Kafka Consumer Connector

Read Apache Kafka topics into FrameworX tags.

Available from FrameworX 10.1.6. Earlier versions do not include this connector.

- Name: **KafkaConsumer**
- Version 1.0.0.0
- Protocol: Kafka (consumer)
- Interface: TCP/IP
- Runtime: Windows x64, .NET Framework 4.8 and .NET 10
- Available from: FrameworX 10.1.6
- Configuration:
  - Devices / Protocols

Overview
Prerequisites
Procedure
Channel Configuration
Node Configuration
Points Configuration
Message Mapping
Delivery Behavior
Station Examples
Platform Support
Troubleshooting
Third-Party Components
Driver Revision History
Related Pages

Overview

The Kafka Consumer connector reads messages from an Apache Kafka cluster into FrameworX tags. Each point subscribes to a topic and takes the latest matching message: the whole message, one field of a JSON message, or the value, quality and timestamp of a message from the Kafka Producer Connector. Each tag holds the latest value of its topic, not a log of every message. The connector does not create tags from topics.

This page covers the channel, node and point settings, the message mapping and delivery behavior. To publish tag values to Kafka, see Kafka Producer Connector. For MQTT brokers, see MQTT Client Connector. For general channel, node and point concepts, see Devices Module.

Communication Driver Information	
Driver name	KafkaConsumer
Assembly Name	T.ProtocolDriver.KafkaConsumer
Assembly Version	1.0.0.0
Protocol description	Kafka Consumer Connector
Available for Linux	False
Direction	Ingress only (Kafka to FrameworX)

Prerequisites

- FrameworX 10.1.6 or later.
- A Windows x64 computer running the FrameworX runtime. See *Platform Support* below for Windows ARM64.
- The Microsoft Visual C++ 2015-2022 x64 runtime. The FrameworX installer installs it by default. If you clear this component in a Custom install and the computer does not have the runtime, the connector reads no messages.
- An Apache Kafka broker reachable from the runtime computer on its listener port, for example `9092`.
- The topics to read exist on the broker.
- For SASL authentication or a cluster with access control: a Kafka user with read permission on the topics and on the consumer group of the node. See *Consumer Group* below for the group name.

Procedure

1. In **Devices Channels**, create a channel and select the protocol **Kafka Consumer Connector**.
2. Open the channel **Protocol Options** and set the consumer group, start position, payload format and timestamp source. See *Channel Configuration*.
3. In **Devices Nodes**, create a node on the channel and set the **Station** fields: broker list, security protocol and credentials. See *Node Configuration*.

4. In **Devices Points**, add one point per tag to update. Set **Address** to `Topic;Key;ValuePath` and **AccessType** to a read type with polling, for example `Read`. See *Points Configuration*.
5. Start the runtime. On each polling cycle, each tag receives the latest matching message.

Channel Configuration

Protocol Options

Field order in the options string: `GroupId;AutoOffsetReset;PayloadFormat;TimestampSource`

Option	Values	Default	Description
GroupId	Text, or blank	(blank)	Kafka consumer group. See *Consumer Group*.
AutoOffsetReset	`Latest`, `Earliest`	`Latest`	`Latest` resumes from the last position committed by the consumer group, or from the end of the topic when the group has no committed position. After a restart, a tag keeps its startup value until the next message arrives. `Earliest` reads every retained message of the topic on each start and after each partition reassignment, for example when a point with a new topic is added online. The tags keep their values until the read completes. With a compacted topic, this restores the last value of each key at start.
PayloadFormat	`Auto`, `Text`	`Auto`	`Auto` applies the JSON mapping in *Message Mapping*. `Text` writes the whole message to the tag as a string.
TimestampSource	`Payload`, `Kafka`, `Arrival`	`Payload`	`Payload` uses the `timestamp` field of a Kafka Producer Connector message, and the Kafka record time for any other message. `Kafka` uses the Kafka record time. `Arrival` uses the time the runtime received the message.

The runtime reads the Protocol Options at start. After a change, restart the runtime.

Options string examples:

```
Default values:                                ;Latest;Auto;Payload
Restore the last value of each key at start:   ;Earliest;Auto;Payload
Named group, text payload, arrival time:       PLANT1.SCADA;Latest;Text;Arrival
```

Consumer Group

Each node consumes as its own Kafka consumer group. With `Latest`, the group keeps the read position of the node, so a restart resumes where the node stopped.

- **GroupId blank:** the group name is `FrameworX.<ComputerName>.<SolutionName>.<CHANNEL>.<NODE>`, with the channel and node names in upper case. Two solutions, or two computers, reading the same topic therefore use different groups, and each receives every message.
- **GroupId set:** the group name is `<GROUPID>.<NODE>`. The runtime converts the GroupId to upper case, so `plant1.scada` becomes `PLANT1.SCADA`. Do not use = or `;` in the GroupId. Grant the Kafka read permission to the upper-case group name.
- Two runtimes of the same solution on the same computer, for example a Development and a Production profile, share the default group and split the topic partitions between them. Set a different GroupId on one of them.

Node Configuration

Station Configuration

Station syntax: **<BootstrapServers>;<SecurityProtocol>;<SaslMechanism>;[SaslUsername];[SaslPassword];[SslCaLocation]**

The Station fields are the same as in the Kafka Producer Connector, so a producer node and a consumer node of the same cluster use the same Station string. The node uses its primary Station only. The connector does not switch to a backup Station.

Field	Values	Default	Description
BootstrapServers	`host:port` list	`localhost:9092`	Comma-separated list of brokers, for example `broker1:9092,broker2:9092`. Required. The client discovers the rest of the cluster from these brokers.
SecurityProtocol	`Plaintext`, `Ssl`, `SaslPlaintext`, `SaslSsl`	`Plaintext`	Security of the broker connection. `Ssl` and `SaslSsl` use TLS. `SaslPlaintext` and `SaslSsl` use SASL authentication. Match the broker listener on the port in **BootstrapServers**.
SaslMechanism	`Plain`, `ScramSha256`, `ScramSha512`	`Plain`	SASL mechanism. Used only when **SecurityProtocol** is `SaslPlaintext` or `SaslSsl`.

SaslUsername	Text	(blank)	SASL user name. Used only with `SaslPlaintext` or `SaslSsl`.
SaslPassword	Password	(blank)	SASL password. The Designer stores it encoded. Used only with `SaslPlaintext` or `SaslSsl`.
SslCaLocation	File path	(blank)	Path to a PEM file with the CA certificate, or certificate chain, of the broker, for example `C:\Kafka\certs\ca.pem`. Used with `Ssl` and `SaslSsl`. When blank, the client uses the Trusted Root Certification Authorities store of Windows.

Enter **SaslPassword** in the Designer station editor. The station string separates fields with semicolons, and the Designer encoding keeps a password with a semicolon in one field. A plain-text password with a semicolon, or one starting with #, written to the station string by an import or a script, reaches the broker altered.

Points Configuration

Address

Address syntax: **<Topic>;[Key];[ValuePath]**

Field	Default	Description
Topic	(required)	Kafka topic to read. Letters, digits, `.`, _ and `-`, up to 249 characters. The runtime rejects a point with any other character in the topic name and writes the reason to the Trace Window.
Key	(blank)	Blank: every message of the topic. Set: only messages with this key. A message without a key matches when its JSON `tag` field equals the Key, so messages from a Kafka Producer Connector with **KeyStrategy** `None` still match. The comparison is case-sensitive.
Value Path	(blank)	Blank: the mapping rules in *Message Mapping*. Set: a JSONPath into the message, for example `$.temp` or `$.line1.speed`. The runtime rejects a point with an invalid JSONPath.

Address examples:

```
Messages with key Line1Temp on topic plant.line1:   plant.line1;Line1Temp;
Every message on topic sensors.raw:                  sensors.raw;;
Field temp of the JSON messages on sensors.json:     sensors.json;;$.temp
```

AccessType

Assign a read AccessType with polling, for example the predefined `Read` type. The polling period is the tag update period: on each polling cycle the tag receives the latest matching message received since start. The connector ignores writes to its points. See Devices AccessTypes Reference.

Message Mapping

For each point, the connector maps a matching message to a value, a quality and a timestamp, in this order:

1. **PayloadFormat `Text`:** the whole message as a string, quality Good (192). The ValuePath is not used.
2. **ValuePath set:** the JSON value at the ValuePath, quality Good. A message without a value at this path, or not in JSON format, leaves the tag unchanged.
3. **Message from the Kafka Producer Connector:** a JSON object with `tag` and `value` fields. The tag receives `value`, the `quality` field when it is an integer, and the `timestamp` field with TimestampSource `Payload`.
4. **Any other message:** the whole message as a string, quality Good.

Producer message example:

```
{"tag":"K_PlainCounter","value":265,"quality":192,"timestamp":"2026-10-08T10:11:00.6674091Z"}
```

- **Types:** JSON numbers become integer or floating-point values, `true` and `false` become Boolean values, strings stay strings, and JSON objects and arrays reach the tag as JSON text. The runtime converts the value to the tag type.
- **Not mapped:** a JSON `null` value, and a Kafka record without a payload, for example a delete marker in a compacted topic, leave the tag unchanged.
- **Dates in JSON strings** stay strings.
- **Quality:** the connector uses the `quality` field of Kafka Producer Connector messages only. For other messages it sets Good.

Delivery Behavior

- **Latest value per tag.** Between two polling cycles, the last matching message wins. Intermediate values between two cycles do not reach the tag or the historian. A shorter polling period skips fewer values. To record every message, read the topic with another Kafka client.
- **Message order.** Kafka orders messages within a partition. When the messages of one point come from several partitions, for example a blank Key on a multi-partition topic, the tag keeps the message with the newest Kafka record time.
- **Catch-up at start.** When the node starts, or Kafka reassigns its partitions, the connector reads up to the end of each partition before it updates the tags. A replay with `Earliest` therefore writes only the last value of each point, not the intermediate values. A partition without data or end-of-partition for 10 seconds counts as read.
- **Startup.** Until Kafka assigns the partitions to the node, the tags keep their startup value and quality. When no assignment arrives within 30 seconds, the tags go Bad after the channel retries.
- **Connection loss.** When the client loses all brokers, or the broker rejects the credentials, the tags go Bad after the channel retries. While the brokers are unreachable, the connector checks the cluster every 2 seconds. When the connection returns, the tags go Good again with the recovery time as timestamp, so the historian keeps the Bad sample before the Good one. A point with no message since start stays Bad until its first message.
- **Read position.** The consumer group commits the read position on its own. A message read but not yet applied to a tag when the runtime stops is not read again with `Latest`.
- **Timestamps of replayed values.** A value replayed at start, or received after a connection loss, carries the start or recovery time when its own timestamp is older. This applies to every TimestampSource.
- **Configuration changes.** When the node Station changes, the connector rebuilds the consumer with the new settings. A change of **BootstrapServers** also discards the values received from the previous cluster. Each tag keeps its current value until a message from the new cluster arrives.
- **Loops.** Do not read and publish the same tag on the same topic.

Station Examples

1. Local Apache Kafka, no authentication, no TLS

Field	Value
BootstrapServers	`localhost:9092`
SecurityProtocol	`Plaintext`
SaslMechanism	`Plain` (not used)
SaslUsername	(blank)
SaslPassword	(blank)
SslCaLocation	(blank)

Station string:

```
localhost:9092;Plaintext;Plain;;;
```

Channel Protocol Options: the default values. Point: Address `sensors.raw;;`, AccessType `Read`.

2. Apache Kafka with SASL_SSL, SCRAM-SHA-256 and a private CA file

Field	Value
BootstrapServers	`kafka01.plant.local:9093`
SecurityProtocol	`SaslSsl`
SaslMechanism	`ScramSha256`
SaslUsername	`fxconsumer`
SaslPassword	The password of `fxconsumer`, entered in the Designer station editor
SslCaLocation	`C:\Kafka\certs\ca.pem`

The broker listener on port `9093` uses SASL_SSL with `SCRAM-SHA-256`, and `ca.pem` holds the CA certificate in PEM format. The broker certificate lists `kafka01.plant.local` as a host name. Create the SCRAM user on the broker with `kafka-configs`, and grant it read permission on the topics and on the consumer group, before you start the runtime.

Platform Support

Platform	Supported
Windows x64, .NET Framework 4.8 runtime	Yes
Windows x64, .NET 10 runtime	Yes
Windows ARM64, .NET Framework 4.8 runtime	Yes, under x64 emulation
Windows ARM64, .NET 10 runtime started with the x64 `dotnet` host	Yes, under x64 emulation
Windows ARM64, .NET 10 runtime with the default ARM64 `dotnet` host	No
Linux, any architecture	No

The connector depends on the native Kafka client library for Windows x64. On a host where this library does not load, the node stays Bad.

Troubleshooting

The connector writes these messages to the Trace Window (see Runtime Diagnostics Reference). Check Devices in the Trace Window settings. Messages marked (Debug) appear only when you also check Debug. For rows with a quoted message, the first column shows the start of the message.

Symptom	Likely Cause	Resolution
`Kafka consumer: BootstrapServers cannot be empty`, the node does not start. After a Station edit: `Kafka consumer node '<node>' is down: BootstrapServers is empty`	The first Station field is blank.	Enter at least one broker as `host:port`.
`Kafka consumer: point '<address>' of tag '<tag>': invalid topic` or `invalid ValuePath`	The topic name has a character Kafka does not accept, or the ValuePath is not a valid JSONPath.	Correct the point Address. See *Address*.
`Kafka consumer node '<node>' is down: ... (Local_AllBrokersDown)`, tags Bad	No broker reachable: broker stopped, wrong host or port, or TLS settings different from the broker listener.	Confirm the runtime computer reaches the broker port. Match **SecurityProtocol** to the listener on the broker port. Tags with a received message return to Good when the cluster is reachable again, with no runtime restart.
`Kafka consumer node '<node>' is down: ... (SaslAuthenticationFailed)` or `(Local_Authentication)`	Wrong SASL user name, password or mechanism.	Check **SaslUsername**, **SaslPassword** and **SaslMechanism** against the broker user.
`Kafka consumer node '<node>' is down: no partition assignment within 30 s`	Kafka did not assign partitions to the node. Common causes: the topics do not exist, the user has no read permission on the consumer group, or the group waits for a previous runtime instance after a forced stop.	Create the topics. Grant read permission on the consumer group name, in upper case when you set a GroupId (see *Consumer Group*). After a forced stop of the runtime, Kafka releases the previous group member within 45 seconds by default. After the assignment, each tag returns to Good on its next message, or at start with `Earliest`.
`Kafka consumer node '<node>' is down: failed to create the consumer`	The client rejected the configuration, for example an **SslCaLocation** file missing or unreadable. Or the native Kafka client library did not load, see the Windows ARM64 and Visual C++ rows.	Check the reason at the end of the message. The connector retries every 5 seconds, so a corrected Station or CA file takes effect without a restart.
`Kafka consumer error on node '<node>'` (Error for a missing topic, Debug for a broker connection error)	The client reported an error for a topic, for example a subscribed topic missing on the broker, or a connection error to one broker. The client keeps retrying.	Read the reason at the end of the message. Create the missing topic, or check the broker named in the message.
`Kafka consumer node '<node>': partition ... counted as caught up` (Warning)	A partition sent neither data nor an end-of-partition signal for 10 seconds after an assignment.	No action when the tags update. Check the broker health when the message repeats.
`Kafka consumer node '<node>': message on topic '<topic>' is not valid JSON`	The point has a ValuePath, and the message is not JSON or has no value at this path. Topics carrying several	No action for mixed topics. Otherwise check the ValuePath against a message read with `kafka-console-consumer`.

or `... has no value at ValuePath` (Debug)	message layouts produce this message for the points of the other layouts.	
A tag keeps its startup value after a restart, no error	**AutoOffsetReset** is `Latest` and no new message arrived since the start.	Expected with `Latest`. To restore the last value at start, publish to a compacted topic and set `Earliest`.
A tag never updates, no error	The Key does not match the message key, including case, or the point AccessType has no read polling.	Read the message keys with `kafka-console-consumer --property print.key=true` and set the Key to the same text. Assign a read AccessType with polling.
TLS handshake fails with `certificate verify failed` in a `Kafka consumer error on node` message (Debug), CA file correct	The broker certificate does not list the host name used in **BootstrapServers**. The client checks the host name of the broker certificate.	Use the host name from the broker certificate in **BootstrapServers**, or issue a broker certificate with this host name in its Subject Alternative Name.
TLS handshake fails with a broker certificate trusted by other clients	The TLS library rejects certificates with RSA keys shorter than 2048 bits. It also does not read a system `openssl.cnf` file or the `OPENSSL_CONF` variable.	Use a broker certificate with an RSA key of 2048 bits or more. Set the CA file in **SslCaLocation** instead of an OpenSSL configuration file.
`failed to create the consumer` with a native library load error, on Windows ARM64 with the .NET 10 runtime	The default ARM64 `dotnet` host does not load the x64 Kafka client library.	Use the .NET Framework 4.8 runtime, or start the .NET 10 runtime with the x64 `dotnet` host.
`failed to create the consumer` with a native library load error, on any runtime	The Visual C++ 2015-2022 x64 runtime is missing.	Install the Microsoft Visual C++ 2015-2022 Redistributable (x64).

Third-Party Components

The connector uses the same components as the Kafka Producer Connector, installed in the `Protocols` folder of the FrameworX installation:

Component	Version
Confluent.Kafka (.NET client)	2.15.1
librdkafka	2.15.1
OpenSSL	3.5.9 LTS
libcurl	8.23.0
zlib	1.3.2
Zstandard (zstd)	1.5.7

License notices for these components ship in `<FrameworX installation>\Protocols\Kafka-ThirdPartyNotices.txt`.

Driver Revision History

KafkaConsumer Revision History	
Version	**Notes**
1.0.0.0	Initial release in FrameworX 10.1.6.

Related Pages

- Kafka Producer Connector
- Devices Module
- Devices Module Reference
- Devices AccessTypes Reference
- Runtime Diagnostics Reference
- MQTT Client Connector
- IT-Cloud and AI Connectors

In this section...