SmartSpace®

Kafka Integration with ObjectView

From version 3.9 SP1

Ubisense Limited, St Andrew's House, St Andrew's Road, Cambridge CB4 1DL, United Kingdom.

Telephone: +44 (0)1223 535170. Website: https://www.ubisense.com

Ubisense Limited
St Andrew's House
St Andrew's Road
Cambridge CB4 1DL
United Kingdom

Tel: +44 (0)1223 535170

WWW: *https://www.ubisense.com*

Contents

Kafka integration with ObjectView

Installing the Kafka feature

The Kafka feature requires SmartSpace version 3.9 or above.

To install the Kafka features:

1. Make sure that the SmartSpace platform includes a license for the correct version of Applications integration.
2. Install the Kafka feature using Service Manager.
3. Install the ObjectView admin tools.

For further information on all aspects of installation, see SmartSpace Installation.

See *Kafka quickstart* if you do not have a Kafka cluster.

Additional requirements for Linux installations

You must install **librdkafka** (version 2.3 or above) so that services can find **librdkafka.so.1**.

Additional requirements for Windows installations

You must install OpenSSL 3.x libraries and add them to your system PATH. For example, when downloading the libraries from *FireDaemon*, use the MSI installer and tick the **Adjust PATH system environment** box.

Configuration parameters

For Kafka setup, there are two standard Ubisense configuration parameters. If your cluster does not have security and you want to use only default values for the Kafka producer configuration, set **kafka_brokers** to point to the cluster, then the value is passed to the Kafka **bootstrap.servers** property. For all other cases, set **kafka_producer_config** to point to a file containing key-value pairs for any keys described in *Kafka Producer Configuration Reference*.

Parameter	Meaning	Default value
kafka_ brokers	The value passed to bootstrap.servers	127.0.0.1:9092
kafka_ producer_ config	Path to a file containing key-value pairs of producer configuration	<empty>

Note: **kafka_producer_config** is a local config param.

If the producer config file contains a value for **bootstrap.servers**, it takes precedence over the value of **kafka_brokers**, so you do not need to configure both parameters. See below for an example of the producer config file:

```
# list of brokers used for bootstrapping knowledge about the rest of the cluster
# format: host1:port1,host2:port2 ...
bootstrap.servers=localhost:9092

# specify the compression codec for all data generated: none, gzip, snappy, lz4, zstd
compression.type=none

# name of the partitioner class for partitioning records;
# The default uses "sticky" partitioning logic which spreads the load evenly between
    ↪ partitions, but improves throughput by attempting to fill the batches sent to
each
    ↪ partition.
#partitioner.class=

# the maximum amount of time the client will wait for the response of a request
#request.timeout.ms=

# how long `KafkaProducer.send` and `KafkaProducer.partitionsFor` will block for
#max.block.ms=

# the producer will wait for up to the given delay to allow other records to be sent
so
    ↪ that the sends can be batched together
#linger.ms=

# the maximum size of a request in bytes
#max.request.size=

# the default batch size in bytes when batching multiple records sent to a partition
#batch.size=

# the total bytes of memory the producer can use to buffer records waiting to be sent
to
    ↪ the server
#buffer.memory=
```

Specifying Kafka in ObjectView definitions

Kafka integration uses ObjectView definitions. View the ObjectView API documentation for full details of how to configure data and location views that can be published to Kafka.

Once you have configured data and location views, edit the file to send messages to Kafka. In the view definition, add an "**integration**" field:

```
{
        "view": "Stations",
        "integration": {
                "transport": "kafka"
        },
        "key": "Station",
        "properties": {
                "lane": "lane number",
                "seq": "sequence number",
                "cars": {
                        "source": "'Car' is in 'Station'",
                        "key": "1",
                        "value": "0"
                }
        },
        "roles": [
                "Any"
        ]
}
```

The supported transport mechanisms (case-insensitive) are:

- "kafka"
- "monitor" (serialized JSON objects in a single-line)

  The monitor mechanisms enable and send messages to standard platform_monitor streams.
- "pretty monitor" (same as monitor, but formatted over multiple lines)

  pretty monitor will not work for large amounts of data because monitor streams use UDP. However, it is very useful for smaller-scale testing and debugging.

> The "**roles**" field configures which users are allowed to access the object view from the web site, via the ObjectView API. All ObjectViews need this field, but Kafka integration ignores it. If the view should be unavailable to the ObjectView API, and only sent to Kafka, use a single role that is not defined in "USERS/ROLES".

Example data model

To use the view definition in this section and to follow along with the examples below, create the following simple data model.

```
ubisense_udm_admin input:

use namespace Custom;
declare type Car : Object;
declare type Station : Object;
declare name property VIN <Car>;
declare name property name <Station>;
declare property lane number<Station> : Int;
declare property sequence number<Station> : Int;
declare property <Car> is in <Station> : Bool;
new Car "car1";
new Car "car2";
new Car "car3";
new Station "station1";
new Station "station2";
new Station "station3";
```

Kafka messages

Data-based views

The default behavior is as follows:

On establish

When first defining a view, or when ObjectView integration services restart:

- Send a message telling consumers that they need to discard any stored data
- For each object in the view, send a message containing the object data

On change

When data changes. For example, when a car enters or leaves a station:

- For the object whose data has changed, send a message containing all the object data

Message format

If you imported the *Example data model* into a new dataset, there are no property rows to show yet. Generate an event using **ubisense_udm_admin**.

```
> set "car1" is in "station1" = true;
```

When an event is generated, a message is sent to Kafka. Based on the example data model and generated event above, a message will look similar to the following:

```
Topic : Stations
Key : {
       "cell": "04007zd9HKU6T8EE000FCW00002:UCell::Cell",
       "id": "04007zd9HMiB0dmw0009EW0000d:UserDataModel::[Custom]Station"
      }
Value : {
       "cell": "Site",
       "seq": 2007088076,
       "val": {
               "cars": [
                   "car1"
               ]
       }
}
```

Topic

The Kafka topic defaults to the view name. Spaces are automatically converted to underscores because they are not allowed in Kafka topic names. You can configure the topic by adding the optional "**topic**" field to the "**integration**" object, so you can send messages from different views to the same topic, if required.

```
{
       "view": "Stations",
       "integration": {
               "topic": "SmartSpace",
               "transport": "kafka"
       },
       "key": "Station",
       "properties": {
               "lane": "lane number",
               "seq": "sequence number",
               "cars": {
                       "source": "'Car' is in 'Station'",
                       "key": "1",
                       "value": "0"
               }
       },
       "roles": [
               "Any"
       ]
}
```

Key

The key is a JSON object that includes the internal ID of the Ubisense object whose value has changed. For data views like the one above, there is also a "**cell**" field. For properties managed by the site-level object property data store or the site-level assertion store, this will always be the site

cell. For properties managed by the cell-level object property data service, this will be the relevant geometry cell.

Although Kafka consumers are assumed not to have access to Ubisense schema data, the key uses internal IDs because they are unique and immutable for the lifetime of the object, which is not the case for object names.

Value

By default, messages contain all the data for the object in the view, so you can turn on *Kafka data compaction* without any risk of losing data. On any change, a new message is sent, containing all the properties in the view, superseding all messages with the same key.

If a value is a named object, the human-readable name is used. This is assumed to be more useful to Kafka consumers than the internal ID because if they had direct access to Ubisense schemas, they would not be using Kafka in the first place.

Advanced configuration to handle large arrays

Since Kafka is designed for high throughput, the default behavior of sending all the object data on every change will normally work fine, but may not always be appropriate. In particular, if there are thousands of complex property rows, the resulting JSON array will be large. To avoid sending the entire array on every change, you can configure the view to send only the changes. Depending on the application, consumers might then need to maintain the view data themselves.

The **"integration"** field has the following optional members:

Field	Value	Defualt
establish	Possible values are 'data', 'event', 'all', 'none'	data
change	Possible values are 'data', 'event', 'all', 'none'	data

To configure the view to send only the changes, add **"change": "event"** to the **"integration"** field. For example:

```
{
        "view": "Stations",
        "integration": {
                "change": "event",
                "transport": "kafka"
        },
        "key": "Station",
        "properties": {
                "lane": "lane number",
                "seq": "sequence number",
                "cars": {
                        "source": "'Car' is in 'Station'",
                        "key": "1",
                        "value": "0"
                }
        },
        "roles": [
                "Any"
        ]
}
```

Now, when the view changes, you get a much smaller message detailing the change. Note, however, that in general, for each station in the view, there will not be any Kafka message containing all the data.

When using change messages rather than full data messages, consumers may need to maintain the data themselves. Use Kafka data compaction with care for these topics.

The value is now a JSON object that includes the view, sequence number (note that this is not the property sequence number of the UDM in the example), property, and type of event. The fields are identical to those passed to an object view **onChange** callback.

Field	Value
type	Possible values are 'ins', 'upd', 'del'
prop	The property name within the document
idx	If the property is an array values property, the index of the value that has changed in that array
val	The value being inserted, updated, or deleted
old	If this is an update, the value being replaced

For example:

```
Topic : Stations
Key : {
       "cell": "04007zd9HKU6T8EE000FCW00002:UCell::Cell",
       "id": "04007zd9HMiB0dmw0009EW0000d:UserDataModel::[Custom]Station"
      }
Value : {
       "idx": 1,
       "prop": "cars",
       "seq": 2007153619,
       "type": "ins",
       "val": "car2",
       "view": "Stations"
       }
```

Event-based views

Sometimes, properties can be used as a way simply to deliver messages, and you don't really need a full view of all the property rows. For example, the Location simulation feature includes an assertion **<Object> reached waypoint <Object>**. This is a transient assertion that goes away automatically: it can be useful to receive a message when an object reaches a waypoint, but you may not want to maintain the complete view of objects that have reached waypoints.

In this case, you can add "**establish**": "**none**" to the configuration, and Kafka will only get change messages.

```
{
       "view": "Waypoints",
       "integration": {
              "change": "event",
              "establish": "none",
              "transport": "kafka"
       },
       "key": "Object",
       "properties": {
              "reached": "'Object' reached waypoint 'Object'",
       },
       "roles": [
              "Any"
       ]
}
```

When an object reaches a waypoint, the message will look similar to the following:

```
Topic : Waypoints
Key : {
       "cell": "04007zd9HKU6T8EE000FCW00002:UCell::Cell",
       "id": "04007zd9HMiB0dmw0009EW0000X:UserDataModel::[Custom]Car"
      }
Value : {
       "idx": 0,
       "prop": "reached",
       "seq": 2361868144,
       "type": "ins",
       "val": "station1",
       "view": "Waypoints"
       }
```

Location views

As with data views, location views use the format from the existing HMIs feature. You can specify an object type, or **ULocationIntegration::Tag** for raw tag locations. For example:

```
[
       {
                "view": "Car Locations",
                "integration": {
                        "transport": "kafka"
                },
                "key": "Car",
                "locations": {
                        "period": 5
                },
                "roles": [
                        "Any"
                ]
       },
       {
                "view": "Tag Locations",
                "integration": {
                        "transport": "kafka"
                },
                "key": "ULocationIntegration::Tag",
                "locations": {
                        "period": 5
                },
                "roles": [
                        "Any"
                ]
       }
]
```

There is one additional optional field in the "**integration**" object, "**cell**", which applies to location views only. The default value is "**Site**", but you can set this in order to receive location messages from only a specified geometry or location cell.

Note that locations always come from the site-level aggregation service: any specified cell is just a simple filter applied in the site-level object view integration service.

Debugging

There are two more valid values for the "**transport**" field. In addition to "**kafka**", you can use "**monitor**" and "**pretty monitor**" to send messages to **ubisense_trace_receiver**. The "**transport**" value can be an array of valid values.

```
[
	{
		"view": "Tag Locations",
		"integration": {
			"transport": ["kafka", "monitor"]
		},
		"key": "ULocationIntegration::Tag",
		"locations": {
			"period": 5
		},
		"roles": [
			"Any"
		]
	}
]
```

"**pretty monitor**" prints messages over multiple lines, which is much more readable, but since monitor streams are sent over UDP you are likely to lose some lines. It is recommended to use "**monitor**" over "**pretty monitor**" for detailed debugging.

Messages are automatically sent to the trace stream "**ObjectView__<topic>**". In this example, the trace goes to **ObjectView__Tag_Locations**.

You can enable the additional trace streams "**object_view_integration**" and "**kafka**", if required.