VOLTDB

New Streaming Data Preview: Topics

January, 2021

Beta release for testing and feedback.

Revision History
10.2

Abstract

The following document describes functionality available in VoltDB 10.1.1 and later. This is a beta release designed specifically to solicit feedback. The new capabilities are believed to work as described. However, further testing and real-world evaluation may identify further changes or improvements needed to optimize the functionality and/or syntax.

Your feedback concerning new functionality is critical to improving VoltDB. Please send all comments, questions, and bug reports to *support@voltdb.com*.

Streaming Data: Topics

VoltDB lets you stream data to or from individual external systems using *export* and *import*. When you configure import, data is pulled from the specified import source and passed to a stored procedure for processing. Similarly, when you configure a stream or table as an export source (using the EXPORT TO TARGET clause), any data written to the stream or table is automatically pushed to the specified external target. However, there are times you do not know in advance what systems will need the data or when. There are also times when more than one system can make use of the same stream. These are the use cases for topic streams.

VoltDB version 10.1 introduces topics as a beta feature. When you define a topic specifying a stream and a stored procedure, any data written to the stream is held in a queue that one or more external systems can subscribe to and pull data from as and when they need it. Each subscribed client has its own cursor into the queued data, or if they are members of a group, each group of clients shares a cursor.

> **Note**
>
> To simplify development for streaming applications, VoltDB topics are modeled after Apache Kafka topics and utilize the Kafka consumer and producer interfaces as its client APIs. This gives application developers the ability to reuse existing Kafka consumers and producers — or utilities that abstract the interface, such as KafkaConnect — to connect their client applications to VoltDB topics. The following discussion assumes a basic understanding of Apache Kafka and its interfaces and tools, as described in the Kafka documentation.

Understanding Topics

Topics allow you to integrate both import and export into a single stream. You declare a topic using the CREATE TOPIC statement.

There are actually two distinct and independent components to a topic: input and output. You declare a topic having either or both, depending on which clauses you include in the SQL declaration. For example, you can declare an output-only topic by specifying the USING STREAM clause but specifying no stored procedure. In this case, any records written to the associated stream are queued for output and available to any consumers of the topic:

```
CREATE TOPIC USING STREAM sessions;
```

If, on the other hand, you include the EXECUTE PROCEDURE clause but not USING STREAM, records written to the topic by producers invoke the specified procedure passing the message contents (and, optionally, the key) as arguments:

```
CREATE TOPIC sessions EXECUTE PROCEDURE ProcessSessions;
```

In both cases, before you declare the topic with either the USING STREAM or EXECUTE PROCEDURE the associated stream and/or stored procedure must already exist in the database.

If you include both the USING STREAM and the EXECUTE PROCEDURE clause, the topic is available for both input and output. What happens to the data as it passes through VoltDB is up to you. You can simply pass it from producers to consumers by taking the data received by the input procedure and inserting it into the stream of the same name. Or the stored procedure can filter, modify, or redirect the content as needed. For example, the following data definitions create a topic where the input procedure uses an existing table in the database (*users*) to fill out additional fields based on the matching *username* in the incoming records while writing the data to the stream for output:

```
CREATE TABLE tempuser ( username VARCHAR(128) NOT NULL);
CREATE TABLE users ( username VARCHAR(128)  NOT NULL,
    country VARCHAR(32), userrank INTEGER);
PARTITION TABLE tempuser on column username;
PARTITION TABLE users on column username;

CREATE STREAM sessions
  PARTITION ON COLUMN username (
    username VARCHAR(128) NOT NULL,
    login TIMESTAMP, country VARCHAR(32), userrank INTEGER);

CREATE PROCEDURE ProcessSessions
  PARTITION ON TABLE users COLUMN username
  AS BEGIN
   INSERT INTO tempuser VALUES(CAST(? AS VARCHAR));
   INSERT INTO sessions SELECT u.username,
     CAST(? AS TIMESTAMP), u.country, u.userrank
     FROM users AS u, tempuser AS t
     WHERE u.username=t.username;
   TRUNCATE TABLE tempuser;
  END;

CREATE TOPIC USING STREAM sessions EXECUTE PROCEDURE ProcessSessions;
```

Finally, if you want to create a topic that is not processed but simply flows through VoltDB from producers to consumers, you can define an *opaque topic* using the CREATE OPAQUE TOPIC statement:

```
CREATE OPAQUE TOPIC sysmsgs;
```

Opaque topics are useful if you want to have a single set of brokers for all your topics but only need to analyze and process some of the data feeds. Opaque topics let VoltDB handle the additional topics without requiring the stored procedure or stream definitions needed for processed topics.

Customizing Topic Behavior

In its simplest form, you define a topic by creating the necessary stored procedure and stream to use for input and output and then defining the topic itself, referencing the procedure and stream. For example:

```
CREATE STREAM auction_bids
  PARTITION ON COLUMN bid_id
    (bid_id INTEGER NOT NULL, user_id INTEGER, bid DECIMAL);

CREATE PROCEDURE
  PARTITION ON auction_bids COLUMN bid_id
    FROM CLASS auctionprocs.EvaluateBids;

CREATE TOPIC USING STREAM auction_bids
  EXECUTE PROCEDURE EvalaueBids
    PROFILE bids;
```

The stream must be partitioned and the topic name is the same as the stream name (in this case *auction_bids*). Declaring the topic and its stream and/or procedure are the only required elements for creating a topic. However, there are several other attributes you can specify as part of the declaration, using the ALLOW, PROFILE, and PROPERTIES clauses. Those attributes include:

- Permissions — Specifying which roles users must have to access the topic as consumers
- Retention — Assigning a profile to control the retention policy
- Data Format — Choosing a format for the data passed to the external clients
- Keys — Specifying key columns

Permissions

When security is enabled for the database, the external clients must authenticate using a username and password when they initiate contact with the server. The valid usernames and passwords are stored as part of the database configuration and each user is assigned one or more *roles* that define what actions they are allowed to perform.

VoltDB manages security for topic consumers and producers separately. For producers, the permissions of the stored procedure named in the EXECUTE PROCEDURE statement control who is allowed to write to the topic. For consumers, permissions are defined by the ALLOW clause of the CREATE TOPIC statement. In other words, the user account for consumers must either be assigned an all-inclusive system role, such as ADMIN, or a role that is listed in the ALLOW clause of the CREATE TOPIC statement to be able to read data from the topic.

You can specify multiple security roles in the ALLOW clause, separated by commas. For example, the following stored procedure and topic declarations allow users assigned the role "producer" to write to the *sessions* topic, "subscriber" to read from the topic, and "manager" to both read and write.

```
CREATE PROCEDURE
  PARTITION ON users COLUMN user_id
  ALLOW producer, manager
    FROM CLASS sessionprocs.ValidateUsers;

CREATE TOPIC USING STREAM sessions EXECUTE PROCEDURE ValidateUsers
   ALLOW subscriber, manager;
```

The specified roles must be defined in the database configuration before the topic is created.

Profiles

Assigning a profile to the topic is strongly recommended. A profile is not required, but if you want to control the retention policy for the topic (that is, how long data is available to clients before it is purged) you must specify a

profile. The attributes of the profile itself are defined in the database configuration file, which we will discuss later. For the sake of example, let's assume the topic is assigned to an existing profile called *daily*:

```
CREATE TOPIC USING STREAM sessions EXECUTE PROCEDURE ValidateUsers
    PROFILE daily;
```

Data Format

VoltDB topics are composed of three elements: a timestamp, a record with one or more fields, and an optional set of keys values. The timestamp is generated automatically when the the record is inserted into the stream. By default, all of the columns of the stream record are included in the topic record. However, you can use the properties `consumer.values` and `consumer.keys` to specify which columns of the stream are included as values in the record and/or key for the topic sent to consumers.

For single value records and keys, the data is sent in the native Kafka binary format for that datatype. For multi-value records or keys, VoltDB sends the content as comma-separated values (CSV) in a text string by default. Similarly, on input from producers, the topic record is interpreted as a single binary format value or a CSV string, depending on the datatype of the content itself.

You can control what format is used to send and receive the topic data using the PROPERTIES clause. The topic properties provide for maximum flexibility, allowing you to specify individual formats for input versus output, the topic message versus the keys, etc. In most cases you will want to have the same format for both input and output, which you can specify using the `topic.format` property and specifying one of the supported formats (AVRO, CSV, or JSON). You can replace "topic" with either "consumer" or "producer" to pick the format specifically for output or input, respectively. Or you can add the suffix "value" or "keys" to pick a format for the data content versus the keys.

For example, the following declaration creates the topic content for both input and output as AVRO rather than CSV format:

```
CREATE TOPIC USING STREAM sessions EXECUTE PROCEDURE ValidateUsers
    PROPERTIES (topic.format=avro);
```

When using AVRO format, you must also have access to an AVRO schema registry, which is where VoltDB stores the schema for AVRO-formatted topics. The URL for the registry is specified in the database configuration file, as described in the section called “Configuring the Subscription Service on the Database Server”.

Topic Messages and Keys

Apache Kafka can use one or more fields as keys to partition the records to different brokers. VoltDB lets you select columns from the stream to declare as keys for the topic by specifying them in the `consumer.keys` property. Since Kafka keys are associated with partitioning and VoltDB topics are partitioned by the partitioning column, it is best to always include the partitioning column when you specify topic keys. Separate multiple column names with commas. For example:

```
CREATE TOPIC USING STREAM sessions
    PROPERTIES (consumer.keys='user_id,login_id');
```

By default, VoltDB includes all columns from the stream in the topic message. So if the topic has one or more keys, those values will be repeated: once in the key and once in the topic message itself. If you want to exclude the key values from the topic message, you can explicitly select which columns to use in the topic's message and in what order using the `consumer.values` property. For example, the following declaration excludes the keys from the message itself.

```
CREATE TOPIC USING STREAM sessions
   PROPERTIES (consumer.keys='user_id,login_id',
     consumer.values='login_time,ip_address'
   );
```

Because VoltDB cannot control what data is sent by client producers, the structure of the data is handled differently for output and input. On input, VoltDB interprets the layout of the message fields and the key at runtime based on the data it receives from the producer.

Only one key field is allowed for input. By default, the key is not passed to the specified stored procedure; only the message fields of the topic are passed as parameters to the stored procedure. If you want to include the key in the list of parameters to the stored procedure, you can set the property `producer.parameters.includeKey` to true and the key will be included as the partitioning parameter for the procedure. For example:

```
CREATE TOPIC USING STREAM sessions
   PROPERTIES (consumer.keys=user_id,
     consumer.values='login_id,login_time,ip_address',
     producer.parameters.includeKey=true
   );
```

Configuring Topic Profiles

The next step is to configure the topics in the database configuration file. First, add the `<topics>` element to the database configuration:

```
<deployment>

    [ . . . ]

  <topics enabled="true">
  </topics>
</deployment>
```

The `enabled="true"` attribute is optional since, by default, topics are enabled whenever the configuration includes a <topic> element. Enabling the topics ensures that the topic port (9092 by default) is open and both consumers and producers can access the topics defined in the database. If the <topics> element is *not* included, topics are disabled, and the port is closed and the subscription service is not accessible.

You can use the `enabled` attribute to explicitly enable or disable the use of topics. Note that even if topics are disabled, the queues associated with topics created for output (that is, with the USING STREAM clause) still exist and any data inserted into the stream will become available to consumers as soon as topics are re-enabled by a configuration update.

Within the <topics> element you can define profiles and associated retention policies which specify when to delete data based either on time or size. For a time-based retention policy, you specify how long the data is kept (in hours, days, weeks, etc) before it is deleted. For a size-based retention policy, you specify a maximum amount of data that will be kept before the oldest data is deleted.

It is a good idea to assign profiles for every topic so you can control how long data is retained. Multiple topics can be assigned the same profile, if you want to manage their retention policy as a group. Or you can assign topics to separate profiles if you want more refined control.

The profile itself and its retention policy are defined within the <topics> and <profiles> elements. For example, to define the *daily* profile used in the previous examples, the configuration file might look like this:

```
<deployment>

  [ . . . ]

  <topics enabled="true">
    <profiles>
```

```
      <profile name="daily">
        <retention policy="time" value="1dy"/>
      </profile>
    </profiles>
  </topics>

</deployment>
```

In this example, the policy type is set to *time* and the value is set to one day ("1dy"). The allowable policy types are *time* and *size*. The allowable units for each type of policy are listed in the following table.

Time-Based Policy	Size-Based Policy
• mn — minutes • hr — hours • dy — days • wk — weeks • mo — months • yr — years	• mb — megabytes • gb — gigabytes
Minimum value = 1 minute	Minimum value = 64 megabytes

Warning

Be sure if you specify a profile in the CREATE TOPIC statement to also define that profile and its retention policy in the configuration file. There is a default policy for topics that are not assigned to profiles. By default, data for a profile-less topic is kept for seven days. However, if you do assign a profile but do *not* define it, the topic has no retention policy and data is kept indefinitely, which can rapidly fill up and exhaust available disk space.

Configuring the Subscription Service on the Database Server

In addition to defining the profiles and retention policies, the configuration file lets you control other aspects of the operation and performance of the server(s). You can specify basic functions, such as which port number external clients use to subscribe and publish to the topics. By default the subscriber port is 9092. You can specify an different port as a property in the configuration file using the <properties> subelement of topics. For example, the following example sets the port number to 9999.

```
<topics enabled="true">
  <properties>
    <property name="port">9999</property>
  </properties>
  <profiles>
    <profile name="daily">
      <retention policy="time" value="1dy"/>
    </profile>
  </profiles>
</topics>
```

Alternately you can set the port number on the **voltdb start** command using the `--topicsport` argument. If both are specified, the command line argument takes precedence.

You can also specify a thread pool for processing the topics. By default, VoltDB uses the default export thread pool to process outbound topics. If you have both export queues and topics, you may want to separate the thread pools. Or if you have multiple consumers accessing topics simultaneously, you may want to increase the number of threads used so client requests are processed concurrently. You alter the pool size by defining a named thread pool then assigning that pool name as a topics attribute. For example, the following configuration uses ten threads to process topics:

```
<threadpools>
  <pool name="topicthreads" size="10"/>
<threadpools>

<topics enabled="true" threadpool="topicthreads">
      [ . . . ]
</topics>
```

If you are using AVRO format for any of your topics, you must also specify the URL of an AVRO schema registry where VoltDB can store the schema for each AVRO topic. You specify the registry in the <avro> element, separate from the <topic> element. You can optionally specify a name space for your schema and a prefix for the topic names as well. For example, if your registry is http://avro.local.lan/ and you want to use the name space mydb.voltdb, the VoltDB configuration file might look like the following:

```
<deployment>

  [ . . . ]

  <avro registry="http://avro.local.lan/" namespace="mydb.voltdb"/>
  <topics enabled="true">
      [ . . . ]
  </topics>
</deployment>
```

Managing the Topic Queues

Managing topics at runtime is a process of balancing the tradeoffs between extended availability (how long data is available to clients) and server resources (how much disk space is consumed by the topic queues). Whether you define the retention policy by time or size, it is a good idea to monitor both aspects of the topic to ensure effective usage.

You can see how much space is being consumed and how long data is being retained using the @Statistics system procedure with the TOPIC selector. The statistics for topics includes columns for the offset and timestamp of the first and last row of data in the queue for each topic in each partition. It also includes the total number of bytes for each as well.

By monitoring the timestamp of the first row you can determine, in general, how long data stays in the queue if you are using a size-based retention plan. Otherwise, you can use the sum of the total byte counts for a topic to determine how much space is consumed by a topic with a time-based retention policy.

By increasing or decreasing the size limit for a sized-based policy you directly control the space consumed, but you also affect how long the data is kept. Similarly, by increasing or decreasing the time limit for a time-based policy you can manage how much space the queue takes up on disk. Adjustments can be made by updating the definition of the topic's profile in the database configuration. For example, if the seven day weekly retention policy is taking up too much space, it can be adjusted to five days with the following definition in the configuration file and applying it to the running database using the **voltadmin update** command:

```
<profiles>
```

```
  <profile name="weekly">
    <retention policy="time" value="5dy"/>
  </profile>
</profiles>
```

Known Limitations for the Beta Preview

The following are the known limitations to the preview of VoltDB topics.

1.1. VoltDB uses a fixed 2MB buffer for topics

Kafka clients may specify a minimum and/or maximum buffer size for fetching topic data. However, VoltDB uses a fixed 2MB buffer, even if the client specifies a lower maximum buffer size. Clients should be aware of this limitation and be prepared to handle the larger data load.

1.2. Using INSERT INTO SELECT to write data into a topic stream does not always succeed

Using INSERT INTO SELECT to insert data into a topic stream can produce unexpected results. The data is not always inserted into the queue in the correct order.

1.3. VoltDB 10.1.2 Enhancements

Support for JSON format topics and keys was added in VoltDB release V10.1.2 and is not available in the earlier 10.1.1.

Reference Documents

The following is a working draft of the reference documentation for the CREATE TOPIC statement.

CREATE TOPIC

CREATE TOPIC — Creates a topic and corresponding flow control for the database.

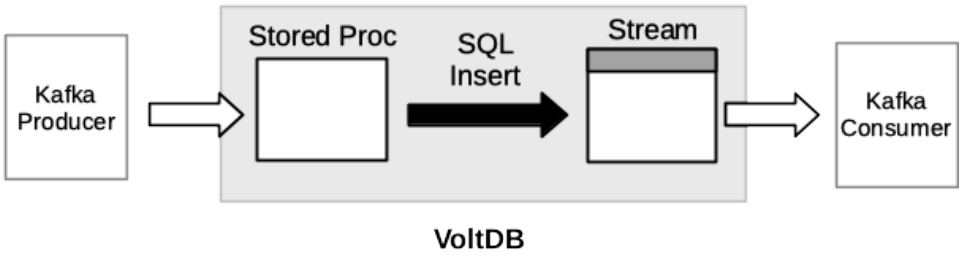
Syntax

```
CREATE TOPIC [USING STREAM] topic-name
        [EXECUTE PROCEDURE procedure-name]
        [ALLOW role-name ]
        [PROFILE profile-name ]
        [PROPERTIES ( property-name=value [,...] )]

CREATE OPAQUE TOPIC topic-name [PARTITIONED]
        [ALLOW role-name ]
        [PROFILE profile-name ]
```

Description

The CREATE TOPIC statement defines a *topic*, a pipeline for streaming data through VoltDB. Topics use the Apache Kafka protocols for producing (input) and consuming (output) the data. The EXECUTE PROCEDURE clause specifies the stored procedure that receives the inbound data and the USING STREAM clause specifies that a stream — with the same name as the topic — is used to queue the outbound data. VoltDB topics operate just like Kafka topics, with the database nodes acting as Kafka brokers. However, unlike Kafka, VoltDB topics also have the ability to analyze, act on, or even modify the data as it passes through.



As the preceding diagram shows, data submitted to the topic from a Kafka producer (either using the Kafka API or using a tool such as Kafka Connect) is passed to the stored procedure, which then interprets and operates on the data before passing it along to the stream through standard VoltDB INSERT semantics. Note that the named procedure and stream must exist prior to declaring the topic. In other words, you must declare the stored procedure and/or stream before issuing a CREATE TOPIC statement with the USING STREAM and EXECUTE PROCEDURE clauses. For example, the following statements declare the necessary stored procedure and stream before creating the topic *eventlog* that uses them:

```
CREATE STREAM eventlog
  PARTITION ON COLUMN e_time
  ( e_time TIMESTAMP NOT NULL,
    e_type INTEGER NOT NULL,
    e_msg  VARCHAR(256)
  );
CREATE PROCEDURE
  PARTITION ON TABLE eventlog COLUMN e_time
    FROM CLASS mycompany.myprocs.processEvent;
```

```
CREATE TOPIC USING STREAM eventLog
  EXECUTE PROCEDURE processEvent;
```

Case Sensitivity

The names of Kafka topics are case-sensitive, and so are VoltDB topics. That means that the name of the topic, like the names of stored procedures, matches exactly how you enter it in the CREATE TOPIC statement. So in the previous example, all lowercase except for the letter "L". However, streams, like tables, are case insensitive. So the topic "eventLog" is matched to any stream with the same spelling, regardless of case.

In other words, any references to the topic, such as external access from Kafka producers and consumers, must spell the name exactly matching the case in the CREATE TOPIC statement. But internal references to the output stream in INSERT statements are case insensitive.

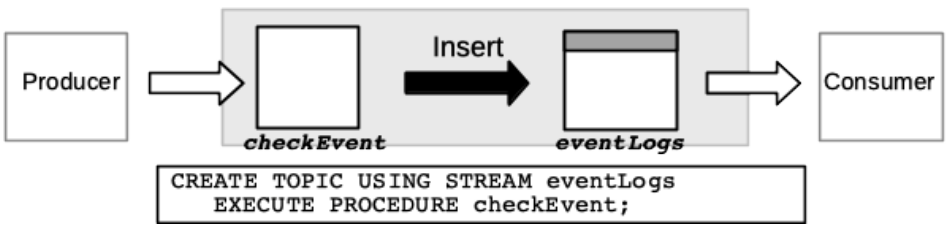
Clauses to the command let you manage the queue associated with the topic, as well as its structure and format. The clauses are summarised here and explained in more detail in the following sections.

ALLOW clause	Specifies user roles that can read from (consume) the topic from outside VoltDB. The ALLOW clause uses the same semantics as the ALLOW clause for stored procedures, in that only users with the specified role(s) are allowed to access the topic for read access. Note that the permission to *write* to the topic from an external topic producer is controlled by the ALLOW cause on the stored procedure named in the EXECUTE PROCEDURE clause, not by the ALLOW clause on the topic itself.
PROFILE clause	Associates the topic with a profile, defined in the database configuration file, that defines the retention policy and other attributes of the output queue.
PROPERTIES clause	Specifies properties defining the structure and format of the topic. The underlying structure of the data within VoltDB is defined by the stream identified in the USING STREAM clause. However, the structure and format of the actual topic message sent as output by VoltDB, and any keys associated with the message, are defined through properties in the PROPERTIES clause. Properties also let you define how incoming messages sent by Kafka producers are interpreted.

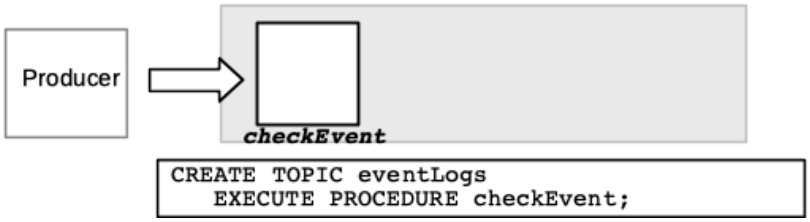
Full, Partial, and Opaque Topics

VoltDB does, in fact, support four different types of topics, depending on whether the topic is a full, partial, or opaque topic:

- A **fully processed topic** as described above, is a pipeline that supports both input and output and passes through a stored procedure. This is defined using both the USING STREAM and EXECUTE PROCEDURE clauses.

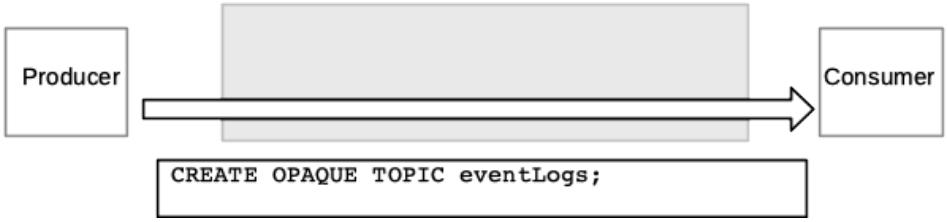


- An **input-only topic** only provides for input from Kafka producers. You define an input-only topic by including the EXECUTE PROCEDURE clause, without the USING STREAM clause.



- An **output-only topic** only provides for output to Kafka consumers but can be written to by VoltDB INSERT statements. You define an output-only topic by including the USING STREAM clause, without the EXECUTE PROCEDURE clause.

- An **opaque topic** supports input and output but provides for no processing or interpretation. You define an opaque topic using the CREATE OPAQUE TOPIC statement, as described in the section called "Using Opaque Topics".



Defining the Topic Structure and Format (Properties)

By default, the structure of the stream named in the USING STREAM clause defines the structure of the topic message sent to consumers. That is, the message fields match the stream columns in type and order. However, VoltDB cannot enforce the structure or the format of the messages that producers send as input. As a result, for input VoltDB interprets the structure from the actual message itself, based on the data format you specify (Avro, CSV, or JSON), then submits the fields as arguments to the stored procedure call.

In other words, the structure and formatting of messages are handled differently for input versus output. Each can be configured in detail, either separately or together, through the PROPERTIES clause. There are three main aspects of the topic that can be configured:

- The structure of the message (the number, order, and datatype of the fields)
- The key or keys associated with the message
- The format of the message and/or its keys (CSV or AVRO)

By default, the data for output is formatted either in the native Kafka binary format for single values or as comma-separated values (CSV) for multiple fields. Similarly, messages received from producers are assumed to be in CSV format.

For output, the default is that all of the columns of the stream, in the order specified, make up the topic message and there is no key. For input, the message is evaluated (based on the specified format) to identify the individual fields, and if there is a key, it is assumed to be a single field in the Kafka binary format.

You can change the default structure and format of the messages for either or both input and output. For example, you can specify which columns to include in the message for output and which to use as the key, by specifying property names and values as a comma-separated list in the PROPERTIES clause. The following topic declaration specifies that the format is CSV, the message includes only three columns from the stream, and a fourth column (not included in the message) is the key:

```
CREATE TOPIC USING STREAM logins EXECUTE PROCEDURE trackUsers
  PROPERTIES(topic.format=csv,
    consumer.values='login_time,ipaddr,location',
    consumer.keys=userid);
```

Table 1, “Properties of the CREATE TOPIC Statement” describes all of the valid properties for the PROPERTIES clause, including their default values. The property names are listed in mixed case for readability. However, the property names are *not* case sensitive.

Table 1. Properties of the CREATE TOPIC Statement

Format Property	Default	Description
consumer.format={AVRO|CSV|JSON}	CSV	Format of the topic message and keys sent to consumers.
consumer.format.values={AVRO|CSV|JSON}	CSV	Format of the topic message sent to consumers.
consumer.format.keys={AVRO|CSV|JSON}	CSV	Format of the topic keys sent to consumers.
producer.format={AVRO|CSV|JSON}	CSV	Format of the topic message received from producers.
producer.format.values={AVRO|CSV|JSON}	CSV	Format of the topic message received from producers. (Synonym for producer.format.)
topic.format={AVRO|CSV|JSON}	CSV	Format of the topic message and keys for both consumers and producers.
topic.format.values={AVRO|CSV|JSON}	CSV	Format of the topic message for both consumers and producers.
topic.format.keys={AVRO|CSV|JSON}	CSV	Format of the topic keys sent to consumers.
Structure Property	**Default**	**Description**
consumer.keys=*column* [,...]	no key	The stream columns to include in the topic key.
consumer.values= *column* [,...]	all columns	The stream columns to be inserted as values into the topic message.
producer.parameters.includeKey=true|false	false	Whether the topic key is included as the partitioning parameter to the stored procedure call.
Format Configuration Property	**Default**	**Description**
config.avro.timestamp={*unit*}	MICRO SECONDS	The unit of measure for timestamps in AVRO formatted fields: MICROSECONDS or MILLISECONDS.
config.avro.geographyPoint={*datatype*}	FIXED_ BINARY	The datatype for GEOGRAPHY_POINT columns in AVRO formatted fields: BINARY, FIXED_BINARY, or STRING.
config.avro.geography={*datatype*}	BINARY	The datatype for GEOGRAPHY columns in AVRO formatted fields: BINARY or STRING.
config.csv.escape={*character*}	Backslash (\)	The character used to escape the next character in a quoted string in CSV format.

Format Configuration Property	Default	Description
config.csv.null={*string*}	Backslash-N (\N)	The character(s) representing a null value in CSV format.
config.csv.quote={*character*}	Double quote (")	The character used to enclose quoted strings in CSV format.
config.csv.separator={*character*}	Comma (,)	The character separating the value fields of a message in CSV format.
config.csv.ignoreLeadingWhitespace={true|false}	true	Whether leading spaces are included in string values in CSV format.
config.csv.alwaysQuote={true|false}	false	Whether all string values are quotes or only strings with special characters (such as commas, line breaks, and quotation marks) in CSV format.
config.json.schema={EMBEDDED|NONE}	NONE	Whether the JSON representation contains an property named "schema" embedded within it or not. If embedded, the schema property describes the layout of the object.

Defining the Retention Policy (PROFILE)

Unlike export, where the connector *pushes* each record to a single "target", topics are queued where one or more consumers *pull* the records as needed. In export, records are deleted from the queue as soon as the target acknowledges receipt. With topics there is no specific event that determines that a record is no longer needed, so you must define a retention policy where records expire and are deleted based on some other triggering event.

You specify the retention policy, based either on the age of the records or the size of the queue, in the configuration file as part of a topic *profile*. You then associate a topic with a specific profile using the PROFILE clause. For example, the following topic declaration associates the *logins* topic with the *weekly* profile:

```
CREATE TOPIC USING STREAM logins EXECUTE PROCEDURE trackUsers
  PROFILE weekly;
```

It is a good idea to assign profiles for every topic so you can control how long data is retained. Multiple topics can be assigned the same profile, if you want to manage their retention policy as a group. Or you can assign topics to separate profiles if you need more refined control.

The profile itself and its retention policy are defined within the <topics> and <profiles> elements. For example, to define both a *daily* and a *weekly* profile, the configuration file might look something like this:

```
<topics enabled="true">
  <profiles>
    <profile name="daily"
      <retention policy="time" value="1dy"/>
    </profile>
    <profile name="weekly">
      <retention policy="time" value="1wk"/>
    </profile>
  </profiles>
</topics>
```

In this example, the policy type is set to *time* and the value is set to either one day ("1dy") or one week ("1wk"). The allowable policy types are *time* and *size*. The allowable units for each type of policy are listed in the following table.

Time-Based Policy	Size-Based Policy
• mn — minutes • hr — hours • dy — days • wk — weeks • mo — months • yr — years	• mb — megabytes • gb — gigabytes
Minimum value = 1 minute	Minimum value = 64 megabytes

Warning

Be sure if you specify a profile in the CREATE TOPICS USING STREAM statement to also define that profile and its retention policy in the configuration file. There is a default policy for topics that are not assigned to profiles. By default, data for a topic without a profile is kept for seven days. However, if you do assign a profile but do *not* define it, the topic has no retention policy and data is kept indefinitely, which can rapidly fill up and exhaust available disk space.

Receiving Topic Records From Producers

When the CREATE TOPIC statement includes the EXECUTE PROCEDURE clause, the topic is available for input from Kafka producers. Records sent to the topic are decomposed to their component fields (based on the specified data format) and then passed as arguments to the named stored procedure. If there is a key for the topic, the key value is added as the partitioning parameter.

Note that you cannot specify a data format for the key key associated with records received from producers, whereas you can define a separate format for keys sent to consumers. This is because, although Kafka allows compound keys containing multiple values, VoltDB constrains topics received as input to a single key value. This key value is, by default, not included in the parameters to the stored procedure call, assuming that the key value may very well already be included as part of the topic record. However, you can explicitly ask to have the key included as the first parameter by setting the property `producer.parameters.includeKey` to true.

Using Opaque Topics

Not all data needs analysis. But maintaining separate infrastructure for intelligent processing and unprocessed topics is expensive. That is why VoltDB provides a mechanism for handling unprocessed, or opaque, topics as well as processed topics, so you can aggregate all your streaming needs into one set of servers.

You create opaque topics using the CREATE OPAQUE TOPIC statement. The opaque topic allows for both input from producers and output to consumers without explicitly declaring an output stream — VoltDB automatically creates the necessary output queue based on the name you provide. At run time, opaque topics pass all records received from producers directly to the output topic queue for consumers.

There are no properties associated with opaque topics, since there is no processing and therefore no settings to tweak. However, you can and should assign a retention policy for them by assigning a profile. For example:

```
CREATE OPAQUE TOPIC system_alerts
    PROFILE daily;
```

Examples

The following example demonstrates a decision-making process using VoltDB topics. The example declares two streams, one stored procedure, and two topics. The first topic, *auction_bids*, provides for both input and output. The second topic, *rejected_bids*, is output only since there is no EXECUTE PROCEDURE clause. The stored procedure, *EvaluateBids*, examines the incoming bids on the *auction_bids* topic and determines whether to pass them through by inserting them into the *auction_bids* stream, or to divert them by inserting them into the *rejected_bids* stream. Note that both topics use the same profile and therefore the same retention policy.

```
CREATE STREAM auction_bids
  PARTITION ON COLUMN bid_id
    (bid_id INTEGER NOT NULL, user_id INTEGER, bid DECIMAL);
CREATE STREAM rejected_bids
  PARTITION ON COLUMN bid_id
    (bid_id INTEGER NOT NULL, user_id INTEGER, bid DECIMAL);
CREATE PROCEDURE
  PARTITION ON auction_bids COLUMN bid_id
    FROM CLASS auctionprocs.EvaluateBids;
CREATE TOPIC USING STREAM auction_bids
  EXECUTE PROCEDURE EvaluateBids
    PROFILE bids;
CREATE TOPIC USING STREAM rejected_bids
    PROFILE bids;
```

The next example declares two topics — one for processing user session logins and another for handling expired sessions. New sessions need to be validated, but expired sessions can simply be passed through the consumers as is. This demonstrates how opaque topics allow VoltDB to manage all of your streaming needs, add intelligence to the pipeline, while reducing the infrastructure needed to support your business goals.

```
CREATE TOPIC USING STREAM new_sessions
  EXECUTE PROCEDURE ValidateUser
    PROFILE sessions;
CREATE OPAQUE TOPIC expired_sessions
    PROFILE sessions;
```

ALTER TOPIC

ALTER TOPIC — Modifies an existing topic definition.

Syntax

ALTER TOPIC *topic-name* ADD PROPERTIES (*property-name=value* [,...])

ALTER TOPIC *topic-name* ALTER PROPERTIES (*property-name=value* [,...])

ALTER TOPIC *topic-name* DROP PROPERTIES (*property-name* [,...])

Description

The ALTER TOPIC statement modifies the set of properties associated with the specified topic. There are three variants of the ALTER TOPIC statement: ALTER TOPIC... ADD, ALTER TOPIC... DROP, and ALTER TOPIC... ALTER, In each case the specified properties are added, dropped, or altered. Other existing properties of the topic remain unchanged. So, for example, if the CREATE TOPIC statement defined values for three properties *a*, *b*, and *c*, then executing the statement `ALTER TOPIC topic-name ADD PROPERTIES (d=value)` would result in the topic having four explicit properties: a, b, c, and d. Similarly, if you then execute the statement `ALTER TOPIC topic-name DROP PROPERTIES (a, b)` there will be two explicit properties remaining: *c* and *d*.

If a property is dropped and there is a default value for the property, the default value is used for all subsequent topic records until a subsequent ALTER TOPIC statement changes it.

Example

The following example adds two properties, specifying the column *sku_num* as the key for the topic and AVRO as the data format.

```
ALTER TOPIC products ADD PROPERTIES (consumer.keys=sku_num, topic.format=avro);
```

The second example drops the topic.format property added previously, causing the format to revert to the default, CSV.

```
ALTER TOPIC products DROP PROPERTIES (topic.format);
```