//@file

Global configuration properties

Property	C/P	Range	Default	Description
builtin.features	*		gzip, snappy, ssl, sasl, regex, lz4, sasl_gssapi, sasl_plain, sasl_scram, plugins, zstd	Indicates the builtin features for this build of librdkafka. An application can either query this value or attempt to set it with its list of required features to check for library support. *Type: CSV flags*
client.id	*		rdkafka	Client identifier. *Type: string*
metadata.broker.list	*			Initial list of brokers as a CSV list of broker host or host:port. The application may also use `rd_kafka_brokers_add()` to add brokers during runtime. *Type: string*
bootstrap.servers	*			Alias for `metadata.broker.list`
message.max.bytes	*	1000 .. 1000000000	1000000	Maximum Kafka protocol request message size. *Type: integer*
message.copy.max.bytes	*	0 .. 1000000000	65535	Maximum size for message to be copied to buffer. Messages larger than this will be passed by reference (zero-copy) at the expense of larger iovecs. *Type: integer*
receive.message.max.bytes	*	1000 .. 2147483647	100000000	Maximum Kafka protocol response message size. This serves as a safety precaution to avoid memory exhaustion in case of protocol hickups. This value must be at least `fetch.max.bytes` + 512 to allow for protocol overhead; the value is adjusted automatically unless the configuration property is explicitly set. *Type: integer*
max.in.flight.requests.per.connection	*	1 .. 1000000	1000000	Maximum number of in-flight requests per broker connection. This is a generic property applied to all broker communication, however it is primarily relevant to produce requests. In particular, note that other mechanisms limit the number of outstanding consumer fetch request per broker to one. *Type: integer*
max.in.flight	*			Alias for `max.in.flight.requests.per.connection`
metadata.request.timeout.ms	*	10 .. 900000	60000	Non-topic request timeout in milliseconds. This is for metadata requests, etc. *Type: integer*
topic.metadata.refresh.interval.ms	*	-1 .. 3600000	300000	Topic metadata refresh interval in milliseconds. The metadata is automatically refreshed on error and connect. Use -1 to disable the intervalled refresh. *Type: integer*
metadata.max.age.ms	*	1 .. 86400000	900000	Metadata cache max age. Defaults to topic.metadata.refresh.interval.ms * 3 *Type: integer*
topic.metadata.refresh.fast.interval.ms	*	1 .. 60000	250	When a topic loses its leader a new metadata request will be enqueued with this initial interval, exponentially increasing until the topic metadata has been refreshed. This is used to recover quickly from transitioning leader brokers. *Type: integer*
topic.metadata.refresh.fast.cnt	*	0 .. 1000	10	**DEPRECATED** No longer used. *Type: integer*
topic.metadata.refresh.sparse	*	true, false	true	Sparse metadata requests (consumes less network bandwidth) *Type: boolean*

Property	C/P	Range	Default	Description
topic.blacklist	*			Topic blacklist, a comma-separated list of regular expressions for matching topic names that should be ignored in broker metadata information as if the topics did not exist. *Type: pattern list*
debug	*	generic, broker, topic, metadata, feature, queue, msg, protocol, cgrp, security, fetch, interceptor, plugin, consumer, admin, eos, all		A comma-separated list of debug contexts to enable. Detailed Producer debugging: broker,topic,msg. Consumer: consumer,cgrp,topic,fetch *Type: CSV flags*
socket.timeout.ms	*	10 .. 300000	60000	Default timeout for network requests. Producer: ProduceRequests will use the lesser value of `socket.timeout.ms` and remaining `message.timeout.ms` for the first message in the batch. Consumer: FetchRequests will use `fetch.wait.max.ms` + `socket.timeout.ms`. Admin: Admin requests will use `socket.timeout.ms` or explicitly set `rd_kafka_AdminOptions_set_operation_timeout()` value. *Type: integer*
socket.blocking.max.ms	*	1 .. 60000	1000	**DEPRECATED** No longer used. *Type: integer*
socket.send.buffer.bytes	*	0 .. 100000000	0	Broker socket send buffer size. System default is used if 0. *Type: integer*
socket.receive.buffer.bytes	*	0 .. 100000000	0	Broker socket receive buffer size. System default is used if 0. *Type: integer*
socket.keepalive.enable	*	true, false	false	Enable TCP keep-alives (SO_KEEPALIVE) on broker sockets *Type: boolean*
socket.nagle.disable	*	true, false	false	Disable the Nagle algorithm (TCP_NODELAY) on broker sockets. *Type: boolean*
socket.max.fails	*	0 .. 1000000	1	Disconnect from broker when this number of send failures (e.g., timed out requests) is reached. Disable with 0. WARNING: It is highly recommended to leave this setting at its default value of 1 to avoid the client and broker to become desynchronized in case of request timeouts. NOTE: The connection is automatically re-established. *Type: integer*
broker.address.ttl	*	0 .. 86400000	1000	How long to cache the broker address resolving results (milliseconds). *Type: integer*
broker.address.family	*	any, v4, v6	any	Allowed broker IP address families: any, v4, v6 *Type: enum value*
enable.sparse.connections	*	true, false	true	When enabled the client will only connect to brokers it needs to communicate with. When disabled the client will maintain connections to all brokers in the cluster. *Type: boolean*
reconnect.backoff.jitter.ms	*	0 .. 3600000	0	**DEPRECATED** No longer used. See `reconnect.backoff.ms` and `reconnect.backoff.max.ms`. *Type: integer*

Property	C/P	Range	Default	Description
reconnect.backoff.ms	*	0 .. 3600000	100	The initial time to wait before reconnecting to a broker after the connection has been closed. The time is increased exponentially until `reconnect.backoff.max.ms` is reached. -25% to +50% jitter is applied to each reconnect backoff. A value of 0 disables the backoff and reconnects immediately. *Type: integer*
reconnect.backoff.max.ms	*	0 .. 3600000	10000	The maximum time to wait before reconnecting to a broker after the connection has been closed. *Type: integer*
statistics.interval.ms	*	0 .. 86400000	0	librdkafka statistics emit interval. The application also needs to register a stats callback using `rd_kafka_conf_set_stats_cb()`. The granularity is 1000ms. A value of 0 disables statistics. *Type: integer*
enabled_events	*	0 .. 2147483647	0	See `rd_kafka_conf_set_events()` *Type: integer*
error_cb	*			Error callback (set with rd_kafka_conf_set_error_cb()) *Type: pointer*
throttle_cb	*			Throttle callback (set with rd_kafka_conf_set_throttle_cb()) *Type: pointer*
stats_cb	*			Statistics callback (set with rd_kafka_conf_set_stats_cb()) *Type: pointer*
log_cb	*			Log callback (set with rd_kafka_conf_set_log_cb()) *Type: pointer*
log_level	*	0 .. 7	6	Logging level (syslog(3) levels) *Type: integer*
log.queue	*	true, false	false	Disable spontaneous log_cb from internal librdkafka threads, instead enqueue log messages on queue set with `rd_kafka_set_log_queue()` and serve log callbacks or events through the standard poll APIs. **NOTE**: Log messages will linger in a temporary queue until the log queue has been set. *Type: boolean*
log.thread.name	*	true, false	true	Print internal thread name in log messages (useful for debugging librdkafka internals) *Type: boolean*
log.connection.close	*	true, false	true	Log broker disconnects. It might be useful to turn this off when interacting with 0.9 brokers with an aggressive `connection.max.idle.ms` value. *Type: boolean*
background_event_cb	*			Background queue event callback (set with rd_kafka_conf_set_background_event_cb()) *Type: pointer*
socket_cb	*			Socket creation callback to provide race-free CLOEXEC *Type: pointer*
connect_cb	*			Socket connect callback *Type: pointer*
closesocket_cb	*			Socket close callback *Type: pointer*
open_cb	*			File open callback to provide race-free CLOEXEC *Type: pointer*
opaque	*			Application opaque (set with rd_kafka_conf_set_opaque()) *Type: pointer*
default_topic_conf	*			Default topic configuration for automatically subscribed topics *Type: pointer*

Property	C/P	Range	Default	Description
internal.termination.signal	*	0 .. 128	0	Signal that librdkafka will use to quickly terminate on rd_kafka_destroy(). If this signal is not set then there will be a delay before rd_kafka_wait_destroyed() returns true as internal threads are timing out their system calls. If this signal is set however the delay will be minimal. The application should mask this signal as an internal signal handler is installed. *Type: integer*
api.version.request	*	true, false	true	Request broker's supported API versions to adjust functionality to available protocol features. If set to false, or the ApiVersionRequest fails, the fallback version `broker.version.fallback` will be used. **NOTE**: Depends on broker version >=0.10.0. If the request is not supported by (an older) broker the `broker.version.fallback` fallback is used. *Type: boolean*
api.version.request.timeout.ms	*	1 .. 300000	10000	Timeout for broker API version requests. *Type: integer*
api.version.fallback.ms	*	0 .. 604800000	1200000	Dictates how long the `broker.version.fallback` fallback is used in the case the ApiVersionRequest fails. **NOTE**: The ApiVersionRequest is only issued when a new connection to the broker is made (such as after an upgrade). *Type: integer*
broker.version.fallback	*		0.9.0	Older broker versions (before 0.10.0) provide no way for a client to query for supported protocol features (ApiVersionRequest, see `api.version.request`) making it impossible for the client to know what features it may use. As a workaround a user may set this property to the expected broker version and the client will automatically adjust its feature set accordingly if the ApiVersionRequest fails (or is disabled). The fallback broker version will be used for `api.version.fallback.ms`. Valid values are: 0.9.0, 0.8.2, 0.8.1, 0.8.0. Any other value, such as 0.10.2.1, enables ApiVersionRequests. *Type: string*
security.protocol	*	plaintext, ssl, sasl_plaintext, sasl_ssl	plaintext	Protocol used to communicate with brokers. *Type: enum value*
ssl.cipher.suites	*			A cipher suite is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. See manual page for `ciphers(1)` and `SSL_CTX_set_cipher_list(3). *Type: string*
ssl.curves.list	*			The supported-curves extension in the TLS ClientHello message specifies the curves (standard/named, or 'explicit' GF(2^k) or GF(p)) the client is willing to have the server use. See manual page for `SSL_CTX_set1_curves_list(3)`. OpenSSL >= 1.0.2 required. *Type: string*
ssl.sigalgs.list	*			The client uses the TLS ClientHello signature_algorithms extension to indicate to the server which signature/hash algorithm pairs may be used in digital signatures. See manual page for `SSL_CTX_set1_sigalgs_list(3)`. OpenSSL >= 1.0.2 required. *Type: string*
ssl.key.location	*			Path to client's private key (PEM) used for authentication. *Type: string*
ssl.key.password	*			Private key passphrase *Type: string*

Property	C/P	Range	Default	Description
ssl.certificate.location	*			Path to client's public key (PEM) used for authentication. *Type: string*
ssl.ca.location	*			File or directory path to CA certificate(s) for verifying the broker's key. *Type: string*
ssl.crl.location	*			Path to CRL for verifying broker's certificate validity. *Type: string*
ssl.keystore.location	*			Path to client's keystore (PKCS#12) used for authentication. *Type: string*
ssl.keystore.password	*			Client's keystore (PKCS#12) password. *Type: string*
sasl.mechanisms	*		GSSAPI	SASL mechanism to use for authentication. Supported: GSSAPI, PLAIN, SCRAM-SHA-256, SCRAM-SHA-512. **NOTE**: Despite the name only one mechanism must be configured. *Type: string*
sasl.mechanism	*			Alias for `sasl.mechanisms`
sasl.kerberos.service.name	*		kafka	Kerberos principal name that Kafka runs as, not including /hostname@REALM *Type: string*
sasl.kerberos.principal	*		kafkaclient	This client's Kerberos principal name. (Not supported on Windows, will use the logon user's principal). *Type: string*
sasl.kerberos.kinit.cmd	*		kinit -S "%{sasl.kerberos.service.name}/%{broker.name}" -k -t "%{sasl.kerberos.keytab}" %{sasl.kerberos.principal}	Full kerberos kinit command string, %{config.prop.name} is replaced by corresponding config object value, %{broker.name} returns the broker's hostname. *Type: string*
sasl.kerberos.keytab	*			Path to Kerberos keytab file. Uses system default if not set.**NOTE**: This is not automatically used but must be added to the template in sasl.kerberos.kinit.cmd as `... -t %{sasl.kerberos.keytab}`. *Type: string*
sasl.kerberos.min.time.before.relogin	*	1 .. 86400000	60000	Minimum time in milliseconds between key refresh attempts. *Type: integer*
sasl.username	*			SASL username for use with the PLAIN and SASL-SCRAM-.. mechanisms *Type: string*
sasl.password	*			SASL password for use with the PLAIN and SASL-SCRAM-.. mechanism *Type: string*
plugin.library.paths	*			List of plugin libraries to load (; separated). The library search path is platform dependent (see dlopen(3) for Unix and LoadLibrary() for Windows). If no filename extension is specified the platform-specific extension (such as .dll or .so) will be appended automatically. *Type: string*
interceptors	*			Interceptors added through rd_kafka_conf_interceptor_add_..() and any configuration handled by interceptors. *Type: *
group.id	C			Client group id string. All clients sharing the same group.id belong to the same group. *Type: string*
partition.assignment.strategy	C		range,roundrobin	Name of partition assignment strategy to use when elected group leader assigns partitions to group members. *Type: string*

Property	C/P	Range	Default	Description
session.timeout.ms	C	1 .. 3600000	10000	Client group session and failure detection timeout. The consumer sends periodic heartbeats (heartbeat.interval.ms) to indicate its liveness to the broker. If no hearts are received by the broker for a group member within the session timeout, the broker will remove the consumer from the group and trigger a rebalance. The allowed range is configured with the **broker** configuration properties `group.min.session.timeout.ms` and `group.max.session.timeout.ms`. *Type: integer*
heartbeat.interval.ms	C	1 .. 3600000	3000	Group session keepalive heartbeat interval. *Type: integer*
group.protocol.type	C		consumer	Group protocol type *Type: string*
coordinator.query.interval.ms	C	1 .. 3600000	600000	How often to query for the current client group coordinator. If the currently assigned coordinator is down the configured query interval will be divided by ten to more quickly recover in case of coordinator reassignment. *Type: integer*
max.poll.interval.ms	C	1 .. 86400000	300000	Maximum allowed time between calls to consume messages (e.g., rd_kafka_consumer_poll()) for high-level consumers. If this interval is exceeded the consumer is considered failed and the group will rebalance in order to reassign the partitions to another consumer group member. Warning: Offset commits may be not possible at this point. The interval is checked two times per second. See KIP-62 for more information. *Type: integer*
enable.auto.commit	C	true, false	true	Automatically and periodically commit offsets in the background. Note: setting this to false does not prevent the consumer from fetching previously committed start offsets. To circumvent this behaviour set specific start offsets per partition in the call to assign(). *Type: boolean*
auto.commit.interval.ms	C	0 .. 86400000	5000	The frequency in milliseconds that the consumer offsets are committed (written) to offset storage. (0 = disable). This setting is used by the high-level consumer. *Type: integer*
enable.auto.offset.store	C	true, false	true	Automatically store offset of last message provided to application. *Type: boolean*
queued.min.messages	C	1 .. 10000000	100000	Minimum number of messages per topic+partition librdkafka tries to maintain in the local consumer queue. *Type: integer*
queued.max.messages.kbytes	C	1 .. 2097151	1048576	Maximum number of kilobytes per topic+partition in the local consumer queue. This value may be overshot by fetch.message.max.bytes. This property has higher priority than queued.min.messages. *Type: integer*
fetch.wait.max.ms	C	0 .. 300000	100	Maximum time the broker may wait to fill the response with fetch.min.bytes. *Type: integer*
fetch.message.max.bytes	C	1 .. 1000000000	1048576	Initial maximum number of bytes per topic+partition to request when fetching messages from the broker. If the client encounters a message larger than this value it will gradually try to increase it until the entire message can be fetched. *Type: integer*
max.partition.fetch.bytes	C			Alias for `fetch.message.max.bytes`

Property	C/P	Range	Default	Description
fetch.max.bytes	C	0 .. 2147483135	52428800	Maximum amount of data the broker shall return for a Fetch request. Messages are fetched in batches by the consumer and if the first message batch in the first non-empty partition of the Fetch request is larger than this value, then the message batch will still be returned to ensure the consumer can make progress. The maximum message batch size accepted by the broker is defined via `message.max.bytes` (broker config) or `max.message.bytes` (broker topic config). `fetch.max.bytes` is automatically adjusted upwards to be at least `message.max.bytes` (consumer config). *Type: integer*
fetch.min.bytes	C	1 .. 100000000	1	Minimum number of bytes the broker responds with. If fetch.wait.max.ms expires the accumulated data will be sent to the client regardless of this setting. *Type: integer*
fetch.error.backoff.ms	C	0 .. 300000	500	How long to postpone the next fetch request for a topic+partition in case of a fetch error. *Type: integer*
offset.store.method	C	none, file, broker	broker	Offset commit store method: 'file' - local file store (offset.store.path, et.al), 'broker' - broker commit store (requires Apache Kafka 0.8.2 or later on the broker). *Type: enum value*
consume_cb	C			Message consume callback (set with rd_kafka_conf_set_consume_cb()) *Type: pointer*
rebalance_cb	C			Called after consumer group has been rebalanced (set with rd_kafka_conf_set_rebalance_cb()) *Type: pointer*
offset_commit_cb	C			Offset commit result propagation callback. (set with rd_kafka_conf_set_offset_commit_cb()) *Type: pointer*
enable.partition.eof	C	true, false	true	Emit RD_KAFKA_RESP_ERR__PARTITION_EOF event whenever the consumer reaches the end of a partition. *Type: boolean*
check.crcs	C	true, false	false	Verify CRC32 of consumed messages, ensuring no on-the-wire or on-disk corruption to the messages occurred. This check comes at slightly increased CPU usage. *Type: boolean*
enable.idempotence	P	true, false	false	When set to `true`, the producer will ensure that messages are successfully produced exactly once and in the original produce order. The following configuration properties are adjusted automatically (if not modified by the user) when idempotence is enabled: `max.inflight.requests.per.connection=5` (must be less than or equal to 5), `retries=INT32_MAX` (must be greater than 0), `acks=all`, `queuing.strategy=fifo`. Producer instantation will fail if user-supplied configuration is incompatible. *Type: boolean*
enable.gapless.guarantee	P	true, false	true	When set to `true`, any error that could result in a gap in the produced message series when a batch of messages fails, will raise a fatal error (ERR__GAPLESS_GUARANTEE) and stop the producer. Requires `enable.idempotence=true`. *Type: boolean*
queue.buffering.max.messages	P	1 .. 10000000	100000	Maximum number of messages allowed on the producer queue. *Type: integer*

Property	C/P	Range	Default	Description
queue.buffering.max.kbytes	P	1 .. 2097151	1048576	Maximum total message size sum allowed on the producer queue. This property has higher priority than queue.buffering.max.messages. *Type: integer*
queue.buffering.max.ms	P	0 .. 900000	0	Delay in milliseconds to wait for messages in the producer queue to accumulate before constructing message batches (MessageSets) to transmit to brokers. A higher value allows larger and more effective (less overhead, improved compression) batches of messages to accumulate at the expense of increased message delivery latency. *Type: integer*
linger.ms	P			Alias for `queue.buffering.max.ms`
message.send.max.retries	P	0 .. 10000000	2	How many times to retry sending a failing MessageSet. **Note:** retrying may cause reordering. *Type: integer*
retries	P			Alias for `message.send.max.retries`
retry.backoff.ms	P	1 .. 300000	100	The backoff time in milliseconds before retrying a protocol request. *Type: integer*
queue.buffering.backpressure.threshold	P	1 .. 1000000	1	The threshold of outstanding not yet transmitted broker requests needed to backpressure the producer's message accumulator. If the number of not yet transmitted requests equals or exceeds this number, produce request creation that would have otherwise been triggered (for example, in accordance with linger.ms) will be delayed. A lower number yields larger and more effective batches. A higher value can improve latency when using compression on slow machines. *Type: integer*
compression.codec	P	none, gzip, snappy, lz4, zstd	none	compression codec to use for compressing message sets. This is the default value for all topics, may be overridden by the topic configuration property `compression.codec`. *Type: enum value*
compression.type	P			Alias for `compression.codec`
batch.num.messages	P	1 .. 1000000	10000	Maximum number of messages batched in one MessageSet. The total MessageSet size is also limited by message.max.bytes. *Type: integer*
delivery.report.only.error	P	true, false	false	Only provide delivery reports for failed messages. *Type: boolean*
dr_cb	P			Delivery report callback (set with rd_kafka_conf_set_dr_cb()) *Type: pointer*
dr_msg_cb	P			Delivery report callback (set with rd_kafka_conf_set_dr_msg_cb()) *Type: pointer*

Topic configuration properties

Property	C/P	Range	Default	Description
request.required.acks	P	-1 .. 1000	1	This field indicates how many acknowledgements the leader broker must receive from ISR brokers before responding to the request: *0*=Broker does not send any response/ack to client, *1*=Only the leader broker will need to ack the message, *-1* or *all*=broker will block until message is committed by all in sync replicas (ISRs) or broker's `min.insync.replicas` setting before sending response. *Type: integer*
acks	P			Alias for `request.required.acks`

Property	C/P	Range	Default	Description
request.timeout.ms	P	1 .. 900000	5000	The ack timeout of the producer request in milliseconds. This value is only enforced by the broker and relies on `request.required.acks` being != 0. *Type: integer*
message.timeout.ms	P	0 .. 900000	300000	Local message timeout. This value is only enforced locally and limits the time a produced message waits for successful delivery. A time of 0 is infinite. This is the maximum time librdkafka may use to deliver a message (including retries). Delivery error occurs when either the retry count or the message timeout are exceeded. *Type: integer*
queuing.strategy	P	fifo, lifo	fifo	Producer queuing strategy. FIFO preserves produce ordering, while LIFO prioritizes new messages. WARNING: `lifo` is experimental and subject to change or removal. *Type: enum value*
produce.offset.report	P	true, false	false	**DEPRECATED** No longer used. *Type: boolean*
partitioner	P		consistent_random	Partitioner: `random` - random distribution, `consistent` - CRC32 hash of key (Empty and NULL keys are mapped to single partition), `consistent_random` - CRC32 hash of key (Empty and NULL keys are randomly partitioned), `murmur2` - Java Producer compatible Murmur2 hash of key (NULL keys are mapped to single partition), `murmur2_random` - Java Producer compatible Murmur2 hash of key (NULL keys are randomly partitioned. This is functionally equivalent to the default partitioner in the Java Producer.). *Type: string*
partitioner_cb	P			Custom partitioner callback (set with rd_kafka_topic_conf_set_partitioner_cb()) *Type: pointer*
msg_order_cmp	P			Message queue ordering comparator (set with rd_kafka_topic_conf_set_msg_order_cmp()). Also see `queuing.strategy`. *Type: pointer*
opaque	*			Application opaque (set with rd_kafka_topic_conf_set_opaque()) *Type: pointer*
compression.codec	P	none, gzip, snappy, lz4, zstd, inherit	inherit	Compression codec to use for compressing message sets. inherit = inherit global compression.codec configuration. *Type: enum value*
compression.type	P			Alias for `compression.codec`
compression.level	P	-1 .. 12	-1	Compression level parameter for algorithm selected by configuration property `compression.codec`. Higher values will result in better compression at the cost of more CPU usage. Usable range is algorithm-dependent: [0-9] for gzip; [0-12] for lz4; only 0 for snappy; -1 = codec-dependent default compression level. *Type: integer*
auto.commit.enable	C	true, false	true	[**LEGACY PROPERTY:** This property is used by the simple legacy consumer only. When using the high-level KafkaConsumer, the global `enable.auto.commit` property must be used instead]. If true, periodically commit offset of the last message handed to the application. This committed offset will be used when the process restarts to pick up where it left off. If false, the application will have to call `rd_kafka_offset_store()` to store an offset (optional). **NOTE:** There is currently no zookeeper integration, offsets will be written to broker or local file according to offset.store.method. *Type: boolean*
enable.auto.commit	C			Alias for `auto.commit.enable`
auto.commit.interval.ms	C	10 .. 86400000	60000	[**LEGACY PROPERTY:** This setting is used by the simple legacy consumer only. When using the high-level KafkaConsumer, the global `auto.commit.interval.ms` property must be used instead]. The frequency in milliseconds that the consumer offsets are committed (written) to offset storage. *Type: integer*

Property	C/P	Range	Default	Description
auto.offset.reset	C	smallest, earliest, beginning, largest, latest, end, error	largest	Action to take when there is no initial offset in offset store or the desired offset is out of range: 'smallest','earliest' - automatically reset the offset to the smallest offset, 'largest','latest' - automatically reset the offset to the largest offset, 'error' - trigger an error which is retrieved by consuming messages and checking 'message->err'. *Type: enum value*
offset.store.path	C		.	Path to local file for storing offsets. If the path is a directory a filename will be automatically generated in that directory based on the topic and partition. *Type: string*
offset.store.sync.interval.ms	C	-1 .. 86400000	-1	fsync() interval for the offset file, in milliseconds. Use -1 to disable syncing, and 0 for immediate sync after each write. *Type: integer*
offset.store.method	C	file, broker	broker	Offset commit store method: 'file' - local file store (offset.store.path, et.al), 'broker' - broker commit store (requires "group.id" to be configured and Apache Kafka 0.8.2 or later on the broker.). *Type: enum value*
consume.callback.max.messages	C	0 .. 1000000	0	Maximum number of messages to dispatch in one `rd_kafka_consume_callback*()` call (0 = unlimited) *Type: integer*

C/P legend: C = Consumer, P = Producer, * = both