1 POST /transfer

Operation: Transfer
Input: TransferRequest
Output: TransferResponse

Called to transfer sensor data or JSON payload from a provider system to destination, as exemplified in Listing 1.

```
POST /transfer HTTP/1.1
Content-Type: application/json

{
  "sensorId": "temp-sensor-01",
  "timestamp": "2025-10-20 14:23:45",
  "measurementType": "temperature",
  "value": 23.5,
  "unit": "celsius",
  "metadata": {
    "location": "room-301",
    "accuracy": "+-0.1"
  }
}
```

Listing 1: A Transfer invocation.

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "transferId": 12847,
  "status": "accepted",
  "timestamp": "2025-10-20 14:23:46",
  "sensorId": "temp-sensor-01"
}
```

Listing 2: A Transfer response.

2.2 GET /transfer/{*id*}

Operation: GetTransfer
Output: TransferRecord

Called to retrieve a transfer record by its identifier, as exemplified in Listing 3.

```
GET /transfer/12847 HTTP/1.1
Accept: application/json
```

Listing 3: A GetTransfer invocation.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "transferId": 12847,
  "sensorId": "temp-sensor-01",
  "timestamp": "2025-10-20 14:23:45",
  "measurementType": "temperature",
  "value": 23.5,
  "unit": "celsius",
  "status": "completed",
```

```
  "metadata": {
    "location": "room-301",
    "accuracy": "+-0.1"
  }
}
```

Listing 4: A GetTransfer response.

2.3 DELETE /transfer/{*id*}

Operation: DeleteTransfer
Output: StatusCodeKind

Called to cancel a pending transfer, as exemplified in Listing 5.

```
DELETE /transfer/12847 HTTP/1.1
```

Listing 5: A DeleteTransfer invocation.

```
HTTP/1.1 200 OK
```

Listing 6: A DeleteTransfer response.

2.4 GET /echo

Operation: Echo
Output: StatusCodeKind

Called to check the service availability, as exemplified in Listing 7.

```
GET /echo HTTP/1.1

HTTP/1.1 200 OK
```

Listing 7: An Echo invocation response.

3 Data Models

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.5, which are used to represent things like hashes, identifiers and texts.

3.1 struct TransferRequest

This structure is used to submit sensor data for transfer.

Object Field	Value Type	Description
”sensorId”	Name	Unique identifier of the sensor.
”timestamp”	DateTime	UTC timestamp when measurement was taken.
”measurementType”	Name	Type of measurement (e.g., temperature, humidity).
”value”	Number	Measured value.
”unit”	Name	Unit of measurement (e.g., celsius, percent).
”metadata”	Metadata	Optional additional metadata.

3.2 struct TransferResponse

This structure is returned after successful data transfer submission.

Object Field	Value Type	Description
”transferId”	Id	Unique identifier for this transfer.
”status”	Status	Transfer status.
”timestamp”	DateTime	UTC timestamp when transfer was accepted.
”sensorId”	Name	Echo of the sensor identifier.

3.3 struct TransferRecord

This structure represents a complete transfer record including historical data.

Object Field	Value Type	Description
”transferId”	Id	Unique identifier for this transfer.
”sensorId”	Name	Unique identifier of the sensor.
”timestamp”	DateTime	UTC timestamp when measurement was taken.
”measurementType”	Name	Type of measurement.
”value”	Number	Measured value.
”unit”	Name	Unit of measurement.
”status”	Status	Current transfer status.
”metadata”	Metadata	Optional additional metadata.

3.4 struct Metadata

A JSON Object which maps String key-value pairs.

3.5 Primitives

As all messages are encoded using the JSON format [**?**], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [**?**], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the transferData Service Description document without a direct equivalent among the JSON types. Concretely, we define the transferData primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.5.1 alias DateTime = String

Pinpoints a moment in time in the format of ”YYYY-MM-DD HH:mm:ss”, where ”YYYY” denotes year (4 digits), ”MM” denotes month starting from 01, ”DD” denotes day starting from 01, ”HH” denotes hour in the 24-hour format (00-23), ”MM” denotes minute (00-59), ”SS” denotes second (00-59). ” ” is used as separator between the date and the time. An example of a valid date/time string is ”2025-10-20 14:23:45”

3.5.2 alias Id = Number

An identifier generated for each transfer that enables distinguishing them and later referring to a specific transfer record.

3.5.3 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

3.5.4 alias Status = String

A String that describes the transfer status. Possible values are *accepted*, *pending*, *completed*, *failed*, or *cancelled*.

4 Revision History

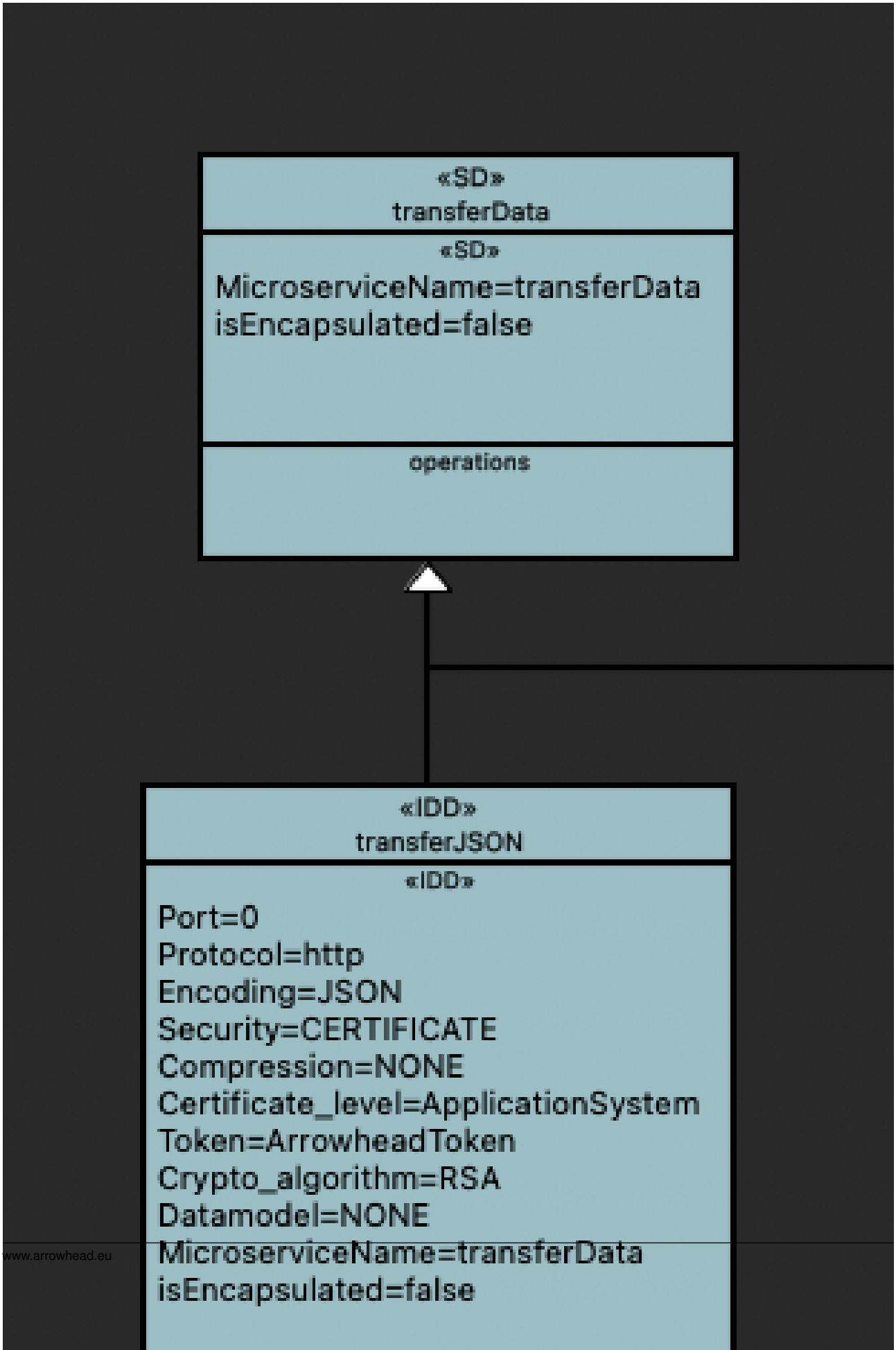
4.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2025-10-22	4.4.1	Initial draft	Your Name

4.2 Quality Assurance

No.	Date	Version	Approved by
1	-	4.4.1	Pending

5 System Architecture Overview


«SD»
transferData
«SD»
MicroserviceName=transferData
isEncapsulated=false
operations
«IDD»
transferJSON
«IDD»
Port=0
Protocol=http
Encoding=JSON
Security=CERTIFICATE
Compression=NONE
Certificate_level=ApplicationSystem
Token=ArrowheadToken
Crypto_algorithm=RSA
Datamodel=NONE
MicroserviceName=transferData
isEncapsulated=false