Atellica® Connectivity Manager

# Data Router Interface Guide

REF 11353454 Rev. 02 2021-09

SIEMENS Healthineers


Siemens Healthcare Diagnostics Inc.
511 Benedict Avenue
Tarrytown, NY 10591 USA

**Siemens Healthineers Headquarters**
Siemens Healthcare GmbH
Henkestraße 127
91052 Erlangen
Germany
Phone: +49 9131 84-0
siemens-healthineers.com

The information in the printed customer documentation was correct at the time of issue. Access the Document Library for current information.

Siemens Healthineers continues to improve products and reserves the right to change specifications, equipment, and maintenance procedures at any time without notice.

THE CUSTOMER DOCUMENTATION INCLUDES INFORMATION ON THE SAFETY HAZARDS ASSOCIATED WITH USE OF THE SYSTEM AND PRECAUTIONS TO BE TAKEN TO AVOID SUCH HAZARDS. FAILURE TO OBSERVE WARNINGS OR USE OF THE SYSTEM IN A MANNER DIFFERENT FROM THAT SPECIFIED BY SIEMENS HEALTHINEERS MAY RESULT IN INJURY TO THE OPERATOR OR OTHER PERSONS. SEE WARNING AND HAZARD STATEMENTS.

# 3 Sample Code

# 1 Overview

The Atellica® Connectivity Manager (Atellica CM) enables secure remote access and proactive monitoring and support. The Atellica CM Data Router facilitates the integration of diagnostic laboratory devices.

This guide describes the Data Router (DR) Application Programming Interface (API) for internal and external use. The DR exposes a set of web-based APIs to obtain laboratory data and DR status information. This API is also available for customer access.

In a diagnostic laboratory, there are many devices that communicate with each other using various protocols. Data exchanged between these devices can be divided into 3 non-exclusive categories:

- Clinical data, such as numerical results and patient demographic data
- Service data, such as instrument logs or sensor readings for troubleshooting
- Operational data, which is information about the management of the laboratory such as inventory or alerts

Laboratory architecture typically consists of closed systems with devices that are tightly connected with each other. Integration of new capability can be a difficult process. Protocols are typically designed for point-to-point systems. Access to the information exchanged between these devices requires software modifications to expose the data to new systems. These protocols require significant domain knowledge to interpret.

The DR provides access to data from diagnostic, automation, and informatics systems in a semi-structured data model to enable realtime monitoring, business intelligence, and data analytics. The DR provides access to data from originating laboratory data producers and publishes that data to any interested subscriber using standard Hypertext Transfer Protocol (HTTP) for batch processing and web sockets for streaming processing.

## Intended Use

The intended use of the Data Router subscriber interface is to facilitate monitoring, business intelligence, and data analytics use cases. Neither the Data Router nor the Atellica CM is classified as a medical device. The operator should not use the subscriber interface in any clinical workflow. Atellica CM is for professional use only.

# 2 Data Router Interface

A device that obtains data from the Data Router is a subscriber. Subscribers use a web API to subscribe to a set of channels and topics to express interest in data. A channel represents a sequence of records that a single device or a group of similar devices produces, while a topic represents a particular type of record within a channel. Topics enable a subscriber to select certain data of interest. For example, to be notified whenever a tube is sealed, a subscriber subscribes to the Sealed topic of the /dms channel. After creating a subscription, the subscriber receives a subscription ID to obtain the subscribed records.

Subscribers can retrieve data using the push or pull web API. The push API uses standard web sockets to deliver data from producer to subscriber and is suitable for near real-time applications. The pull API uses HTTP GET requests to obtain the latest data. Both APIs provide records using JavaScript Object Notation (JSON).

Data subscribers can subscribe to channels where data is published. Data within a channel is further subdivided into topics that can be individually subscribed to, or the subscriber can subscribe to the special “all” topic that transfers all topic data for the specified channel. The subscriber can create a single subscription with multiple channels and multiple topics, or the subscriber can create individual subscriptions. The subscriber uses the subscription ID from the web API to retrieve records. The Data Router delivers records in the order received.

The Data Router exposes a set of API endpoints for different purposes. The following table represents the endpoints and the endpoint general purpose. All endpoints are accessed through encrypted HTTPS.

| Name | Endpoint | Purpose |
|---|---|---|
| JWT | /oauth2/token | Obtains a JSON Web Token for API authentication. |
| Subscription | /api/v1/subscription | Authenticated HTTPS interface to create, update, and delete subscriptions. |
| Pull | /api/v1/data | Authenticated HTTPS interface to pull records for a given subscription ID. |

| Name | Endpoint | Purpose |
|---|---|---|
| Push | /ws/data | Authenticated secure websocket interface to push records for a given subscription ID. |
| Monitoring | /signalr | HTTPS interface to obtain Data Router connector status and other monitoring data. This interface is not authenticated. |

Once a connection either pulls or pushes an endpoint, the Data Router responds with any data currently available for the requested subscription. The provided content is in JSON in accordance with *RFC 7159, JavaScript Object Notation (JSON) Data Interchange Format*. The Data Router retains data for up to 7 days and tracks the current record for the given subscription ID.

## About Response Overview

The following figure is the JSON response from the push and pull endpoints. The subscriber connects to the desired endpoint using the ID of a previously created subscription. After connecting, the Data Router provides the records that correspond to that subscription wrapped in the following JSON object.

```
{
  "lastRecordId": <long>,
  "recordCount": <integer>,
  "nextURL": <string>,
  "records": [
    {
      "channel": <string>,
      "topic": <string>,
      "connector": <string>,
      "record": {
        ...
      }
    }, ...
  ]
}
```

The following table is the Response Schema elements, data types, and descriptions.

| Element | Data Type | Description |
|---|---|---|
| lastRecordId | Long (64 bits) | Record ID of the last record in this sequence. The subscriber should retain this ID in the event of a failure. |
| recordCount | Integer (32 bits) | Number of records contained in this response. |
| nextURL | String | Pull only. The URL of the next response in sequence and used for pagination. This element is only included if there were more records available at the time the response was created. This indicates to the subscriber that the subscriber should immediately request more records. |
| records | Array of Objects | Records that correspond to this subscription. |
| channel | String | Channel producing the current record. |
| topic | String | Topic producing the current record. |
| connector | String | Name of the connector receiving the current record. |
| record | Object | A record. The format depends on the channel and topic. |

## About Record Delivery Sequence

The Data Router retains records in the order in which the records were received and assigns each record an identifier that is monotonically increasing, but not necessarily sequential. Each subscription retains the current position in the record sequence, called nextRecordId. The Data Router tracks the next record ID in the database so that the subscriber's position in the sequence is retained through failures, either in the subscriber or in the Data Router .

**NOTE:** There are some caveats to the record delivery guarantee and are described in *About Record Delivery Guarantee*.

The Data Router only retains data for a subscription that is not yet satisfied, and for those topics with a retention policy of KeepSubscribed**.** The other retention policy is KeepLatest, (see *About Retention Policy*). Each subscription tracks the ID of the last record acknowledged by the subscriber. The Data Router uses this acknowledged record ID to confirm that a subscription is satisfied. When all subscriptions for a record are satisfied, the Data Router purges that record from the database.

There are 2 identifiers to consider: nextRecordId and acknowledgedRecordId. The nextRecordId is a pointer to the next record for a subscription. When a subscriber requests data, the subscriber receives the next record pointed to by the next record ID unless that subscriber overrides the default behavior. The acknowledgedRecordId is a pointer to the last record the subscriber acknowledges. The acknowledgedRecordId pointer tells the Data Router that the subscription is satisfied and the Data Router can purge this record and all prior records, as long as there are no other subscriptions for the record, and the record's retention policy is KeepSubscribed. KeepSubscribed is the default setting.

When creating a subscription, the nextRecordId is set to the ID of the last record received by the Data Router . The subscription's pointer in the record sequence is initialized to *right now*. A subscriber can override the nextRecordId, either when creating the subscription or when requesting data. This enables a subscriber to request older data if the data has not been purged.

## About Acknowledging Records

When records are received, the response includes the identifier of the last record in the payload, called the lastRecordId. Record IDs are monotonically increasing but are not guaranteed to be sequential. Therefore, only the ID of the last record in the payload is known. By acknowledging the lastRecordId, the subscriber acknowledges the identified record and all records before the lastRecordId.

The acknowledgment mechanism is slightly different between the push and pull endpoints.

The pull endpoint includes an optional URL query parameter called acknowledgedRecordId. If this query parameter is not set when accessing this endpoint, then the Data Router automatically acknowledges all records whose record ID is less than the current nextRecordId. In other words, if the Data Router sends a set of records, then the next time the subscriber requests data (for the same subscription) then the previous set of records (and all records prior to this set of records) is considered acknowledged by default.

The subscriber can override the default acknowledgement behavior of the pull endpoint using the nextRecordId and acknowledgedRecordId query parameters. This is useful if the subscriber fails while processing the previous set of records. To receive those records again, the subscriber must set the nextRecordId so that the Data Router resends the records. If the subscriber has previously acknowledged the records, then there is no guarantee the records will be available.

If the subscriber requires more control over the acknowledgments, then the subscriber can set the acknowledgedRecordId query parameter. This is useful, for example, if the records are processed by the subscriber in a separate thread of execution and the subscriber only wants to acknowledge those records after the records have been fully processed by the system. The subscriber can set the acknowledgedRecordId to the last record that the subscriber processed, ensuring the Data Router retains that data until the data is completely processed by the subscriber.

The push endpoint behaves similarly to the pull endpoint, but the mechanism for communicating an acknowledged record ID is different. The push endpoint includes a nextRecordId query parameter that behaves identically to the pull endpoint. However, the push endpoint does not include an acknowledgedRecordId query parameter. When using the push endpoint, the subscriber must send a JSON packet back to the Data Router through the websocket connection. This JSON packet includes the acknowledgedRecordId and indicates to the Data Router that the identified record, and all records prior to it, are acknowledged. This JSON packet can be sent at any time, for example, after the system processes the received records.

If a subscriber never acknowledges records, then the Data Router retains those records with the KeepLatest retention policy for a maximum of 7 days.

If a subscriber sets the nextRecordId to an ID that is older than any record in the Data Router, then the Data Router sends the next record in the sequence that satisfies the subscription. If a subscriber sets the nextRecordId to an ID that is greater than the ID of any record in the Data Router, the subscriber does not receive any records for that subscription until the Data Router catches up. Record IDs are 64-bit signed integers. The minimum value for a record ID is 0, and the maximum value for a record ID is $2^{63}$-1.

## About Record Delivery Guarantee

The Data Router guarantees that each record it successfully receives is delivered at least once. In the following scenario, the Data Router sends a payload of records to the subscriber and then fails before the Data Router can store the nextRecordId in the database. When the Data Router restarts, the Data Router initializes the subscription with the old value for the nextRecordId and resends the previous data.

The Data Router cannot guarantee every record sent to the Data Router by a producer is delivered to a subscriber. Many of the producers sending messages to the Data Router do not support an acknowledgment mechanism. These producers assume the Data Router has successfully received the message. The producers delete the message from their storage as soon as the message is sent to the Data Router. The Data Router buffers received messages up to 250 ms before committing the records to the DB. If the Data Router fails during this time, those messages are lost.

A subscriber can help the system guarantee *exactly-once record delivery* semantics. This method requires cooperation between the Data Router and the subscriber. For example, the Data Router has successfully sent a set of records to the subscriber but the Data Router failed before storing the nextRecordId in the database. The connection to the subscriber is lost because of the failure. When the subscriber reconnects, the subscriber can set the nextRecordId query parameter, telling the Data Router where it should begin in the record sequence. With proper coordination between the Data Router and the subscriber, the Data Router does not resend data under a failure. However, the subscriber must now retain the nextRecordId. If the subscriber processed all the received records and then failed before storing the nextRecordId in its database, the subscriber receives duplicate records. To ensure *exactly-once record delivery semantics,* the subscriber must store all received records and the nextRecordId in a single database transaction. The subscriber must provide the nextRecordId to the Data Router when connecting to ensure *exactly-once* record delivery semantics.

# About Retention Policy

Every topic is assigned 1 of the following 2 retention policies:

- KeepSubscribed
- KeepLatest(N)

Most topics are assigned the KeepSubscribed retention policy. This policy represents streaming data that can be purged once all the subscriptions are satisfied.

The KeepLatest(N) retention policy states that the Data Router should keep the most recent N records for this topic. This retention policy is for records that are only received once, usually when the Data Router first connects to the producer of that data. This kind of data typically represents configuration data and does not typically fit into the streaming model. The KeepLatest(N) retention policy keeps these records from being purged and keeps the records available to the Data Router.

The Data Router does not purge records with the KeepLatest(N) retention policy after 7 days, and does not depend on any subscription to be retained. This means that the Data Router always keeps the most recent N records for each topic with the KeepLatest(N) retention policy.

The topics with the KeepLatest(N) retention policy are in the following table. If a topic is not listed in this table, then the retention policy is KeepSubscribed.

| Channel | Topic | N (number of records) |
|---|---|---|
| /dms | Layout | 15 |
| /dms | TestList | 150 |
| /dms | Ward | 50 |
| /dms/typedRaw | Layout | 15 |
| /dms/typedRaw | TestList | 150 |
| /dms/typedRaw | Ward | 50 |
| /i2i | DeviceListUpdate | 10 |
| /aptio/apm | ConfigLayout | 1 |
| /aptio/apm | ConfigNodeType | 1 |
| /aptio/apm | ConfigErrorHandling | 1 |

| Channel | Topic | N (number of records) |
|---|---|---|
| /aptio/apm | ConfigLIS | 1 |
| /aptio/apm | ConfigErrorHandlingOverride | 1 |

## About Channels and Topics

The following table summarizes the channels and topics available for subscription. In addition to the listed topics, each channel supports the special "all" topic that provides records for all topics for the associated channel.

| Source / Channel | Description | Topics |
|---|---|---|
| **Source: Data Management Software** | | |
| /dms | Patient sample events generated by the Aptio Automation system. Many sample routing events are generated including check-in, reload, entering and exiting storage, samples, centrifuged, sealed and unsealed, decapped and recapped, and more. Sample workflow events are also generated including ordered, resulted, queried, and QC results. Sample weight and volume events are provided. Also includes DMS configuration data and events including test mappings, module layout information, and ward configuration. | Layout, TestList, Ward, Order, Result, QcResult, CheckedIn, Reloaded, ReIdentified, ExitedStorage, AliquoterCheckedIn, Routed, Sampled, Queried, Stored, Removed, Centrifuged, Sealed, Unsealed, Decapped, Recapped, Disposed, RackDisposed, RackRemoved, SampleWeight, PriorityIncrease, SampleVolume, Warning, AliquotedInformation, InventoryInformation, Aliquoted, VerticalTransportCheckin, PreAdmissionOrder, DirectEmerge, ColorEmerge, Shaken, Transferred, SortInfo, TestStatusChange, SampleStatusUpdate, EventAcknowledgement, Request, Enable, Disable, PlateInfo |
| **Source: Raw Data Management Software** | | |
| /dms/typedRaw | Identical to the /dms channel, but records are sent in raw ASTM format. | Same Topics as /dms channel |

| Source / Channel | Description | Topics |
|---|---|---|
| **Source: Atellica Data Manager** | | |
| /adm | Test result data. When a laboratory includes both a DMS and an Atellica DM system, this is the preferred channel to use as the Atellica DM system has a more accurate depiction of the result. | Order, Result, QcResult, Queried, CheckedIn, Removed, Centrifuged, Sealed, Unsealed, Recapped, Decapped, Disposed, AliquotedInformation, PriorityIncrease, Stored, Warning, ExitedStorage |
| **Source: Raw Atellica Data Manager** | | |
| /adm/typedRaw | Identical to the /adm channel, but records are sent in raw ASTM format. | Same Topics as /adm channel |
| **Source: CentraLink System** | | |
| /centraLink | Deprecated. Results from the Centralink system. | OUL |
| **Source: Aptio Sample Management System** | | |
| /aptio/apm | Node status, node error events, and node configuration generated by the Aptio Automation system, specific to the requirements of Atellica PM. | ControlNodeError, ControlRunStatus, ConfigLayout, ConfigNodeType, ConfiguErrorHandling, ConfigErrorHandlingOverride, ConfigLIS |
| **Source: Instrument to Informatics** | | |
| /i2i | Events generated by Siemens analyzers and instruments, including device connect and disconnect, calibrations, status updates, inventory status, and generic event and status notifications. Also includes non-clinical information about test results. | EventNotification, ClearedEventNotification, CalibrationUpdate, PropertyUpdate, InventoryStatusUpdate, DataSetUpdate, StatusUpdate, RtsTestResultUpdate, DeviceConnect, DeviceDisconnect, InventoryReportUpdate, ServerConnect, ServerDisconnect, DeviceListUpdate, DeviceConfigurationAdd_Data, DeviceConfigurationUpdate_Data, DeviceConfigurationRemove_Data |

# Setting Up Access to the Data Router Web API

Before a client can access the Data Router Web API, the following configuration must be complete:

- Checking the Status and Enabling the Data Router
- Configuring the database
- Installing the certificate
- Creating Connectors
- Creating a subscriber with the Data Router web configuration tool

## Checking Status and Enabling the Data Router

During system installation, the Data Router was disabled by default.

### Checking the Data Router Status

1. On the desktop, open the Services application.

   Search for Services by entering **Services** in the Microsoft Windows search.

2. On the Services screen, scroll to the Data Router service.

   If the Status column displays Running and the Startup Type column displays Automatic, the Data Router is enabled. Continue to *Configuring the Database*.

3. If the Status is Disabled, continue to *Enabling the Data Router*.

### Enabling the Data Router

1. On the Services application, right-select the Data Router service and select **Properties**.
2. In the Startup type drop-down list, select **Automatic**.
3. Select **Apply**.
4. In Service status area, select **Start**.
5. Select **OK**.

## Configuring the Database

The Data Router database may store patient's Protected Health Information (PHI) and Personally Identifiable Information (PII) that has been transferred through the system. The database is encrypted with a unique encryption key to protect the information. In accordance with strict privacy regulations, this information should only be accessible by personnel who have been provided permission by the patient. Therefore, the unique database encryption key should only be known by a laboratory administrator and used by the Data Router. Occasionally a Siemens service technician may require the database encryption key to troubleshoot an issue. If the Siemens service technician requires the key, the service technician requests encryption key from the laboratory administrator. All Siemens Healthineers representatives are committed to maintaining patient data confidentiality and are trained to comply with regional procedures for handling PHI, PII, and Electronic Protected Health Information (ePHI).

1. On the computer where the Data Router is installed, open the Data Router Configuration utility.

   Start the utility by one of the following methods:

   - Using the Atellica Connectivity Manager Group
     i Select **Start**.
     ii Select the Atellica Connectivity Manager group.
     iii Select **Data Router Configuration**.
   - Using Google Chrome
     i Start Google Chrome.
     ii Browse to https://127.0.0.1:8002.

2. Log into the Data Router Configuration utility using credentials that have local administrator privileges.

   The configuration utility supports local and Active Directory users that have local administrator privileges. (Siemens Service can use the Level2 credentials.)

3. Select a folder location for the database.

   The default path is %PROGRAMDATA%\Siemens\Data Router\Data.

4. Select **Configure**.

   A screen displays with an automatically generated database encryption key.

5. Copy this key and store the information in a secure location.

   **NOTE:** The encryption key protects patient PHI/PII. Ensure the encryption key is stored in a secure location that is protected by a laboratory administrator. Once selecting Confirm, this key cannot be retrieved. In the future, this key may be necessary for troubleshooting.

6. Select **Confirm**.

Continue with *Running the Data Router Bootstrapper*.

### Running the Data Router Bootstrapper

The bootstrapper requires temporary local administrator rights as part of the Data Router configuration. The bootstrapper presents a User Account Control (UAC) prompt, if necessary, to elevate the privileges to perform the administrative tasks necessary to configure the database.

1. Select **Start**.
2. Select the Atellica Connectivity Manager group.
3. Select **Data Router Bootstrapper**.
4. If a UAC prompt displays, select **Yes** to temporarily elevate the bootstrapper permissions to local administrator.

   Wait for the bootstrapper to complete.

5. Refresh the Data Router Configuration utility.
6. Log in to the Data Router Configuration utility to confirm that the database configuration is complete.

## Installing the Certificate

The Data Router requires a properly configured x509 certificate. This certificate contains a private key to digitally sign JSON Web Tokens (JWT) and verify the digital signature of JWTs. The JWTs represent claims of the authenticity of a subscriber and are required to access the Data Router Web API.

The certificate also confirms the host is trusted by the client that is accessing the Data Router and helps eliminate man-in-the-middle attacks. The Data Router uses this certificate to identify the Data Router server when a client connects using HTTPS. Each client that accesses the Data Router must trust this certificate or bypass the security check. When the client machine receives this certificate during the HTTPS handshake, the client machine verifies that the host name (IP address, computer name, or domain name) used to access the Data Router matches one of the Subject Alternate Names in the trusted certificate. The client machine verifies the digital signature of the certificate.

When the Data Router was installed, the Data Router generated a self-signed digital certificate. By default, the Data Routere server uses the self-signed certificate. The laboratory has the option to use a certificate provided by laboratory IT instead. This can be helpful if IT security policies do not allow self-signed certificates, or if the laboratory IT already provides their own certificate trust chain.

If you using the self-signed certificate, perform the steps in *Trusting the Self-Signed Certificate*. If the laboratory-IT has an existing certificate infrastructure, perform the steps in *Configuring a Laboratory IT Provided Certificate*.

## Trusting the Self-Signed Certificate

Every client that connects to the Data Router, either through the web API or the web-based configuration utility, must trust the certificate used to host the Data Router. When the Data Router was installed, the Data Router generated a self-signed digital certificate. To verify the identity of the Data Router while connecting, each client must install the Data Router root Certificate Authority client certificate (CA Certificate). The CA Certificate is exported in the following location on the Data Router computer:

%PROGRAMDATA%\Siemens\Data Router

The name of the CA Certificate file is the computer name of the Data Router, followed by _CA.cer. Transfer this file to each computer that will connect to the Data Router. Once the certificate is copied to the computer, the CA Certificate must be trusted, either by the application accessing the Data Router or the computer.

This document provides overview instructions for the most common method to trust a certificate. This method is to install the certificate into the Local Machine store in the Microsoft Management Console (MMC). Consult your IT security to ensure the following procedure meets your organization's security policies.

**NOTE:** If the laboratory-IT has an existing certificate infrastructure, perform the steps in *Configuring a Laboratory IT Provided Certificate*.

1. Copy the CA Certificate file from %PROGRAMDATA%\Siemens\Data Router\<ComputerName>_CA.cer to the client PC.
2. On the client PC, run certlm.msc.

   The MMC with the Certificate plugin targeted to the local machine displays.
3. Right select **Trusted Root Certification Authorities**.
4. Select **All Tasks -> Import**.
5. Select **Next**.

6. Browse to the CA Certificate.
7. Select **Next > Next > Finish**.

   An Import was successful message displays.

Verify the Certificate Import using the steps in *Verifying the Certificate Import, page 23*.

## Configuring a Laboratory IT Provided Certificate

The Data Router can be configured to use an existing certificate infrastructure provided by the laboratory IT. In this scenario a laboratory IT provided certificate chain of trust is already installed on each customer-supplied computer in the laboratory network.

Use the following procedure to configure the Data Router to use a laboratory IT-supplied server certificate.

1. On the computer where the Data Router is installed, open the Data Router Configuration utility.

   Start the utility by one of the following methods:

   - Using the Atellica Connectivity Manager Group
     i Select **Start**.
     ii Select the Atellica Connectivity Manager group.
     iii Select **Data Router Configuration**.
   - Using Google Chrome
     i Start Google Chrome.
     ii Browse to https://127.0.0.1:8002.
2. Log into the Data Router Configuration utility using credentials that have local administrator privileges.

   The configuration utility supports local and Active Directory users that have local administrator privileges. (Siemens Service can use the Level2 credentials.)
3. On the Administration tab, select Certificate.
4. On the computer where the Data Router is installed, run **certlm.msc**.

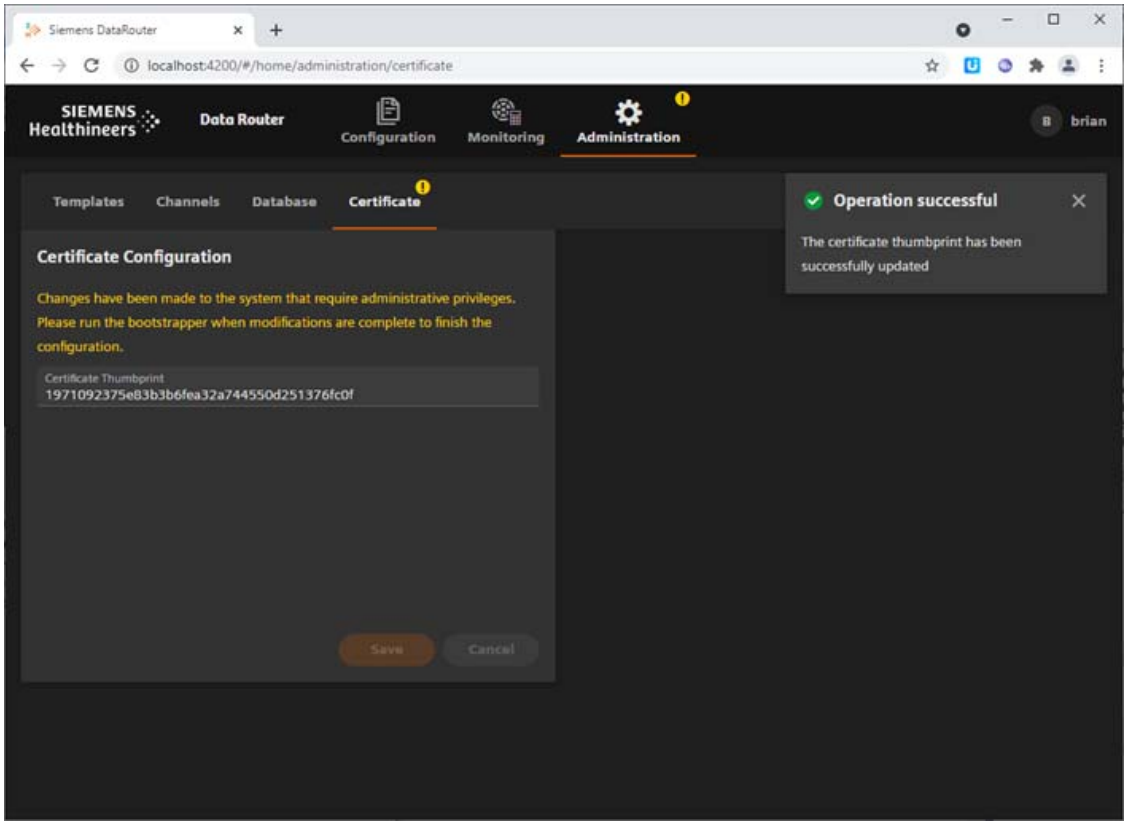
   The MMC with the Certificate plugin targeted to the local machine displays.
5. Locate the customer-provided server certificate that will be used to host the Data Router.

   This certificate must contain a private key and must be in the Personal certificate store of the computer hosting the Data Router.
6. Double-select the certificate.
7. Select the Details tab.

8. Scroll to the Thumbprint property.
9. Copy and paste the certificate thumbprint into the Data Router certificate configuration.
10. Select **Save**.

    If a warning message displays, review the warning message description for further resolution, if needed.
11. Select **Confirm**.

    An Operation successful message displays.




12. Run the Data Router Bootstrapper.

    Refer to *Running the Data Router Bootstrapper, page 19*.
13. Verify the certificate is properly configured.

    Refer to the *Verifying the Certificate Import* procedure.

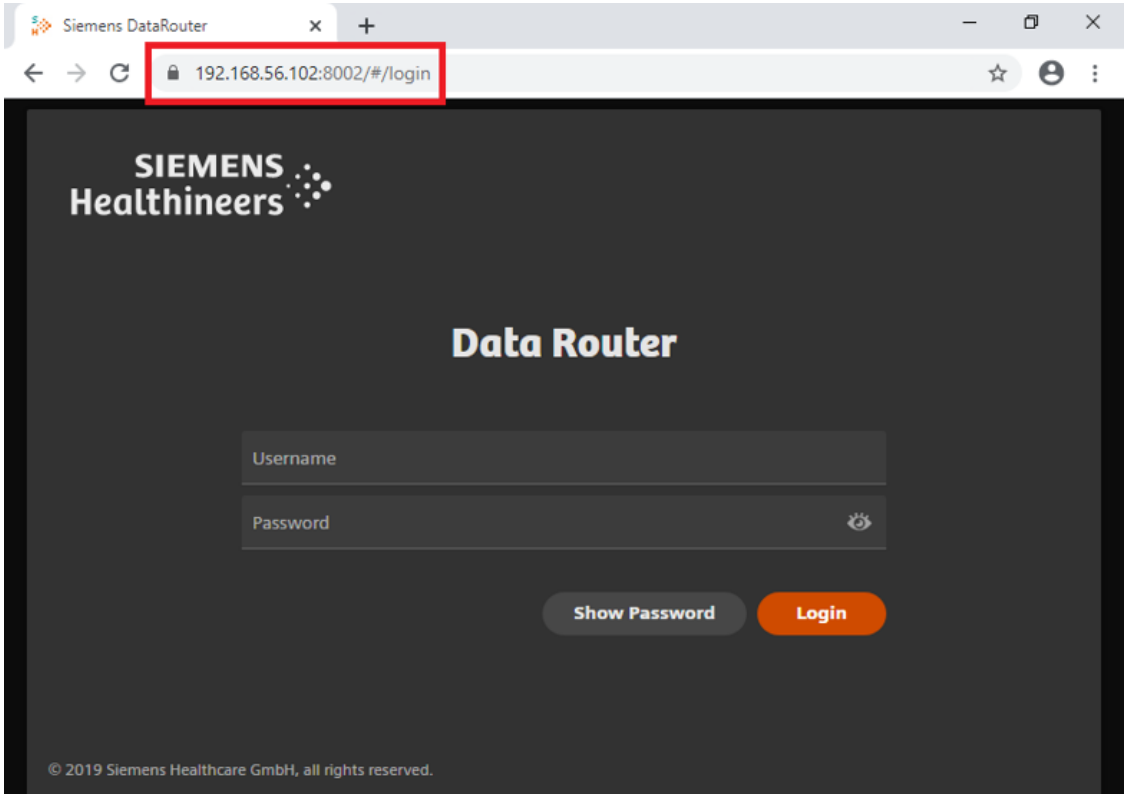
### Verifying the Certificate Import

1. Using Google Chrome, browse to the Data Router web configuration utility.

   The URL is https://<host>:<port>, where host is the IP address, computer name, or domain name of the Data Router. The port is 8002 by default. For example, https://192.168.56.102:8002.

2. If the client certificate is properly imported, the Data Router login displays.

   The following figure shows the login screen. The padlock by the address indicates a trusted connection.




If the certificate was not imported correctly, a privacy error message displays. This indicates that the connection is not private and not secure. If this message displays, **contact the Siemens Remote Services Center**.

## Configuring Connectors

Connectors are how the Data Router obtains data from producers. Each instance of a producer has a corresponding connector to receive data. The connector understands the protocol of the producer and translates received messages into records. *Table 1* lists the supported connector types and corresponding channels for which the connector produces records.

**Table 1: Connector Types**

| Type | Producer | Channels |
|---|---|---|
| DMS | Aptio Data Management System | /dms<br>/dms/typedRaw |
| i2i | Atellica Connectivity Manager Instrument to Informatics Service | /i2i |
| Aptio system | Aptio Sample Management System File Share | /aptio/apm |
| CentraLink System | CentraLink | /centralink |
| Atellica Data Manager | Atellica Data Manager | /adm<br>/adm/typedRaw |
| AptioAgent | Aptio Sample Management System DAS Agent | /aptio/apm |

The set of connectors required for the Data Router depends on the application and the laboratory configuration. Contact your local technical support representative with questions about creating and configuring the connectors.

Ensure the certificate is installed before configuring the connector. Refer to *Installing the Certificate, page 19*.

1. Using Google Chrome, browse to the Data Router web configuration utility.

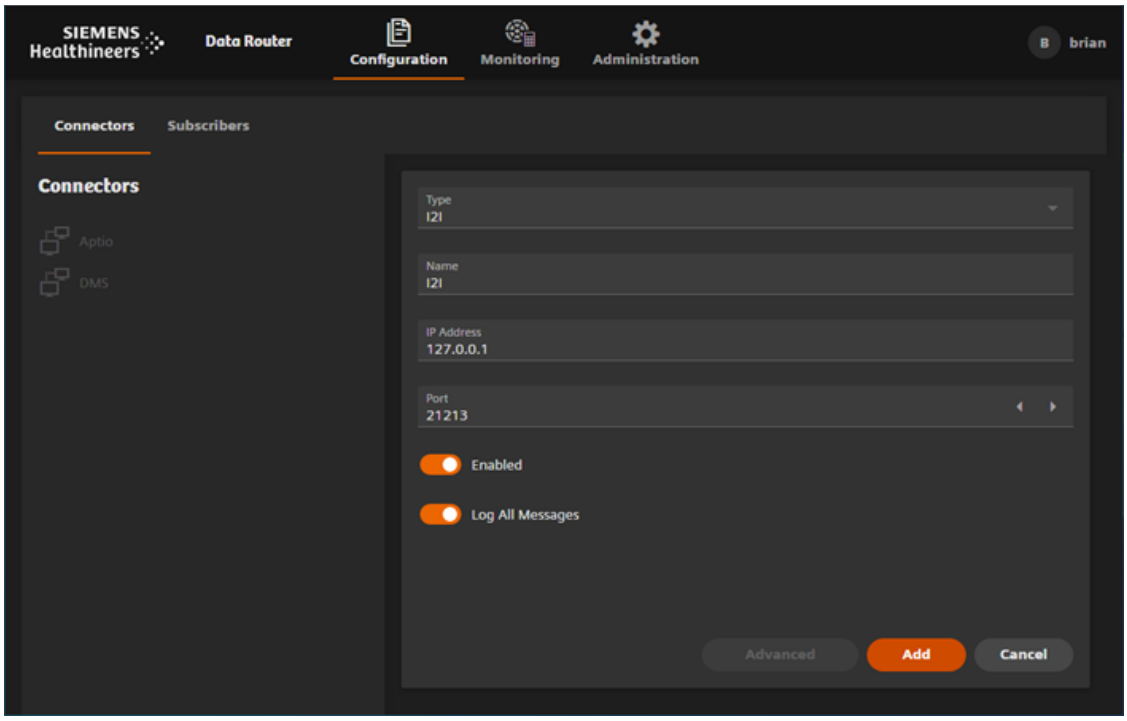
   The URL is https://<host>:<port>, where host is the IP address, computer name, or domain name of the Data Router. The port is 8002 by default. For example, https://192.168.56.102:8002.

2. Log into the Data Router Configuration utility using credentials that have local administrator privileges.

   The configuration utility supports local and Active Directory users that have local administrator privileges. (Siemens Service can use the Level2 credentials.)

3. On the Connectors page, select **+** (plus sign) to create a new connector.
4. On the Connectors page, select a Type from the drop-down list.

   If you are unsure about which type to select, contact your local technical support representative.




5. A connector name is automatically created in the Name field.

   The name can be changed to meet the laboratory naming conventions.

6. Enter the IP address.

   a. The AptioAgent connector type is a server. Enter the IP address of a local interface card, or leave the IP address at the default address "0.0.0.0" to listen for incoming connections on all interfaces.

   b. All other connector types are clients. Enter the IP address of the device to which the systems is connecting.

7. The remaining fields and settings can be left at default.

   Contact your technical support representative before modifying any other settings.

8. Select **Add** to create the connector.

## Creating a Subscriber

A subscriber represents an entity, usually a computer or an application, that obtains records from the Data Router. A subscriber is identified by a name and an API key and is created using a configuration utility hosted by the Data Router web server. When a subscriber is created, an API key is generated by the Data Router. This key is required to access the subscription API.

Configure subscribers by accessing the web-based configuration utility using Google Chrome. An administrator account that has access to the computer where the Data Router is installed is required. This administrator account is a local or domain account that is a member of the local machine administrator's group.

1. Using Google Chrome, browse to the Data Router web configuration utility.

   The URL is https://<host>:<port>, where host is the IP address, computer name, or domain name of the Data Router. The port is 8002 by default. For example, https://192.168.56.102:8002.

   **NOTE:** Only Google Chrome is supported.

2. Log in to the Data Router web configuration utility using an administrator account that has access to the computer where the Data Router is installed.

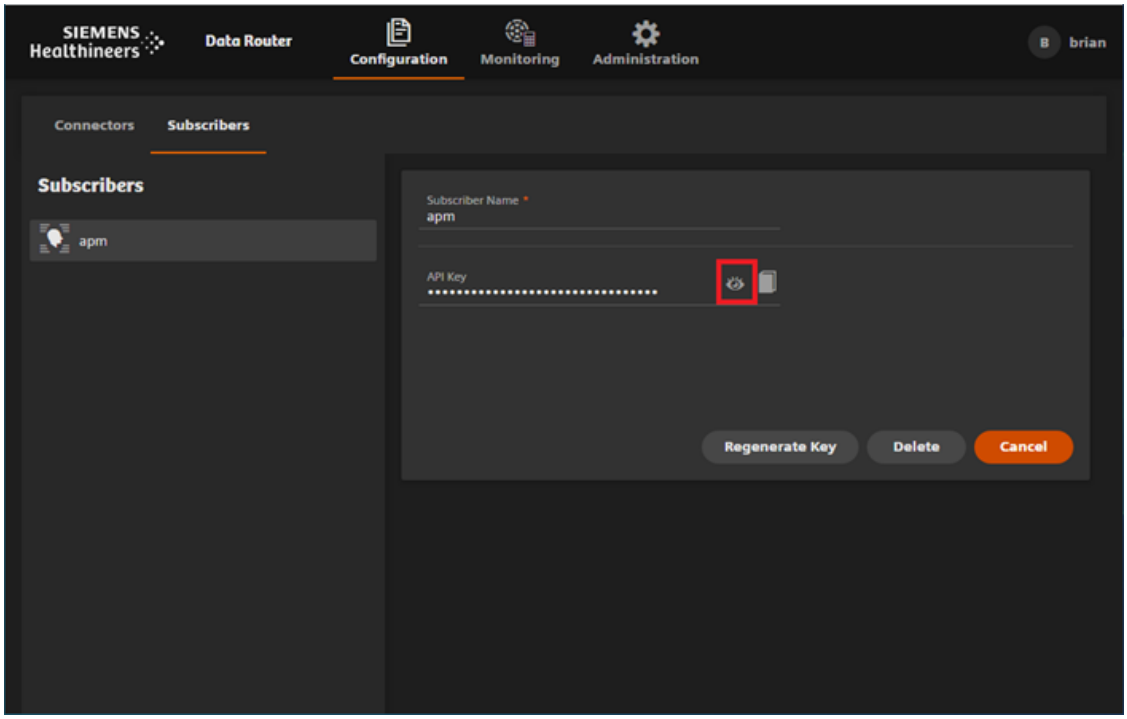
   If the administrator is logging in with a domain account, prefix the user name with the domain name and a backslash. For example my_domain\administrator. If the backslash is not present, the Data Router assumes the login is for a local machine user.

3. Select **Configuration**.
4. Select **Subscribers**.
5. Select the **+** (plus sign) to add a subscriber.
6. In the Subscriber Name area, enter a name for the new subscriber.
7. Select **Save**.

View or copy the auto-generated API key to provide to the application that accesses the Data Router.

## Viewing the API Key

1. Select **Configuration > Subscribers**.
2. Select the Subscriber.
3. Select the Show key icon in the API Key area to view the API Key.




## Regenerating the API Key

The API key cannot be modified and never expires, but the API key can be regenerated. Regenerating the API key causes any currently connected subscriber to disconnect. The API key can be regenerated periodically to satisfy local IT security requirements, or if the existing API key has been compromised.

1. Select **Configuration > Subscribers**.
2. Select the Subscriber.
3. Select **Regenerate Key**.

## Monitoring the Connectors

The Monitoring screen provides a real-time indication of the connectivity status for each of the configured connectors.

1. Using Google Chrome, browse to the Data Router web configuration utility.

   The URL is https://<host>:<port>, where host is the IP address, computer name, or domain name of the Data Router. The port is 8002 by default. For example, https://192.168.56.102:8002.

   **NOTE:** Only Google Chrome is supported.

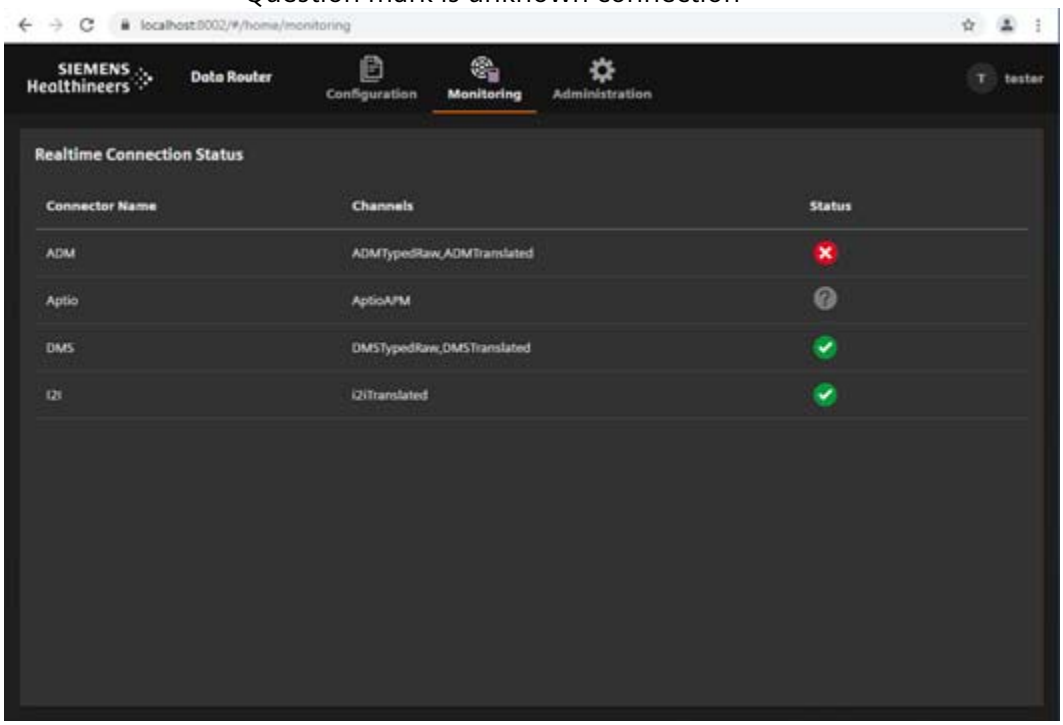
2. Log in to the Data Router web configuration utility using an administrator account that has access to the computer where the Data Router is installed.

   If the administrator is logging in with a domain account, prefix the user name with the domain name and a backslash. For example my_domain\administrator. If the backslash is not present, the Data Router assumes the login is for a local machine user.

3. Select the Monitoring page.

   The connector status displays:

   - Green is an active connection
   - Red is no connection
   - Question mark is unknown connection



# Interface Specification

In this section the details of the Data Router Web API are presented. The URL for each request is constructed according to the following layout:

| **scheme://host:port/endpoint** | |
|---|---|
| scheme | Either https or wss |
| host | The IP address, computer name, or domain name of the Data Router |
| port | The Data Router's port number, usually 8002 |
| endpoint | The specific API endpoint to be addressed. See the following sections for the list of API endpoints representing the Data Router's capability. |

# About the JSON Web Token

The JWT endpoint authenticates with the Data Router to obtain a JWT, which grants access to the Subscription and Pull Data Router API endpoints. A JWT is not required to access the Push websocket endpoint.

A JWT is a digitally signed JSON object that contains a set of claims. Once a subscriber provides the login credentials and obtains a JWT, the subscriber is considered logged in until the token expires. (The JWT encapsulates the authentication and is stored in the client. The Data Router verifies the digital signature of the JWT and, if valid, trusts the claims.)

The subscriber provides the JWT with each subsequent request. The Data Router only verifies the signature, ensuring the token is valid.

To obtain a JWT, the subscriber must form an HTTPS request as follows:

| **Description** | **Get a JWT for API access** |
|---|---|
| URL | https://<host>:<port>/oauth2/token |
| Method | Post |
| **Headers** | |
| Content-Type | application/x-www-form-urlencoded |
| **Body** | |
| grant_type | client_credentials |
| client_id | <username> |
| client_secret | <API key> |

The username and API key must be obtained from the web-based Data Router configuration utility user management screen. A properly formatted and valid request results in a 200 OK response as follows:

```
{
  "access_token": <access token>,
  "token_type": "bearer",
  "expires_in": 1799
}
```

The given access token authenticates the Subscription or Pull endpoints. When making an HTTPS request to one of these endpoints, the authorization header must be provided as follows, where the <access token> is a JWT received from the JSON Web Token endpoint. See RFC 6750 for more details.

| Key | Value |
|---|---|
| Authorization | Bearer <access token> |

The JWT can be reused for each access to the API. The JWT can be used for up to 1799 seconds, or just under 30 minutes, at which point the JWT expires and a new JWT must be obtained.

A 401 Unauthorized response to an API request can indicate the JWT has expired. If this response occurs, obtain a new JWT and retry the previous request.

Alternatively a subscriber can request a new JWT each time the subscriber accesses the Subscription or Pull API endpoints. The JWT mechanism is stateless, meaning the server does not retain any information when a JWT is created.

## Subscription

The subscription API manages subscriptions for the authenticated user as claimed by the provided JWT. Each method in this endpoint requires bearer authorization with a valid JWT. The following methods are supported:

- Get All Subscriptions
- Get Specific Subscriptions
- Add Subscription
- Update Subscription
- Delete Subscription
- Delete All Subscriptions

# Get All Subscriptions

| | |
|---|---|
| **Description** | **Retrieves all (Get all) subscriptions for the authenticated user.** |
| URL | https://<host>:<port>/api/v1/subscription/all |
| Method | GET |
| **Headers** | |
| Authorization | Bearer <access token> |

## Response Example


```
[
    {
        "id": "5b9012bc8e4440c5b2de065eed115690",
        "subscribables": [
            {
                "channel": "/dms",
                "topics": [
                    "TestList"
                    "Ward",
                    "Layout"
                ]
            },
            {
                "channel": "/i2i",
                "topics": [
                    "all"
                ]
            }
        ]
    },
    {
        "id": "7c05709ee7194bc082fd84ca42c58229",
        "subscribables": [
            {
                "channel": "/adm",
                "topics": [
                    "Result"
                ]
            }
        ]
    }
]
```

## Get Subscription

| | |
|---|---|
| **Description** | **Retrieves (Get) a specific subscription for the authenticated user.** |
| URL | https://<ip address>:<port>/api/v1/subscription/{id} |
| Method | GET |
| **Headers** | |
| Authorization | Bearer <access token> |

### Response Example


```
{
     "id": "5b9012bc8e4440c5b2de065eed115690",
     "subscribables": [
          {
               "channel": "/dms",
               "topics": [
                    "all"
               ]
          }
     ]
}
```


## Add Subscription

| | |
|---|---|
| **Description** | **Adds a new subscription for the authenticated user.** |
| URL | https://<host>:<port>/api/v1/subscription?nextRecordId=<id> |
| Method | POST |
| **Headers** | |
| Authorization | Bearer <access token> |
| **Optional Query Parameters** | |
| nextRecordId | Sets the next record ID. If this parameter is not included, then the next record ID is set to the current record. If the subscriber would like to receive older data, then the ID can be set to a previous value. Under normal operation only the KeepLatest(N) records are retained in the absence of an unsatisfied subscription. |

### Body Example

```
[
  {
      "channel": /dms",
      "topics": [
          "TestList",
          "Ward",
          "Layout"
      ]
    },
    {
      "channel": "/i2i",
      "topics": [
          "all"
      ]
    }
]
```

### Response Example

```
{
    "id": "5b9012bc8e4440c5b2de065eed115690",
    "subscribables": [
        {
            "channel": /dms",
            "topics": [
                "TestList",
                "Ward",
                "Layout"
            ]
        },
        {
            "channel": "/i2i",
            "topics": [
                "all"
            ]
        }
    ]
}
```

If the next record ID query parameter is provided, then the Data Router begins delivering records for this subscription beginning with the provided ID. If the provided ID doesn't exist, then the sequence begins at the next record whose ID is greater than the provided ID, and that record belongs to one of the requested subscribables.

A subscriber either receives records right now (default) or from the beginning record. If a subscriber wants to receive records from the beginning, which are records up to 7 days old, then the subscriber sets the next record ID to zero (0). Most records are purged from the system. Only the KeepLatest(N) records are retained if there is no subscription.

## Update Subscription

When a subscription is updated, any active websocket connections are closed.

| | |
|---|---|
| **Description** | **Updates an existing subscription for the authenticated user.** |
| URL | https://<host>:<port>/api/v1/subscription/{id}?nextRecordId=<id> |
| Method | PUT |
| **Headers** | |
| Authorization | Bearer <access token> |
| **Optional Query Parameters** | |
| nextRecordId | Sets the next record ID. If this parameter is not included, then no change is made to the next record ID. If the subscriber would like to receive older data, then the ID can be set to a previous value. Only KeepLatest(N) records are guaranteed to be retained in the absence of an unsatisfied subscription. |

### Body Example

```
{
        "channel": /dms",
        "topics": [
            "TestList"
            "Ward"
            "Layout"
        ]
    },
    {
      "channel": "/i2i",
      "topics": [
            "all"
        ]
    }
]
```

### Response Example


```
{
     "id": "5b9012bc8e4440c5b2de065eed115690",
     "subscribables": [
          {
               "channel": /dms",
               "topics": [
                    "TestList"
                    "Ward"
                    "Layout"
                ]
          },
          {
               "channel": "/i2i",
               "topics": [
                    "all"
               ]
          }
     ]
}
```


## Delete Subscription

Use Delete Subscription to delete a single subscription. When a subscription is deleted, any active websocket connections will be closed.

| Description | Deletes an existing subscription for the authenticated user. |
|---|---|
| URL | https://<host>:<port>/api/v1/subscription/{id} |
| Method | DELETE |
| **Headers** | |
| Authorization | Bearer <access token> |

### Response Example


```
"Subscription ID - 5b9012bc8e4440c5b2de065eed115690
deleted successfully."
```

### Delete All Subscriptions

Delete Subscriptions deletes all subscriptions and closes all related websocket connections.

| | |
|---|---|
| **Description** | **Deletes all subscription for the authenticated user.** |
| URL | https://<host>:<port>/api/v1/subscription/all |
| Method | DELETE |
| **Headers** | |
| Authorization | Bearer <access token> |

**Response Example**

```
"All subscriptions deleted successfully."
```

## Using Pull (HTTP GET) Method

The pull method for retrieving data from the Data Router enables the client to receive data in the HTTP response. The client periodically requests data using the pull interface to retrieve up-to-date information. The format for the HTTP GET request is as follows:

| | |
|---|---|
| **Description** | **Gets data for the given subscription ID.** |
| URL | https://<address>:<port>/api/v1/data/{subscriptionId}?nextRecordId=<id>&limit=<limit> |
| Method | GET |
| **Headers** | |
| Authorization | Bearer <access token> |
| **Optional Query Parameters** | |
| nextRecordId | Sets the Data Router position in the record stream. If this parameter is not provided, then the Data Router continues where the Data Router last left off. The first record in the response has an ID that is equal to or greater than this ID and that also applies to the given subscription. |

| | |
|---|---|
| acknowledgedRecordId | Acknowledges a record. Notifies the Data Router that the Data Router can purge this record and all previous records from the database. If this parameter is not provided, then the Data Router automatically acknowledges data received in the previous request. This value can be set to -1 to indicate no data should be acknowledged. |
| limit | The maximum number of records to send in the response. The maximum value for this limit is 1000. The default value is 100. |

**Response Example**


```
{
    "lastRecordId": "12345",
    "recordCount": "100",
    "nextUrl":
"https://localhost:8002/api/v1/data/
5b9012bc8e4440c5b2de065eed115690&nextRecordId=12446",
    "records": [
        {
            "channel": "/dms",
            "topic": "TestList",
            "connector": "DMS",
            "record": {
                ...
            }
        }
    ]
}
```


The nextUrl property indicates that there were more records available for the subscription at the time the response was created. If there were no more records available, then the 'nextUrl' element is not included. This property is useful for pagination.

# Using Push (websocket)

The push mechanism for retrieving data from the Data Router enables a client to receive real-time data without the need for polling. Once a websocket connection is established, the Data Router pushes data to the client. Any older data will be quickly pushed to the subscriber until the subscription has caught up with the current records. When new records arrive, the new records will be pushed to the subscriber. The format for the websocket connection is as follows:

| | |
|---|---|
| **Description** | **Opens a websocket to retrieve data for the given subscription ID.** |
| URL | wss://<address>:<port>/ws/data/{subscriptionId}?user=<username>&pass=<APIKey>&nextRecordId=<id> |
| Method | GET |
| **Headers** | |
| Authorization | Basic <encoded username and password>. One of either the username and password query parameters or the basic authorization header must be provided |
| **Optional Query Parameters** | |
| nextRecordId | Sets the Data Router position in the record stream. If this parameter is not provided, then the Data Router continues at the position where the Data Router last left off. The first record in the response has an ID that is equal to or greater than this ID and that also applies to the given subscription. |
| limit | The maximum number of records to send in a single websocket message. The maximum value for this limit is 1000. The default is 100. |
| user | Username obtained from the Data Router web-based configuration utility. Either the username and password or the basic authorization headers must be provided. |
| pass | API key obtained from the Data Router web-based configuration utility. Either the username and password or the basic authorization header must be provided. |

### Response Example

```
{
    "lastRecordId": "12345",
    "recordCount": "100",
    "records"[
        {
            "channel": "/dms",
            "topic": "TestList",
            "connector": "DMS",
            "record": {
                ...
            }
        }
    ]
}
```

### Acknowledgment Example

This message is sent from the subscriber to the Data Router to acknowledge records, indicating to the Data Router that the identified record and all prior records can be purged. If an acknowledgment is not sent, then the Data Router retains records for 7 days.

```
{
    "acknowledgedRecordId": "12345"
}
```

The websocket does not require a JWT to gain access to the endpoint. Using a JWT would be unnecessary since, once the websocket is connected, the websocket stays connected until disconnected by the subscriber. Instead either the basic authorization header is provided, or the username and password query parameters are provided. The basic authorization header must conform with RFC 7617. The username and password, both in the basic authorization header and the query parameters, are obtained from the user manager in the Data Router web-based configuration utility.

The response is an example of a message that the subscriber receives from the connected websocket. Unlike every other HTTP request in this document, the HTTP request does not represent the body of the response. Unlike the pull request, the response message does not include the nextUrl element since the websocket connection automatically sends the latest data.

The subscriber must acknowledge received data. The subscriber does this by sending an acknowledgment message back to the Data Router through the websocket. This message is in JSON and an example is provided in the JWT Response in *Using the Push (websocket) Interface*. The acknowledgment message can be sent at any time. For example, the subscriber could batch many received records into a single database transaction. When the transaction successfully completes, the subscriber can then send a single acknowledgment message indicating the Data Router may purge those records from its database.

A successful push connection results in an HTTP 101 Switching Protocols status code. An unsuccessful push connection results in an HTTP 401 Unauthorized, or 404 Not Found.

# About Monitoring

The Data Router exposes a SignalR hub for monitoring the status of connectors in real time. The hub does not require authorization.

| | |
|---|---|
| **Description** | **A SignalR hub for real-time monitoring of connector status.** |
| URL | https://<host>:<port>/signalr |
| Hub name | ConnectorStatusHub |
| **Client Methods** | void SendConnectorStatus(ConnectorStatus) |
| | void SendConnectorStatuses(IEnumerable<ConnectorStatus>) |

On the first connection to the hub, the Data Router calls the SendConnectorStatuses method, providing the status of all configured connectors. When a change to the status of a connector occurs, the Data Router automatically sends an update. Each update results in a call to the SendConnectorStatus method on the client. This method receives a single argument representing the connector's status. The ConnectorStatusis an object with the properties provided in the following table.

| Property Name | Data Type | Description |
|---|---|---|
| Name | string | Name of the connector for which the status has updated. Connector name comes from the Data Router web-based configuration utility. |
| Channels | string[] | List of channel names that may be affected by this connector status change. Note that multiple connectors can send data to a single channel, and that a single connector can send data to multiple channels. |

| Property Name | Data Type | Description |
|---|---|---|
| Status | string | One of the following values:<br>• Connected<br>Connector has connected to the corresponding data provider.<br>• Disconnected<br>Connector has disconnected from its corresponding data provider.<br>• Disabled<br>Connector has been disabled in the web-based configuration utility.<br>• Deleted<br>Connector has been deleted. |

### Example

```
{
    "Name": "DMS",
    "Channels": [
        "/dms/raw",
        "/dms"
    ],
    "Status": "Connected",
}
```

# 3 Sample Code

This section provides sample code for connecting to and retrieving data from the Data Router. Adapt the sample code to your specific application. The code must be tested by a third party. This code is not production ready and is optimized for brevity and clarity.

## Obtaining an Access Token

The access token is necessary to authorize with other endpoints. The following snippet of code demonstrates a method to obtain an access token:

```
Regex ACCESS_TOKEN_REX = new Regex("\"access_token\"\\s*:\\s*\"([^\"]+)\"");

using (HttpClient client = new HttpClient())
{
   client.BaseAddress = new Uri("https://localhost:8002");

   var accessTokenRequestData = new List<KeyValuePair<string, string>>()
   {
      new KeyValuePair<string, string>("grant_type", "client_credentials"),
      new KeyValuePair<string, string>("client_id", "<user>"),
      new KeyValuePair<string, string>("client_secret", "<api key>")
   };
   var accessTokenRequest = new HttpRequestMessage(HttpMethod.Post, "oauth2/token");
   accessTokenRequest.Content = new FormUrlEncodedContent(accessTokenRequestData);
   HttpResponseMessage accessTokenResponse = client.SendAsync(accessTokenRequest).Result;
   string accessTokenResult = accessTokenResponse.Content.ReadAsStringAsync().Result;
   Match tokenMatch = ACCESS_TOKEN_REX.Match(accessTokenResult);
   string accessToken = tokenMatch.Groups[1].Value;
}
```

# Creating a Subscription

A subscription requires an access token. The subscription need only be created once. Afterward the subscription ID can be reused. The following is an example of subscription creation code:

```
Regex SUBSCRIPTION_REX = new Regex("\"id\"\\s*:\\s*\"([^\"]+)\"");

using (HttpClient client = new HttpClient())
{
   client.BaseAddress = new Uri("https://localhost:8002");

   var subscriptionRequest = new HttpRequestMessage(HttpMethod.Post,
      "api/v1/subscription");
   subscriptionRequest.Headers.Add("Authorization", $"Bearer {accessToken}");
   object[] subscription = new object[]
   {
      new
      {
         ChannelName = "/dms",
         TopicNames = new string[] { "all" }
      }
   };
   subscriptionRequest.Content = new StringContent(
      JsonConvert.SerializeObject(subscription), Encoding.UTF8, "application/json");
   HttpResponseMessage subscriptionResponse =
      client.SendAsync(subscriptionRequest).Result;
   string subscriptionResult = subscriptionResponse.Content.ReadAsStringAsync().Result;
   Match subscriptionMatch = SUBSCRIPTION_REX.Match(subscriptionResult);
   string subscriptionId = subscriptionMatch.Groups[1].Value;
}
```

## Using the Pull Endpoint (HTTP GET)

The following code demonstrates obtaining records using the pull endpoint. An access token and subscription ID are required. This sample code does not show a retry mechanism for an expired token.

```
using (HttpClient client = new HttpClient())
{
   client.BaseAddress = new Uri("https://localhost:8002");

   var dataRequest = new HttpRequestMessage(HttpMethod.Get,
      $"api/v1/data/{subscriptionId}");
   dataRequest.Headers.Add("Authorization", $"Bearer {accessToken}");
   HttpResponseMessage dataResponse = client.SendAsync(dataRequest).Result;
   string dataResult = dataResponse.Content.ReadAsStringAsync().Result;
}
```

## Using the Push (websocket) Interface

The following section of code demonstrates obtaining records from the push (websocket) interface. The code uses basic authorization. Bearer authorization with a JWT is not supported. In this example, the websocket is opened, read from, and then closed. In a production example, the websocket remains open.

```
var websocket = new System.Net.WebSockets.ClientWebSocket();
string encoded = System.Convert.ToBase64String(
    Encoding.ASCII.GetBytes($"{username}:{password}"));
websocket.Options.SetRequestHeader("Authorization", $"Basic {encoded}");
await websocket.ConnectAsync(new Uri($"wss://localhost:8002/ws/data/{subscriptionId}"),
    cancelToken);

byte[] buffer = new byte[65536];
var segment = new ArraySegment<byte>(buffer, 0, buffer.Length);
System.Net.WebSockets.WebSocketReceiveResult receiveResult;
receiveResult = await websocket.ReceiveAsync(segment, cancelToken);
string result = Encoding.UTF8.GetString(buffer, 0, receiveResult.Count);

await websocket.CloseAsync(System.Net.WebSockets.WebSocketCloseStatus.NormalClosure,
    string.Empty, cancelToken);
```