

GstEvent

The event class provides factory methods to construct events for sending and functions to query (parse) received events.

Events are usually created with `gst_event_new_*`() which takes event-type specific parameters as arguments. To send an event application will usually use `gst_element_send_event` ([gststreamer/gstelement.html#gst_element_send_event](#)) and elements will use `gst_pad_send_event` ([gststreamer/gstpad.html#gst_pad_send_event](#)) or `gst_pad_push_event` ([gststreamer/gstpad.html#gst_pad_push_event](#)). The event should be unrefed with `gst_event_unref` ([gststreamer/gstevent.html#gst_event_unref](#)) if it has not been sent.

Events that have been received can be parsed with their respective `gst_event_parse_*`() functions. It is valid to pass NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>) for unwanted details.

Events are passed between elements in parallel to the data stream. Some events are serialized with buffers, others are not. Some events only travel downstream, others only upstream. Some events can travel both upstream and downstream.

The events are used to signal special conditions in the datastream such as EOS (end of stream) or the start of a new stream-segment. Events are also used to flush the pipeline of any pending data.

Most of the event API is used inside plugins. Applications usually only construct and use seek events. To do that `gst_event_new_seek` ([gststreamer/gstevent.html#gst_event_new_seek](#)) is used to create a seek event. It takes the needed parameters to specify seeking time and mode.

```
GstEvent *event;
gboolean result;
...
// construct a seek event to play the media from second 2 to 5, flush
// the pipeline to decrease latency.
event = gst_event_new_seek (1.0,
    GST_FORMAT_TIME,
    GST_SEEK_FLAG_FLUSH,
    GST_SEEK_TYPE_SET, 2 * GST_SECOND,
    GST_SEEK_TYPE_SET, 5 * GST_SECOND);
...
result = gst_element_send_event (pipeline, event);
if (!result)
    g_warning ("seek failed");
...
```

GstEvent

A GstEvent ([gststreamer/gstevent.html#GstEvent](#)).

Members

mini_object (GstMiniObject (gststreamer/gstminiobject.html#GstMiniObject)) – the parent structure

type (GstEventType (gststreamer/gstevent.html#GstEventType)) – the GstEventType (gststreamer/gstevent.html#GstEventType) of the event

timestamp (guint64 (https://docs.gtk.org/glib/types.html#guint64)) – the timestamp of the event

seqnum (guint32 (https://docs.gtk.org/glib/types.html#guint32)) – the sequence number of the event

Constructors

gst_event_new_buffer_size

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_buffer_size (GstFormat (gststreamer/gstformat.html#GstFormat) format,
                           gint64 (https://docs.gtk.org/glib/types.html#gint64) minsize,
                           gint64 (https://docs.gtk.org/glib/types.html#gint64) maxsize,
                           gboolean (https://docs.gtk.org/glib/types.html#gboolean) async)
```

Create a new buffersize event. The event is sent downstream and notifies elements that they should provide a buffer of the specified dimensions.

When the *async* flag is set, a thread boundary is preferred.

Parameters:

format – buffer format

minsize – minimum buffer size

maxsize – maximum buffer size

async – thread behavior

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_caps

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_caps (GstCaps (gststreamer/gstcaps.html#GstCaps) * caps)
```

Create a new CAPS event for *caps*. The caps event can only travel downstream synchronized with the buffer flow and contains the format of the buffers that will follow after the event.

Parameters:

caps ([transfer: none (Don't free data after the code is done)]) – a GstCaps (gststreamer/gstcaps.html#GstCaps)

Returns ([transfer: full (Free data after the code is done)]) – the new CAPS event.

gst_event_new_custom

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_custom (GstEventType (gststreamer/gstevent.html#GstEventType) type,
                     GstStructure (gststreamer/gststructure.html#GstStructure) * structure)
```

Create a new custom-typed event. This can be used for anything not handled by other event-specific functions to pass an event to another element.

Make sure to allocate an event type with the GST_EVENT_MAKE_TYPE (gststreamer/gstevent.html#GST_EVENT_MAKE_TYPE) macro, assigning a free number and filling in the correct direction and serialization flags.

New custom events can also be created by subclassing the event type if needed.

Parameters:

type – The type of the new event

structure ([transfer: full (Free data after the code is done)]) – the structure for the event. The event will take ownership of the structure.

Returns ([transfer: full (Free data after the code is done)]) – the new custom event.

gst_event_new_eos

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_eos ()
```

Create a new EOS event. The eos event can only travel downstream synchronized with the buffer flow. Elements that receive the EOS event on a pad can return GST_FLOW_EOS (gststreamer/gstpad.html#GST_FLOW_EOS) as a GstFlowReturn (gststreamer/gstpad.html#GstFlowReturn) when data after the EOS event arrives.

The EOS event will travel down to the sink elements in the pipeline which will then post the GST_MESSAGE_EOS (gststreamer/gstmessage.html#GST_MESSAGE_EOS) on the bus after they have finished playing any buffered data.

When all sinks have posted an EOS message, an EOS message is forwarded to the application.

The EOS event itself will not cause any state transitions of the pipeline.

Returns ([transfer: full (Free data after the code is done)]) – the new EOS event.

gst_event_new_flush_start

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_flush_start ()
```

Allocate a new flush start event. The flush start event can be sent upstream and downstream and travels out-of-bounds with the dataflow.

It marks pads as being flushing and will make them return GST_FLOW_FLUSHING (gststreamer/gstpad.html#GST_FLOW_FLUSHING) when used for data flow with `gst_pad_push` (gststreamer/gstpad.html#gst_pad_push), `gst_pad_chain` (gststreamer/gstpad.html#gst_pad_chain), `gst_pad_get_range` (gststreamer/gstpad.html#gst_pad_get_range) and `gst_pad_pull_range` (gststreamer/gstpad.html#gst_pad_pull_range). Any event (except a GST_EVENT_FLUSH_STOP (gststreamer/gstevent.html#GST_EVENT_FLUSH_STOP)) received on a flushing pad will return FALSE (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#FALSE:CAPS>) immediately.

Elements should unlock any blocking functions and exit their streaming functions as fast as possible when this event is received.

This event is typically generated after a seek to flush out all queued data in the pipeline so that the new media is played as soon as possible.

Returns ([transfer: full (Free data after the code is done)]) – a new flush start event.

gst_event_new_flush_stop

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_flush_stop (gboolean (https://docs.gtk.org/glib/types.html#gboolean) reset_time)
```

Allocate a new flush stop event. The flush stop event can be sent upstream and downstream and travels serialized with the dataflow. It is typically sent after sending a FLUSH_START event to make the pads accept data again.

Elements can process this event synchronized with the dataflow since the preceding FLUSH_START event stopped the dataflow.

This event is typically generated to complete a seek and to resume dataflow.

Parameters:

reset_time – if time should be reset

Returns ([transfer: full (Free data after the code is done)]) – a new flush stop event.

gst_event_new_gap

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_gap (GstClockTime (gststreamer/gstclock.html#GstClockTime) timestamp,
                  GstClockTime (gststreamer/gstclock.html#GstClockTime) duration)
```

Create a new GAP event. A gap event can be thought of as conceptually equivalent to a buffer to signal that there is no data for a certain amount of time. This is useful to signal a gap to downstream elements which may wait for data, such as muxers or mixers or overlays, especially for sparse streams such as subtitle streams.

Parameters:

timestamp – the start time (pts) of the gap

duration – the duration of the gap

Returns ([transfer: full (Free data after the code is done)]) – the new GAP event.

gst_event_new_instant_rate_change

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_instant_rate_change (gdouble (https://docs.gtk.org/glib/types.html#gdouble) rate_mu
                                GstSegmentFlags (gststreamer/gstsegment.html#GstSegmentFlags) new
```

Create a new instant-rate-change event. This event is sent by seek handlers (e.g. demuxers) when receiving a seek with the GST_SEEK_FLAG_INSTANT_RATE_CHANGE (gststreamer/gstsegment.html#GST_SEEK_FLAG_INSTANT_RATE_CHANGE) and signals to downstream elements that the playback rate in the existing segment should be immediately multiplied by the *rate_multiplier* factor.

The flags provided replace any flags in the existing segment, for the flags within the GST_SEGMENT_INSTANT_FLAGS (gststreamer/gstsegment.html#GST_SEGMENT_INSTANT_FLAGS) set. Other GstSegmentFlags are ignored and not transferred in the event.

Parameters:

rate_multiplier – the multiplier to be applied to the playback rate

new_flags – A new subset of segment flags to replace in segments

Returns ([transfer: full (Free data after the code is done)]) – the new instant-rate-change event.

Since : 1.18

gst_event_new_instant_rate_sync_time

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_instant_rate_sync_time (gdouble (https://docs.gtk.org/glib/types.html#gdouble) rate_
                                GstClockTime (gststreamer/gstclock.html#GstClockTime) running_
                                GstClockTime (gststreamer/gstclock.html#GstClockTime) upstre
```

Create a new instant-rate-sync-time event. This event is sent by the pipeline to notify elements handling the instant-rate-change event about the running-time when the new rate should be applied. The running time may be in the past when elements handle this event, which can lead to switching artifacts. The magnitude of those

depends on the exact timing of event delivery to each element and the magnitude of the change in playback rate being applied.

The *running_time* and *upstream_running_time* are the same if this is the first instant-rate adjustment, but will differ for later ones to compensate for the accumulated offset due to playing at a rate different to the one indicated in the playback segments.

Parameters:

rate_multiplier – the new playback rate multiplier to be applied

running_time – Running time when the rate change should be applied

upstream_running_time – The upstream-centric running-time when the rate change should be applied.

Returns ([transfer: full (Free data after the code is done)]) – the new instant-rate-sync-time event.

Since : 1.18

gst_event_new_latency

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_latency (GstClockTime (gststreamer/gstclock.html#GstClockTime) latency)
```

Create a new latency event. The event is sent upstream from the sinks and notifies elements that they should add an additional *latency* to the running time before synchronising against the clock.

The latency is mostly used in live sinks and is always expressed in the time format.

Parameters:

latency – the new latency value

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_navigation

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_navigation (GstStructure (gststreamer/gststructure.html#GstStructure) * structure)
```

Create a new navigation event from the given description.

Parameters:

structure ([transfer: full (Free data after the code is done)]) – description of the event. The event will take ownership of the structure. See GstNavigation (video/gstnavigation.html#GstNavigation) for more specific constructors.

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_protection

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_protection (const gchar (https://docs.gtk.org/glib/types.html#gchar) * system_id,
                        GstBuffer (gststreamer/gstbuffer.html#GstBuffer) * data,
                        const gchar (https://docs.gtk.org/glib/types.html#gchar) * origin)
```

Creates a new event containing information specific to a particular protection system (uniquely identified by *system_id*), by which that protection system can acquire key(s) to decrypt a protected stream.

In order for a decryption element to decrypt media protected using a specific system, it first needs all the protection system specific information necessary to acquire the decryption key(s) for that stream. The functions defined here enable this information to be passed in events from elements that extract it (e.g., ISOBMFF demuxers, MPEG DASH demuxers) to protection decrypter elements that use it.

Events containing protection system specific information are created using *gst_event_new_protection* (gststreamer/gstevent.html#gst_event_new_protection), and they can be parsed by downstream elements using *gst_event_parse_protection* (gststreamer/gstevent.html#gst_event_parse_protection).

In Common Encryption, protection system specific information may be located within ISOBMFF files, both in movie (moov) boxes and movie fragment (moof) boxes; it may also be contained in ContentProtection elements within MPEG DASH MPDs. The events created by *gst_event_new_protection* (gststreamer/gstevent.html#gst_event_new_protection) contain data identifying from which of these locations the encapsulated protection system specific information originated. This origin information is required as some protection systems use different encodings depending upon where the information originates.

The events returned by *gst_event_new_protection* (gststreamer/gstevent.html#gst_event_new_protection) are implemented in such a way as to ensure that the most recently-pushed protection info event of a particular *origin* and *system_id* will be stuck to the output pad of the sending element.

Parameters:

system_id ([transfer: none (Don't free data after the code is done)]) – a string holding a UUID that uniquely identifies a protection system.

data ([transfer: none (Don't free data after the code is done)]) – a GstBuffer (gststreamer/gstbuffer.html#GstBuffer) holding protection system specific information. The reference count of the buffer will be incremented by one.

origin – a string indicating where the protection information carried in the event was extracted from. The allowed values of this string will depend upon the protection scheme.

Returns ([transfer: full (Free data after the code is done)]) – a GST_EVENT_PROTECTION (gststreamer/gstevent.html#GST_EVENT_PROTECTION) event.

Since : 1.6

gst_event_new_qos

```
GstEvent (gstreamer/gstevent.html#GstEvent) *
gst_event_new_qos (GstQOSType (gstreamer/gstevent.html#GstQOSType) type,
                  gdouble (https://docs.gtk.org/glib/types.html#gdouble) proportion,
                  GstClockTimeDiff (gstreamer/gstclock.html#GstClockTimeDiff) diff,
                  GstClockTime (gstreamer/gstclock.html#GstClockTime) timestamp)
```

Allocate a new qos event with the given values. The QOS event is generated in an element that wants an upstream element to either reduce or increase its rate because of high/low CPU load or other resource usage such as network performance or throttling. Typically sinks generate these events for each buffer they receive.

type indicates the reason for the QoS event. `GST_QOS_TYPE_OVERFLOW` (gstreamer/gstevent.html#GST_QOS_TYPE_OVERFLOW) is used when a buffer arrived in time or when the sink cannot keep up with the upstream datarate. `GST_QOS_TYPE_UNDERFLOW` (gstreamer/gstevent.html#GST_QOS_TYPE_UNDERFLOW) is when the sink is not receiving buffers fast enough and thus has to drop late buffers. `GST_QOS_TYPE_THROTTLE` (gstreamer/gstevent.html#GST_QOS_TYPE_THROTTLE) is used when the datarate is artificially limited by the application, for example to reduce power consumption.

proportion indicates the real-time performance of the streaming in the element that generated the QoS event (usually the sink). The value is generally computed based on more long term statistics about the streams timestamps compared to the clock. A value < 1.0 indicates that the upstream element is producing data faster than real-time. A value > 1.0 indicates that the upstream element is not producing data fast enough. 1.0 is the ideal *proportion* value. The proportion value can safely be used to lower or increase the quality of the element.

diff is the difference against the clock in running time of the last buffer that caused the element to generate the QOS event. A negative value means that the buffer with *timestamp* arrived in time. A positive value indicates how late the buffer with *timestamp* was. When throttling is enabled, *diff* will be set to the requested throttling interval.

timestamp is the timestamp of the last buffer that cause the element to generate the QOS event. It is expressed in running time and thus an ever increasing value.

The upstream element can use the *diff* and *timestamp* values to decide whether to process more buffers. For positive *diff*, all buffers with timestamp <= *timestamp* + *diff* will certainly arrive late in the sink as well. A (negative) *diff* value so that *timestamp* + *diff* would yield a result smaller than 0 is not allowed.

The application can use general event probes to intercept the QoS event and implement custom application specific QoS handling.

Parameters:

type – the QoS type

proportion – the proportion of the qos message

diff – The time difference of the last Clock sync

timestamp – The timestamp of the buffer

Returns ([transfer: full (Free data after the code is done)]) – a new QOS event.

gst_event_new_reconfigure

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_reconfigure ()
```

Create a new reconfigure event. The purpose of the reconfigure event is to travel upstream and make elements renegotiate their caps or reconfigure their buffer pools. This is useful when changing properties on elements or changing the topology of the pipeline.

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_seek

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_seek (gdouble (https://docs.gtk.org/glib/types.html#gdouble) rate,
                   GstFormat (gststreamer/gstformat.html#GstFormat) format,
                   GstSeekFlags (gststreamer/gstsegment.html#GstSeekFlags) flags,
                   GstSeekType (gststreamer/gstsegment.html#GstSeekType) start_type,
                   gint64 (https://docs.gtk.org/glib/types.html#gint64) start,
                   GstSeekType (gststreamer/gstsegment.html#GstSeekType) stop_type,
                   gint64 (https://docs.gtk.org/glib/types.html#gint64) stop)
```

Allocate a new seek event with the given parameters.

The seek event configures playback of the pipeline between *start* to *stop* at the speed given in *rate*, also called a playback segment. The *start* and *stop* values are expressed in *format*.

A *rate* of 1.0 means normal playback rate, 2.0 means double speed. Negatives values means backwards playback. A value of 0.0 for the rate is not allowed and should be accomplished instead by PAUSING the pipeline.

A pipeline has a default playback segment configured with a start position of 0, a stop position of -1 and a rate of 1.0. The currently configured playback segment can be queried with GST_QUERY_SEGMENT (gststreamer/gstquery.html#GST_QUERY_SEGMENT).

start_type and *stop_type* specify how to adjust the currently configured start and stop fields in playback segment. Adjustments can be made relative or absolute to the last configured values. A type of GST_SEEK_TYPE_NONE (gststreamer/gstsegment.html#GST_SEEK_TYPE_NONE) means that the position should not be updated.

When the rate is positive and *start* has been updated, playback will start from the newly configured start position.

For negative rates, playback will start from the newly configured stop position (if any). If the stop position is updated, it must be different from -1 (#GST_CLOCK_TIME_NONE) for negative rates.

It is not possible to seek relative to the current playback position, to do this, PAUSE the pipeline, query the current playback position with GST_QUERY_POSITION (gststreamer/gstquery.html#GST_QUERY_POSITION) and update the playback segment current position with a GST_SEEK_TYPE_SET (gststreamer/gstsegment.html#GST_SEEK_TYPE_SET) to the desired position.

Parameters:

rate – The new playback rate

format – The format of the seek values

flags – The optional seek flags

start_type – The type and flags for the new start position

start – The value of the new start position

stop_type – The type and flags for the new stop position

stop – The value of the new stop position

Returns ([transfer: full (Free data after the code is done)]) – a new seek event.

gst_event_new_segment

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_segment (const GstSegment (gststreamer/gstsegment.html#GstSegment) * segment)
```

Create a new SEGMENT event for *segment*. The segment event can only travel downstream synchronized with the buffer flow and contains timing information and playback properties for the buffers that will follow.

The segment event marks the range of buffers to be processed. All data not within the segment range is not to be processed. This can be used intelligently by plugins to apply more efficient methods of skipping unneeded data. The valid range is expressed with the *start* and *stop* values.

The time value of the segment is used in conjunction with the start value to convert the buffer timestamps into the stream time. This is usually done in sinks to report the current stream_time. *time* represents the stream_time of a buffer carrying a timestamp of *start*. *time* cannot be -1.

start cannot be -1, *stop* can be -1. If there is a valid *stop* given, it must be greater or equal the *start*, including when the indicated playback *rate* is < 0.

The *applied_rate* value provides information about any rate adjustment that has already been made to the timestamps and content on the buffers of the stream. (*@rate* * *applied_rate*) should always equal the rate that has been requested for playback. For example, if an element has an input segment with intended playback *rate* of 2.0 and *applied_rate* of 1.0, it can adjust incoming timestamps and buffer content by half and output a segment event with *rate* of 1.0 and *applied_rate* of 2.0

After a segment event, the buffer stream time is calculated with:

$\text{time} + (\text{TIMESTAMP}(\text{buf}) - \text{start}) * \text{ABS}(\text{rate} * \text{applied_rate})$

Parameters:

segment ([transfer: none (Don't free data after the code is done)]) – a GstSegment
(gststreamer/gstsegment.html#GstSegment)

Returns ([transfer: full (Free data after the code is done)]) – the new SEGMENT event.

gst_event_new_segment_done

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_segment_done (GstFormat (gststreamer/gstformat.html#GstFormat) format,
                           gint64 (https://docs.gtk.org/glib/types.html#gint64) position)
```

Create a new segment-done event. This event is sent by elements that finish playback of a segment as a result of a segment seek.

Parameters:

format – The format of the position being done

position – The position of the segment being done

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_select_streams

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_select_streams (GList (https://docs.gtk.org/glib/struct.List.html) * streams)
```

Allocate a new select-streams event.

The select-streams event requests the specified *streams* to be activated.

The list of *streams* corresponds to the "Stream ID" of each stream to be activated. Those ID can be obtained via the GstStream (gststreamer/gststreams.html#GstStream) objects present in GST_EVENT_STREAM_START (gststreamer/gstevent.html#GST_EVENT_STREAM_START), GST_EVENT_STREAM_COLLECTION (gststreamer/gstevent.html#GST_EVENT_STREAM_COLLECTION) or GST_MESSAGE_STREAM_COLLECTION (gststreamer/gstmessage.html#GST_MESSAGE_STREAM_COLLECTION).

Note: The list of *streams* can not be empty.

Parameters:

streams ([element-type (Generic and defining element of containers and arrays) utf8][transfer: none (Don't free data after the code is done)]) – the list of streams to activate

Returns ([transfer: full (Free data after the code is done)]) – a new select-streams event.

Since : 1.10

gst_event_new_sink_message

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_sink_message (const gchar (https://docs.gtk.org/glib/types.html#gchar) * name,
                           GstMessage (gststreamer/gstmessage.html#GstMessage) * msg)
```

Create a new sink-message event. The purpose of the sink-message event is to instruct a sink to post the message contained in the event synchronized with the stream.

name is used to store multiple sticky events on one pad.

Parameters:

name – a name for the event

msg ([transfer: none (Don't free data after the code is done)]) – the GstMessage (gststreamer/gstmessage.html#GstMessage) to be posted

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_step

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_step (GstFormat (gststreamer/gstformat.html#GstFormat) format,
                    guint64 (https://docs.gtk.org/glib/types.html#guint64) amount,
                    gdouble (https://docs.gtk.org/glib/types.html#gdouble) rate,
                    gboolean (https://docs.gtk.org/glib/types.html#gboolean) flush,
                    gboolean (https://docs.gtk.org/glib/types.html#gboolean) intermediate)
```

Create a new step event. The purpose of the step event is to instruct a sink to skip *amount* (expressed in *format*) of media. It can be used to implement stepping through the video frame by frame or for doing fast trick modes.

A rate of ≤ 0.0 is not allowed. Pause the pipeline, for the effect of rate = 0.0 or first reverse the direction of playback using a seek event to get the same effect as rate < 0.0.

The *flush* flag will clear any pending data in the pipeline before starting the step operation.

The *intermediate* flag instructs the pipeline that this step operation is part of a larger step operation.

Parameters:

format – the format of *amount*

amount – the amount of data to step

rate – the step rate

flush – flushing steps

intermediate – intermediate steps

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_stream_collection

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_stream_collection (GstStreamCollection (gststreamer/gststreamcollection.html#GstStreamCollection) collection)
```

Create a new STREAM_COLLECTION event. The stream collection event can only travel downstream synchronized with the buffer flow.

Source elements, demuxers and other elements that manage collections of streams and post GstStreamCollection (gststreamer/gststreamcollection.html#GstStreamCollection) messages on the bus also send this event downstream on each pad involved in the collection, so that activation of a new collection can be tracked through the downstream data flow.

Parameters:

collection ([transfer: none (Don't free data after the code is done)]) – Active collection for this data flow

Returns ([transfer: full (Free data after the code is done)]) – the new STREAM_COLLECTION event.

Since : 1.10

gst_event_new_stream_group_done

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_stream_group_done (guint (https://docs.gtk.org/glib/types.html#guint) group_id)
```

Create a new Stream Group Done event. The stream-group-done event can only travel downstream synchronized with the buffer flow. Elements that receive the event on a pad should handle it mostly like EOS, and emit any data or pending buffers that would depend on more data arriving and unblock, since there won't be any more data.

This event is followed by EOS at some point in the future, and is generally used when switching pads - to unblock downstream so that new pads can be exposed before sending EOS on the existing pads.

Parameters:

group_id – the group id of the stream group which is ending

Returns ([transfer: full (Free data after the code is done)]) – the new stream-group-done event.

Since : 1.10

gst_event_new_stream_start

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_stream_start (const gchar (https://docs.gtk.org/glib/types.html#gchar) * stream_id)
```

Create a new STREAM_START event. The stream start event can only travel downstream synchronized with the buffer flow. It is expected to be the first event that is sent for a new stream.

Source elements, demuxers and other elements that create new streams are supposed to send this event as the first event of a new stream. It should not be sent after a flushing seek or in similar situations and is used to mark the beginning of a new logical stream. Elements combining multiple streams must ensure that this event is only forwarded downstream once and not for every single input stream.

The *stream_id* should be a unique string that consists of the upstream stream-id, / as separator and a unique stream-id for this specific stream. A new stream-id should only be created for a stream if the upstream stream is split into (potentially) multiple new streams, e.g. in a demuxer, but not for every single element in the pipeline. `gst_pad_create_stream_id` (gststreamer/gstpad.html#gst_pad_create_stream_id) or `gst_pad_create_stream_id_printf` (gststreamer/gstpad.html#gst_pad_create_stream_id_printf) can be used to create a stream-id. There are no particular semantics for the stream-id, though it should be deterministic (to support stream matching) and it might be used to order streams (besides any information conveyed by stream flags).

Parameters:

stream_id – Identifier for this stream

Returns ([transfer: full (Free data after the code is done)]) – the new STREAM_START event.

gst_event_new_tag

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_tag (GstTagList (gststreamer/gsttaglist.html#GstTagList) * taglist)
```

Generates a metadata tag event from the given *taglist*.

The scope of the taglist specifies if the taglist applies to the complete medium or only to this specific stream. As the tag event is a sticky event, elements should merge tags received from upstream with a given scope with their own tags with the same scope and create a new tag event from it.

Parameters:

taglist ([transfer: full (Free data after the code is done)]) – metadata list. The event will take ownership of the taglist.

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent)

gst_event_new_toc

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_new_toc (GstToc (gststreamer/gsttoc.html#GstToc) * toc,
                  gboolean (https://docs.gtk.org/glib/types.html#gboolean) updated)
```

Generate a TOC event from the given *toc*. The purpose of the TOC event is to inform elements that some kind of the TOC was found.

Parameters:

toc ([transfer: none (Don't free data after the code is done)]) – GstToc (gststreamer/gsttoc.html#GstToc) structure.

updated – whether *toc* was updated or not.

Returns ([transfer: full (Free data after the code is done)]) – a new GstEvent (gststreamer/gstevent.html#GstEvent).

gst_event_new_toc_select

```
GstEvent (gststreamer/gstevent.html#GstEvent) *  
gst_event_new_toc_select (const gchar (https://docs.gtk.org/glib/types.html#gchar) * uid)
```

Generate a TOC select event with the given *uid*. The purpose of the TOC select event is to start playback based on the TOC's entry with the given *uid*.

Parameters:

uid – UID in the TOC to start playback from.

Returns – a new GstEvent (gststreamer/gstevent.html#GstEvent).

Methods

gst_event_copy

```
GstEvent (gststreamer/gstevent.html#GstEvent) *  
gst_event_copy (const GstEvent (gststreamer/gstevent.html#GstEvent) * event)
```

Copy the event using the event specific copy function.

Parameters:

event – The event to copy

Returns ([transfer: full (Free data after the code is done)]) – the new event

gst_event_copy_segment

```
gst_event_copy_segment (GstEvent (gststreamer/gstevent.html#GstEvent) * event,  
                        GstSegment (gststreamer/gstsegment.html#GstSegment) * segment)
```

Parses a segment *event* and copies the GstSegment (gststreamer/gstsegment.html#GstSegment) into the location given by *segment*.

Parameters:

event – The event to parse

segment – a pointer to a GstSegment (gststreamer/gstsegment.html#GstSegment)

gst_event_get_running_time_offset

```
gint64 (https://docs.gtk.org/glib/types.html#gint64)
gst_event_get_running_time_offset (GstEvent (gstreamer/gstevent.html#GstEvent) * event)
```

Retrieve the accumulated running time offset of the event.

Events passing through GstPad ([gstreamer/gstpad.html#GstPad](https://docs.gtk.org/gstreamer/gstpad.html#GstPad)) that have a running time offset set via `gst_pad_set_offset` ([gstreamer/gstpad.html#gst_pad_set_offset](https://docs.gtk.org/gstreamer/gstpad.html#gst_pad_set_offset)) will get their offset adjusted according to the pad's offset.

If the event contains any information that related to the running time, this information will need to be updated before usage with this offset.

Parameters:

event – A GstEvent ([gstreamer/gstevent.html#GstEvent](https://docs.gtk.org/gstreamer/gstevent.html#GstEvent)).

Returns – The event's running time offset MT safe.

Since : 1.4

gst_event_get_seqnum

```
guint32 (https://docs.gtk.org/glib/types.html#guint32)
gst_event_get_seqnum (GstEvent (gstreamer/gstevent.html#GstEvent) * event)
```

Retrieve the sequence number of a event.

Events have ever-incrementing sequence numbers, which may also be set explicitly via `gst_event_set_seqnum` ([gstreamer/gstevent.html#gst_event_set_seqnum](https://docs.gtk.org/gstreamer/gstevent.html#gst_event_set_seqnum)). Sequence numbers are typically used to indicate that a event corresponds to some other set of events or messages, for example an EOS event corresponding to a SEEK event. It is considered good practice to make this correspondence when possible, though it is not required.

Note that events and messages share the same sequence number incrementor; two events or messages will never have the same sequence number unless that correspondence was made explicitly.

Parameters:

event – A GstEvent ([gstreamer/gstevent.html#GstEvent](https://docs.gtk.org/gstreamer/gstevent.html#GstEvent)).

Returns – The event's sequence number. MT safe.

gst_event_get_structure

```
const GstStructure (gstreamer/gststructure.html#GstStructure) *
gst_event_get_structure (GstEvent (gstreamer/gstevent.html#GstEvent) * event)
```

Access the structure of the event.

Parameters:

event – The GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)).

Returns ([transfer: none (Don't free data after the code is done)][nullable (NULL may be passed to the value)]) – The structure of the event. The structure is still owned by the event, which means that you should not free it and that the pointer becomes invalid when you free the event. MT safe.

gst_event_has_name

```
gboolean (https://docs.gtk.org/glib/types.html#gboolean)
gst_event_has_name (GstEvent (gstreamer/gstevent.html#GstEvent) * event,
                   const gchar (https://docs.gtk.org/glib/types.html#gchar) * name)
```

Checks if *event* has the given *name*. This function is usually used to check the name of a custom event.

Parameters:

event – The GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)).

name – name to check

Returns – TRUE (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#TRUE:CAPS>) if *name* matches the name of the event structure.

gst_event_is_writable

```
gboolean (https://docs.gtk.org/glib/types.html#gboolean)
gst_event_is_writable (const GstEvent (gstreamer/gstevent.html#GstEvent) * event)
```

Tests if you can safely modify *event*. It is only safe to modify event when there is only one owner of the event - ie, the object is writable.

Parameters:

event – a GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent))

Returns – *No description available*

gst_event_make_writable

```
GstEvent (gstreamer/gstevent.html#GstEvent) *
gst_event_make_writable (GstEvent (gstreamer/gstevent.html#GstEvent) * event)
```

Returns a writable copy of *event*.

If there is only one reference count on *event*, the caller must be the owner, and so this function will return the event object unchanged. If on the other hand there is more than one reference on the object, a new event object will be returned. The caller's reference on *event* will be removed, and instead the caller will own a reference to the returned object.

In short, this function unrefs the event in the argument and refs the event that it returns. Don't access the argument after calling this function. See also: `gst_event_ref` ([gststreamer/gstevent.html#gst_event_ref](#)).

Parameters:

event ([transfer: full (Free data after the code is done)]) – a GstEvent ([gststreamer/gstevent.html#GstEvent](#))

Returns ([transfer: full (Free data after the code is done)]) – a writable event which may or may not be the same as *event*

gst_event_parse_buffer_size

```
gst_event_parse_buffer_size (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                             GstFormat (gststreamer/gstformat.html#GstFormat) * format,
                             gint64 (https://docs.gtk.org/glib/types.html#gint64) * minsize,
                             gint64 (https://docs.gtk.org/glib/types.html#gint64) * maxsize,
                             gboolean (https://docs.gtk.org/glib/types.html#gboolean) * async)
```

Get the format, minsize, maxsize and async-flag in the buffersize event.

Parameters:

event – The event to query

format ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the format in

minsize ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the minsize in

maxsize ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the maxsize in

async ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the async-flag in

gst_event_parse_caps

```
gst_event_parse_caps (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                     GstCaps (gststreamer/gstcaps.html#GstCaps) ** caps)
```

Get the caps from *event*. The caps remains valid as long as *event* remains valid.

Parameters:

event – The event to parse

caps ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)] [transfer: none (Don't free data after the code is done)]) – A pointer to the caps

gst_event_parse_flush_stop

```
gst_event_parse_flush_stop (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                           gboolean (https://docs.gtk.org/glib/types.html#gboolean) * reset_time
```

Parse the FLUSH_STOP event and retrieve the *reset_time* member.

Parameters:

event – The event to parse

reset_time ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – if time should be reset

gst_event_parse_gap

```
gst_event_parse_gap (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                    GstClockTime (gststreamer/gstclock.html#GstClockTime) * timestamp,
                    GstClockTime (gststreamer/gstclock.html#GstClockTime) * duration)
```

Extract timestamp and duration from a new GAP event.

Parameters:

event – a GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP)

timestamp ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location where to store the start time (pts) of the gap, or NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

duration ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location where to store the duration of the gap, or NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

gst_event_parse_gap_flags

```
gst_event_parse_gap_flags (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                           GstGapFlags (gststreamer/gstevent.html#GstGapFlags) * flags)
```

Retrieve the gap flags that may have been set on a gap event with *gst_event_set_gap_flags* (gststreamer/gstevent.html#gst_event_set_gap_flags).

Parameters:

event – a GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP)

flags ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a GstGapFlags (gststreamer/gstevent.html#GstGapFlags) or NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

Since : 1.20

gst_event_parse_group_id

```
gboolean (https://docs.gtk.org/glib/types.html#gboolean)
gst_event_parse_group_id (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                          guint (https://docs.gtk.org/glib/types.html#guint) * group_id)
```

Parameters:

event – a stream-start event

group_id ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – address of variable where to store the group id

Returns – TRUE (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#TRUE:CAPS) if a group id was set on the event and could be parsed, FALSE (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#FALSE:CAPS) otherwise.

Since : 1.2

gst_event_parse_instant_rate_change

```
gst_event_parse_instant_rate_change (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                     gdouble (https://docs.gtk.org/glib/types.html#gdouble) * rate,
                                     GstSegmentFlags (gststreamer/gstsegment.html#GstSegmentFlags) * flags)
```

Extract rate and flags from an instant-rate-change event.

Parameters:

event – a GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_INSTANT_RATE_CHANGE (gststreamer/gstevent.html#GST_EVENT_INSTANT_RATE_CHANGE)

rate_multiplier ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location in which to store the rate multiplier of the instant-rate-change event, or NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

new_flags ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location in which to store the new segment flags of the instant-rate-change event, or NULL

(<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>)

Since : 1.18

gst_event_parse_instant_rate_sync_time

```
gst_event_parse_instant_rate_sync_time (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                         gdouble (https://docs.gtk.org/glib/types.html#gdouble) * rate_multiplier,
                                         GstClockTime (gststreamer/gstclock.html#GstClockTime) * running_time,
                                         GstClockTime (gststreamer/gstclock.html#GstClockTime) * upstream_running_time)
```

Extract the rate multiplier and running times from an instant-rate-sync-time event.

Parameters:

event – a GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_INSTANT_RATE_CHANGE (gststreamer/gstevent.html#GST_EVENT_INSTANT_RATE_CHANGE)

rate_multiplier ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location where to store the rate of the instant-rate-sync-time event, or NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>)

running_time ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location in which to store the running time of the instant-rate-sync-time event, or NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>)

upstream_running_time ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – location in which to store the upstream running time of the instant-rate-sync-time event, or NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>)

Since : 1.18

gst_event_parse_latency

```
gst_event_parse_latency (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                        GstClockTime (gststreamer/gstclock.html#GstClockTime) * latency)
```

Get the latency in the latency event.

Parameters:

event – The event to query

latency ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the latency in.

gst_event_parse_protection

```
gst_event_parse_protection (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                           const gchar (https://docs.gtk.org/glib/types.html#gchar) ** system_id,
                           GstBuffer (gststreamer/gstbuffer.html#GstBuffer) ** data,
                           const gchar (https://docs.gtk.org/glib/types.html#gchar) ** origin)
```

Parses an event containing protection system specific information and stores the results in *system_id*, *data* and *origin*. The data stored in *system_id*, *origin* and *data* are valid until *event* is released.

Parameters:

event – a GST_EVENT_PROTECTION (gststreamer/gstevent.html#GST_EVENT_PROTECTION) event.

system_id ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]] – pointer to store the UUID string uniquely identifying a content protection system.

data ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]] – pointer to store a GstBuffer (gststreamer/gstbuffer.html#GstBuffer) holding protection system specific information.

origin ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]] – pointer to store a value that indicates where the protection information carried by *event* was extracted from.

Since : 1.6

gst_event_parse_qos

```
gst_event_parse_qos (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                    GstQOSType (gststreamer/gstevent.html#GstQOSType) * type,
                    gdouble (https://docs.gtk.org/glib/types.html#gdouble) * proportion,
                    GstClockTimeDiff (gststreamer/gstclock.html#GstClockTimeDiff) * diff,
                    GstClockTime (gststreamer/gstclock.html#GstClockTime) * timestamp)
```

Get the type, proportion, diff and timestamp in the qos event. See *gst_event_new_qos* (gststreamer/gstevent.html#gst_event_new_qos) for more information about the different QoS values.

timestamp will be adjusted for any pad offsets of pads it was passing through.

Parameters:

event – The event to query

type ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]] – A pointer to store the QoS type in

proportion ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]] – A pointer to store the proportion in

diff ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)])
 – A pointer to store the diff in

timestamp ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – A pointer to store the timestamp in

gst_event_parse_seek

```
gst_event_parse_seek (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                     gdouble (https://docs.gtk.org/glib/types.html#gdouble) * rate,
                     GstFormat (gststreamer/gstformat.html#GstFormat) * format,
                     GstSeekFlags (gststreamer/gstsegment.html#GstSeekFlags) * flags,
                     GstSeekType (gststreamer/gstsegment.html#GstSeekType) * start_type,
                     gint64 (https://docs.gtk.org/glib/types.html#gint64) * start,
                     GstSeekType (gststreamer/gstsegment.html#GstSeekType) * stop_type,
                     gint64 (https://docs.gtk.org/glib/types.html#gint64) * stop)
```

Parses a seek *event* and stores the results in the given result locations.

Parameters:

event – a seek event

rate ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)])
 – result location for the rate

format ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – result location for the stream format

flags ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – result location for the GstSeekFlags (gststreamer/gstsegment.html#GstSeekFlags)

start_type ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – result location for the GstSeekType (gststreamer/gstsegment.html#GstSeekType) of the start position

start ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – result location for the start position expressed in *format*

stop_type ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – result location for the GstSeekType (gststreamer/gstsegment.html#GstSeekType) of the stop position

stop ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)])
 – result location for the stop position expressed in *format*

gst_event_parse_seek_trickmode_interval

```
gst_event_parse_seek_trickmode_interval (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                         GstClockTime (gststreamer/gstclock.html#GstClockTime) * in
```

Retrieve the trickmode interval that may have been set on a seek event with

`gst_event_set_seek_trickmode_interval (gststreamer/gstevent.html#gst_event_set_seek_trickmode_interval).`

Parameters:

event – No description available

interval ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – interval

Since : 1.16

gst_event_parse_segment

```
gst_event_parse_segment (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                        const GstSegment (gststreamer/gstsegment.html#GstSegment) ** segment)
```

Parses a segment *event* and stores the result in the given *segment* location. *segment* remains valid only until the *event* is freed. Don't modify the segment and make a copy if you want to modify it or store it for later use.

Parameters:

event – The event to parse

segment ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]) – a pointer to a GstSegment (gststreamer/gstsegment.html#GstSegment)

gst_event_parse_segment_done

```
gst_event_parse_segment_done (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                             GstFormat (gststreamer/gstformat.html#GstFormat) * format,
                             gint64 (https://docs.gtk.org/glib/types.html#gint64) * position)
```

Extracts the position and format from the segment done message.

Parameters:

event – A valid GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_SEGMENT_DONE.

format ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – Result location for the format, or NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

position ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – Result location for the position, or NULL (https://web.mit.edu/barnowl/share/gtk-

doc/html/glib/glib-Standard-Macros.html#NULL:CAPS)

gst_event_parse_select_streams

```
gst_event_parse_select_streams (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                               GList (https://docs.gtk.org/glib/struct.List.html) ** streams)
```

Parse the SELECT_STREAMS event and retrieve the contained streams.

Parameters:

event – The event to parse

streams ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][element-type (Generic and defining element of containers and arrays) utf8][transfer: full (Free data after the code is done)]] – the streams

Since : 1.10

gst_event_parse_sink_message

```
gst_event_parse_sink_message (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                              GstMessage (gststreamer/gstmessage.html#GstMessage) ** msg)
```

Parse the sink-message event. Unref *msg* after usage.

Parameters:

event – The event to query

msg ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: full (Free data after the code is done)]] – a pointer to store the GstMessage (gststreamer/gstmessage.html#GstMessage) in.

gst_event_parse_step

```
gst_event_parse_step (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                     GstFormat (gststreamer/gstformat.html#GstFormat) * format,
                     guint64 (https://docs.gtk.org/glib/types.html#guint64) * amount,
                     gdouble (https://docs.gtk.org/glib/types.html#gdouble) * rate,
                     gboolean (https://docs.gtk.org/glib/types.html#gboolean) * flush,
                     gboolean (https://docs.gtk.org/glib/types.html#gboolean) * intermediate)
```

Parse the step event.

Parameters:

event – The event to query

format ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a pointer to store the format in

amount ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a pointer to store the amount in

rate ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a pointer to store the rate in

flush ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a pointer to store the flush boolean in

intermediate ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – a pointer to store the intermediate boolean in

gst_event_parse_stream

```
gst_event_parse_stream (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                        GstStream (gststreamer/gststreams.html#GstStream) ** stream)
```

Parse a stream-start *event* and extract the *GstStream* (gststreamer/gststreams.html#GstStream) from it.

Parameters:

event – a stream-start event

stream ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: full (Free data after the code is done))] – address of variable to store the stream

Since : 1.10

gst_event_parse_stream_collection

```
gst_event_parse_stream_collection (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                   GstStreamCollection (gststreamer/gststreamcollection.html#GstStreamCollection) ** collection)
```

Retrieve new *GstStreamCollection* (gststreamer/gststreamcollection.html#GstStreamCollection) from *STREAM_COLLECTION* event *event*.

Parameters:

event – a stream-collection event

collection ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: full (Free data after the code is done))] – pointer to store the collection.

Since : 1.10

gst_event_parse_stream_flags

```
gst_event_parse_stream_flags (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                             GstStreamFlags (gststreamer/gstevent.html#GstStreamFlags) * flags)
```

Parameters:

event – a stream-start event

flags ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – address of variable where to store the stream flags

Since : 1.2

gst_event_parse_stream_group_done

```
gst_event_parse_stream_group_done (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                   guint (https://docs.gtk.org/glib/types.html#guint) * group_id)
```

Parse a stream-group-done *event* and store the result in the given *group_id* location.

Parameters:

event – a stream-group-done event.

group_id ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – address of variable to store the group id into

Since : 1.10

gst_event_parse_stream_start

```
gst_event_parse_stream_start (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                              const gchar (https://docs.gtk.org/glib/types.html#gchar) ** stream_id)
```

Parse a stream-id *event* and store the result in the given *stream_id* location. The string stored in *stream_id* must not be modified and will remain valid only until *event* gets freed. Make a copy if you want to modify it or store it for later use.

Parameters:

event – a stream-start event.

stream_id ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]) – pointer to store the stream-id

gst_event_parse_tag

```
gst_event_parse_tag (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                    GstTagList (gststreamer/gsttaglist.html#GstTagList) ** taglist)
```

Parses a tag *event* and stores the results in the given *taglist* location. No reference to the taglist will be returned, it remains valid only until the *event* is freed. Don't modify or free the taglist, make a copy if you want to modify it or store it for later use.

Parameters:

event – a tag event

taglist ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: none (Don't free data after the code is done)]) – pointer to metadata list

gst_event_parse_toc

```
gst_event_parse_toc (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                    GstToc (gststreamer/gsttoc.html#GstToc) ** toc,
                    gboolean (https://docs.gtk.org/glib/types.html#gboolean) * updated)
```

Parse a TOC *event* and store the results in the given *toc* and *updated* locations.

Parameters:

event – a TOC event.

toc ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)][transfer: full (Free data after the code is done)]) – pointer to GstToc (gststreamer/gsttoc.html#GstToc) structure.

updated ([out (Parameter for returning results)][optional (NULL may be passed instead of a pointer to a location)]) – pointer to store TOC updated flag.

gst_event_parse_toc_select

```
gst_event_parse_toc_select (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                            gchar (https://docs.gtk.org/glib/types.html#gchar) ** uid)
```

Parse a TOC select *event* and store the results in the given *uid* location.

Parameters:

event – a TOC select event.

uid ([out (Parameter for returning results)][transfer: full (Free data after the code is done)][optional (NULL may be passed instead of a pointer to a location)]) – storage for the selection UID.

gst_event_ref

```
GstEvent (gststreamer/gstevent.html#GstEvent) *
gst_event_ref (GstEvent (gststreamer/gstevent.html#GstEvent) * event)
```

Increase the refcount of this event.

Parameters:

event – The event to refcount

Returns ([transfer: full (Free data after the code is done)]) – *event* (for convenience when doing assignments)

gst_event_set_gap_flags

```
gst_event_set_gap_flags (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                        GstGapFlags (gststreamer/gstevent.html#GstGapFlags) flags)
```

Sets *flags* on *event* to give additional information about the reason for the GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP).

Parameters:

event – a GstEvent (gststreamer/gstevent.html#GstEvent) of type GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP)

flags – a GstGapFlags (gststreamer/gstevent.html#GstGapFlags)

Since : 1.20

gst_event_set_group_id

```
gst_event_set_group_id (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                        guint (https://docs.gtk.org/glib/types.html#guint) group_id)
```

All streams that have the same group id are supposed to be played together, i.e. all streams inside a container file should have the same group id but different stream ids. The group id should change each time the stream is started, resulting in different group ids each time a file is played for example.

Use `gst_util_group_id_next` (gststreamer/gstutils.html#gst_util_group_id_next) to get a new group id.

Parameters:

event – a stream-start event

group_id – the group id to set

Since : 1.2

gst_event_set_running_time_offset

```
gst_event_set_running_time_offset (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                   gint64 (https://docs.gtk.org/glib/types.html#gint64) offset)
```

Set the running time offset of a event. See `gst_event_get_running_time_offset` ([gststreamer/gstevent.html#gst_event_get_running_time_offset](#)) for more information.

MT safe.

Parameters:

event – A GstEvent ([gststreamer/gstevent.html#GstEvent](#)).

offset – A the new running time offset

Since : 1.4

gst_event_set_seek_trickmode_interval

```
gst_event_set_seek_trickmode_interval (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                                       GstClockTime (gststreamer/gstclock.html#GstClockTime) interval)
```

Sets a trickmode interval on a (writable) seek event. Elements that support TRICKMODE_KEY_UNITS seeks SHOULD use this as the minimal interval between each frame they may output.

Parameters:

event – No description available

interval – No description available

Since : 1.16

gst_event_set_seqnum

```
gst_event_set_seqnum (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                      guint32 (https://docs.gtk.org/glib/types.html#guint32) seqnum)
```

Set the sequence number of a event.

This function might be called by the creator of a event to indicate that the event relates to other events or messages. See `gst_event_get_seqnum` ([gststreamer/gstevent.html#gst_event_get_seqnum](#)) for more information.

MT safe.

Parameters:

event – A GstEvent ([gststreamer/gstevent.html#GstEvent](#)).

seqnum – A sequence number.

gst_event_set_stream

```
gst_event_set_stream (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                     GstStream (gststreamer/gststreams.html#GstStream) * stream)
```

Set the *stream* on the stream-start *event*

Parameters:

event – a stream-start event

stream ([transfer: none (Don't free data after the code is done)]) – the stream object to set

Since : 1.10

gst_event_set_stream_flags

```
gst_event_set_stream_flags (GstEvent (gststreamer/gstevent.html#GstEvent) * event,
                           GstStreamFlags (gststreamer/gstevent.html#GstStreamFlags) flags)
```

Parameters:

event – a stream-start event

flags – the stream flags to set

Since : 1.2

gst_event_unref

```
gst_event_unref (GstEvent (gststreamer/gstevent.html#GstEvent) * event)
```

Decrease the refcount of an event, freeing it if the refcount reaches 0.

Parameters:

event ([transfer: full (Free data after the code is done)]) – the event to refcount

gst_event_writable_structure

```
GstStructure (gststreamer/gststructure.html#GstStructure) *
gst_event_writable_structure (GstEvent (gststreamer/gstevent.html#GstEvent) * event)
```

Get a writable version of the structure.

Parameters:

event – A writable GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)).

Returns ([\[transfer: none \(Don't free data after the code is done\)\]](#)) – The structure of the event. The structure is still owned by the event, which means that you should not free it and that the pointer becomes invalid when you free the event. This function ensures that *event* is writable, and if so, will never return NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>). MT safe.

Functions

gst_event_replace

```
gboolean (https://docs.gtk.org/glib/types.html#gboolean)
gst_event_replace (GstEvent (gstreamer/gstevent.html#GstEvent) ** old_event,
                  GstEvent (gstreamer/gstevent.html#GstEvent) * new_event)
```

Modifies a pointer to a GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) to point to a different GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)). The modification is done atomically (so this is useful for ensuring thread safety in some cases), and the reference counts are updated appropriately (the old event is unrefed, the new one is refed).

Either *new_event* or the GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) pointed to by *old_event* may be NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>).

Parameters:

old_event ([\[inout \(Parameter for input and for returning results\)\]](#)[\[transfer: full \(Free data after the code is done\)\]](#)[\[nullable \(NULL may be passed to the value\)\]](#)) – pointer to a pointer to a GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) to be replaced.

new_event ([\[nullable \(NULL may be passed to the value\)\]](#)[\[transfer: none \(Don't free data after the code is done\)\]](#)) – pointer to a GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) that will replace the event pointed to by *old_event*.

Returns – TRUE (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#TRUE:CAPS>) if *new_event* was different from *old_event*

gst_event_steal

```
GstEvent (gstreamer/gstevent.html#GstEvent) *
gst_event_steal (GstEvent (gstreamer/gstevent.html#GstEvent) ** old_event)
```

Atomically replace the GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) pointed to by *old_event* with NULL (<https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS>) and return the original event.

Parameters:

old_event ([inout (Parameter for input and for returning results)][transfer: full (Free data after the code is done)][nullable (NULL may be passed to the value))] – pointer to a pointer to a GstEvent (gststreamer/gstevent.html#GstEvent) to be stolen.

Returns – the GstEvent (gststreamer/gstevent.html#GstEvent) that was in *old_event*

gst_event_take

```
gboolean (https://docs.gtk.org/glib/types.html#gboolean)
gst_event_take (GstEvent (gststreamer/gstevent.html#GstEvent) ** old_event,
               GstEvent (gststreamer/gstevent.html#GstEvent) * new_event)
```

Modifies a pointer to a GstEvent (gststreamer/gstevent.html#GstEvent) to point to a different GstEvent (gststreamer/gstevent.html#GstEvent). This function is similar to *gst_event_replace* (gststreamer/gstevent.html#gst_event_replace) except that it takes ownership of *new_event*.

Either *new_event* or the GstEvent (gststreamer/gstevent.html#GstEvent) pointed to by *old_event* may be NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS).

Parameters:

old_event ([inout (Parameter for input and for returning results)][transfer: full (Free data after the code is done)][nullable (NULL may be passed to the value))] – pointer to a pointer to a GstEvent (gststreamer/gstevent.html#GstEvent) to be stolen.

new_event ([nullable (NULL may be passed to the value)][transfer: full (Free data after the code is done))] – pointer to a GstEvent (gststreamer/gstevent.html#GstEvent) that will replace the event pointed to by *old_event*.

Returns – TRUE (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#TRUE:CAPS) if *new_event* was different from *old_event*

Functions

gst_clear_event

```
gst_clear_event (GstEvent (gststreamer/gstevent.html#GstEvent) ** event_ptr)
```

Clears a reference to a GstEvent (gststreamer/gstevent.html#GstEvent).

event_ptr must not be NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS).

If the reference is NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS) then this function does nothing. Otherwise, the reference count of the event is decreased and the pointer is set to NULL (https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Standard-Macros.html#NULL:CAPS).

Parameters:

event_ptr – a pointer to a GstEvent ([gstreamer/gstevent.html#GstEvent](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEvent)) reference

Since : 1.16

gst_event_type_get_flags

```
GstEventTypeFlags (gstreamer/gstevent.html#GstEventTypeFlags)  
gst_event_type_get_flags (GstEventType (gstreamer/gstevent.html#GstEventType) type)
```

Gets the GstEventTypeFlags ([gstreamer/gstevent.html#GstEventTypeFlags](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEventTypeFlags)) associated with *type*.

Parameters:

type – a GstEventType ([gstreamer/gstevent.html#GstEventType](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEventType))

Returns – a GstEventTypeFlags ([gstreamer/gstevent.html#GstEventTypeFlags](https://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEventTypeFlags)).

gst_event_type_get_name

```
const gchar (https://docs.gtk.org/glib/types.html#gchar) *  
gst_event_type_get_name (GstEventType (gstreamer/gstevent.html#GstEventType) type)
```

Get a printable name for the given event type. Do not modify or free.

Parameters:

type – the event type

Returns – a reference to the static name of the event.

gst_event_type_to_quark

```
GQuark (https://docs.gtk.org/glib/alias.Quark.html)  
gst_event_type_to_quark (GstEventType (gstreamer/gstevent.html#GstEventType) type)
```

Get the unique quark for the given event type.

Parameters:

type – the event type

Returns – the quark associated with the event type

gst_event_type_to_sticky_ordering

```
guint (https://docs.gtk.org/glib/types.html#guint)
gst_event_type_to_sticky_ordering (GstEventType (gststreamer/gstevent.html#GstEventType) type)
```

Converts the GstEventType (gststreamer/gstevent.html#GstEventType) to an unsigned integer that represents the ordering of sticky events when re-sending them. A lower value represents a higher-priority event.

Parameters:

type – a GstEventType (gststreamer/gstevent.html#GstEventType)

Returns – an unsigned integer

Since : 1.22

Function Macros

GST_EVENT_CAST

```
#define GST_EVENT_CAST(obj)          ((GstEvent *) (obj))
```

GST_EVENT_IS_DOWNSTREAM

```
#define GST_EVENT_IS_DOWNSTREAM(ev)  (!((GST_EVENT_TYPE (ev) & GST_EVENT_TYPE_DOWNSTREAM))
```

Check if an event can travel downstream.

Parameters:

ev – the event to query

GST_EVENT_IS_SERIALIZED

```
#define GST_EVENT_IS_SERIALIZED(ev)  (!((GST_EVENT_TYPE (ev) & GST_EVENT_TYPE_SERIALIZED))
```

Check if an event is serialized with the data stream.

Parameters:

ev – the event to query

GST_EVENT_IS_STICKY

```
#define GST_EVENT_IS_STICKY(ev)    (!((GST_EVENT_TYPE (ev) & GST_EVENT_TYPE_STICKY))
```

Check if an event is sticky on the pads.

Parameters:

ev – the event to query

GST_EVENT_IS_UPSTREAM

```
#define GST_EVENT_IS_UPSTREAM(ev)    (!((GST_EVENT_TYPE (ev) & GST_EVENT_TYPE_UPSTREAM))
```

Check if an event can travel upstream.

Parameters:

ev – the event to query

GST_EVENT_MAKE_TYPE

```
#define GST_EVENT_MAKE_TYPE(num,flags) \
    (((num) << GST_EVENT_NUM_SHIFT) | (flags))
```

when making custom event types, use this macro with the num and the given flags

Parameters:

num – the event number to create

flags – the event flags

GST_EVENT_SEQNUM

```
#define GST_EVENT_SEQNUM(event)    (GST_EVENT_CAST(event)->seqnum)
```

The sequence number of *event*.

Parameters:

event – the event to query

GST_EVENT_TIMESTAMP

```
#define GST_EVENT_TIMESTAMP(event)      (GST_EVENT_CAST(event)->timestamp)
```

Get the GstClockTime (gststreamer/gstclock.html#GstClockTime) timestamp of the event. This is the time when the event was created.

Parameters:

event – the event to query

GST_EVENT_TYPE

```
#define GST_EVENT_TYPE(event)          (GST_EVENT_CAST(event)->type)
```

Get the GstEventType (gststreamer/gstevent.html#GstEventType) of the event.

Parameters:

event – the event to query

GST_EVENT_TYPE_NAME

```
#define GST_EVENT_TYPE_NAME(event)     (gst_event_type_get_name(GST_EVENT_TYPE(event)))
```

Get a constant string representation of the GstEventType (gststreamer/gstevent.html#GstEventType) of the event.

Parameters:

event – the event to query

_FLAG

```
#define _FLAG(name) GST_EVENT_TYPE_##name
```

Enumerations

GstEventType

GstEventType ([gstreamer/gstevent.html#GstEventType](http://gstreamer.freedesktop.org/documentation/gstreamer/gstevent.html#GstEventType)) lists the standard event types that can be sent in a pipeline.

The custom event types can be used for private messages between elements that can't be expressed using normal GStreamer buffer passing semantics. Custom events carry an arbitrary *GstStructure* ([gstreamer/gststructure.html#GstStructure](http://gstreamer.freedesktop.org/documentation/gstreamer/gststructure.html#GstStructure)). Specific custom events are distinguished by the name of the structure.

Members

GST_EVENT_UNKNOWN (0) – unknown event.

GST_EVENT_FLUSH_START (2563) – Start a flush operation. This event clears all data from the pipeline and unblock all streaming threads.

GST_EVENT_FLUSH_STOP (5127) – Stop a flush operation. This event resets the running-time of the pipeline.

GST_EVENT_STREAM_START (10254) – Event to mark the start of a new stream. Sent before any other serialized event and only sent at the start of a new stream, not after flushing seeks.

GST_EVENT_CAPS (12814) – *GstCaps* ([gstreamer/gstcaps.html#GstCaps](http://gstreamer.freedesktop.org/documentation/gstreamer/gstcaps.html#GstCaps)) event. Notify the pad of a new media type.

GST_EVENT_SEGMENT (17934) – A new media segment follows in the dataflow. The segment events contains information for clipping buffers and converting buffer timestamps to running-time and stream-time.

GST_EVENT_STREAM_COLLECTION (19230) – A new *GstStreamCollection* ([gstreamer/gststreamcollection.html#GstStreamCollection](http://gstreamer.freedesktop.org/documentation/gstreamer/gststreamcollection.html#GstStreamCollection)) is available (Since: 1.10)

GST_EVENT_TAG (20510) – A new set of metadata tags has been found in the stream.

GST_EVENT_BUFFERSIZE (23054) – Notification of buffering requirements. Currently not used yet.

GST_EVENT_SINK_MESSAGE (25630) – An event that sinks turn into a message. Used to send messages that should be emitted in sync with rendering.

GST_EVENT_STREAM_GROUP_DONE (26894) – Indicates that there is no more data for the stream group ID in the message. Sent before EOS in some instances and should be handled mostly the same. (Since: 1.10)

GST_EVENT_EOS (28174) – End-Of-Stream. No more data is to be expected to follow without either a *STREAM_START* event, or a *FLUSH_STOP* and a *SEGMENT* event.

GST_EVENT_TOC (30750) – An event which indicates that a new table of contents (TOC) was found or updated.

GST_EVENT_PROTECTION (33310) – An event which indicates that new or updated encryption information has been found in the stream.

GST_EVENT_SEGMENT_DONE (38406) – Marks the end of a segment playback.

GST_EVENT_GAP (40966) – Marks a gap in the datastream.

GST_EVENT_INSTANT_RATE_CHANGE (46090) – Notify downstream that a playback rate override should be applied as soon as possible. (Since: 1.18)

- GST_EVENT_QOS* (48641) – A quality message. Used to indicate to upstream elements that the downstream elements should adjust their processing rate.
- GST_EVENT_SEEK* (51201) – A request for a new playback position and rate.
- GST_EVENT_NAVIGATION* (53761) – Navigation events are usually used for communicating user requests, such as mouse or keyboard movements, to upstream elements.
- GST_EVENT_LATENCY* (56321) – Notification of new latency adjustment. Sinks will use the latency information to adjust their synchronisation.
- GST_EVENT_STEP* (58881) – A request for stepping through the media. Sinks will usually execute the step operation.
- GST_EVENT_RECONFIGURE* (61441) – A request for upstream renegotiating caps and reconfiguring.
- GST_EVENT_TOC_SELECT* (64001) – A request for a new playback position based on TOC entry's UID.
- GST_EVENT_SELECT_STREAMS* (66561) – A request to select one or more streams (Since: 1.10)
- GST_EVENT_INSTANT_RATE_SYNC_TIME* (66817) – Sent by the pipeline to notify elements that handle the instant-rate-change event about the running-time when the rate multiplier should be applied (or was applied). (Since: 1.18)
- GST_EVENT_CUSTOM_UPSTREAM* (69121) – Upstream custom event
- GST_EVENT_CUSTOM_DOWNSTREAM* (71686) – Downstream custom event that travels in the data flow.
- GST_EVENT_CUSTOM_DOWNSTREAM_OOB* (74242) – Custom out-of-band downstream event.
- GST_EVENT_CUSTOM_DOWNSTREAM_STICKY* (76830) – Custom sticky downstream event.
- GST_EVENT_CUSTOM_BOTH* (79367) – Custom upstream or downstream event. In-band when travelling downstream.
- GST_EVENT_CUSTOM_BOTH_OOB* (81923) – Custom upstream or downstream out-of-band event.

GstEventTypeFlags

GstEventTypeFlags (gststreamer/gstevent.html#GstEventTypeFlags) indicate the aspects of the different GstEventType (gststreamer/gstevent.html#GstEventType) values. You can get the type flags of a GstEventType (gststreamer/gstevent.html#GstEventType) with the `gst_event_type_get_flags` (gststreamer/gstevent.html#gst_event_type_get_flags) function.

Members

- GST_EVENT_TYPE_UPSTREAM* (1) – Set if the event can travel upstream.
- GST_EVENT_TYPE_DOWNSTREAM* (2) – Set if the event can travel downstream.
- GST_EVENT_TYPE_SERIALIZED* (4) – Set if the event should be serialized with data flow.
- GST_EVENT_TYPE_STICKY* (8) – Set if the event is sticky on the pads.
- GST_EVENT_TYPE_STICKY_MULTIPLE* (16) – Multiple sticky events can be on a pad, each identified by the event name.

GstGapFlags

The different flags that can be set on GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP) events. See `gst_event_set_gap_flags` (gststreamer/gstevent.html#gst_event_set_gap_flags) for details.

Members

GST_GAP_FLAG_MISSING_DATA (1) – The GST_EVENT_GAP (gststreamer/gstevent.html#GST_EVENT_GAP) signals missing data, for example because of packet loss.

Since : 1.20

GstQOSType

The different types of QoS events that can be given to the `gst_event_new_qos` (gststreamer/gstevent.html#gst_event_new_qos) method.

Members

GST_QOS_TYPE_OVERFLOW (0) – The QoS event type that is produced when upstream elements are producing data too quickly and the element can't keep up processing the data. Upstream should reduce their production rate. This type is also used when buffers arrive early or in time.

GST_QOS_TYPE_UNDERFLOW (1) – The QoS event type that is produced when upstream elements are producing data too slowly and need to speed up their production rate.

GST_QOS_TYPE_THROTTLE (2) – The QoS event type that is produced when the application enabled throttling to limit the data rate.

GstStreamFlags

Members

GST_STREAM_FLAG_NONE (0) – This stream has no special attributes

GST_STREAM_FLAG_SPARSE (1) – This stream is a sparse stream (e.g. a subtitle stream), data may flow only in irregular intervals with large gaps in between.

GST_STREAM_FLAG_SELECT (2) – This stream should be selected by default. This flag may be used by demuxers to signal that a stream should be selected by default in a playback scenario.

GST_STREAM_FLAG_UNSELECT (4) – This stream should not be selected by default. This flag may be used by demuxers to signal that a stream should not be selected by default in a playback scenario, but only if explicitly selected by the user (e.g. an audio track for the hard of hearing or a director's commentary track).

Since : 1.2

Constants

GST_EVENT_NUM_SHIFT

```
#define GST_EVENT_NUM_SHIFT      (8)
```

GST_EVENT_TYPE_BOTH

```
#define GST_EVENT_TYPE_BOTH \
    ((GstEventTypeFlags)(GST_EVENT_TYPE_UPSTREAM | GST_EVENT_TYPE_DOWNSTREAM))
```

The same thing as `GST_EVENT_TYPE_UPSTREAM`

([gstreamer/gstevent.html#GST_EVENT_TYPE_UPSTREAM](#)) | `GST_EVENT_TYPE_DOWNSTREAM`

([gstreamer/gstevent.html#GST_EVENT_TYPE_DOWNSTREAM](#)).
