Guile-GNOME: GStreamer

version 0.9.92, updated 10 November 2007

Wim Taymans
many others

Short Contents

1 Overview

The GStreamer wrapper for Guile is a part of Guile-GNOME. Maybe write more here at some point.

2 GstBin

Base class and element that can contain other elements

2.1 Overview

`<gst-bin>` is an element that can contain other `<gst-element>`, allowing them to be managed as a group. Pads from the child elements can be ghosted to the bin, see `<gst-ghost-pad>`. This makes the bin look like any other elements and enables creation of higher-level abstraction elements.

A new `<gst-bin>` is created with `gst-bin-new`. Use a `<gst-pipeline>` instead if you want to create a toplevel bin because a normal bin doesn't have a bus or handle clock distribution of its own.

After the bin has been created you will typically add elements to it with `gst-bin-add`. You can remove elements with `gst-bin-remove`.

An element can be retrieved from a bin with `gst-bin-get-by-name`, using the elements name. `gst-bin-get-by-name-recurse-up` is mainly used for internal purposes and will query the parent bins when the element is not found in the current bin.

An iterator of elements in a bin can be retrieved with `gst-bin-iterate-elements`. Various other iterators exist to retrieve the elements in a bin.

`gst-object-unref` is used to drop your reference to the bin.

The element-added signal is fired whenever a new element is added to the bin. Likewise the element-removed signal is fired whenever an element is removed from the bin.

2.2 Notes

A `<gst-bin>` internally intercepts every `<gst-message>` posted by its children and implements the following default behaviour for each of them:

GST_MESSAGE_SEGMENT_START
GST_MESSAGE_SEGMENT_DONE
GST_MESSAGE_DURATION
GST_MESSAGE_CLOCK_LOST
GST_MESSAGE_CLOCK_PROVIDE
OTHERS

This message is only posted by sinks in the PLAYING state. If all sinks posted the EOS message, this bin will post and EOS message upwards.

just collected and never forwarded upwards. The messages are used to decide when all elements have completed playback of their segment.

Is posted by `<gst-bin>` when all elements that posted a SEGMENT_START have posted a SEGMENT_DONE.

Is posted by an element that detected a change in the stream duration. The default bin behaviour is to clear any cached duration values so that the next duration query will perform a full duration recalculation. The duration change is posted to the application so that it can refetch the new duration with a duration query.

This message is posted by an element when it can no longer provide a clock. The default bin behaviour is to check if the lost clock was the one provided by the bin. If so and the bin is currently in the PLAYING state, the message is forwarded to the bin parent. This message is also generated when a clock provider is removed from the bin. If this message is received by the application, it should PAUSE the pipeline and set it back to PLAYING to force a new clock distribution.

This message is generated when an element can provide a clock. This mostly happens when a new clock provider is added to the bin. The default behaviour of the bin is to mark the currently selected clock as dirty, which will perform a clock recalculation the next time the bin is asked to provide a clock. This message is never sent tot the application but is forwarded to the parent of the bin.

posted upwards.

A `<gst-bin>` implements the following default behaviour for answering to a `<gst-query:>`

GST_QUERY_POSITION
OTHERS

If the query has been asked before with the same format and the bin is a toplevel bin (ie. has no parent), use the cached previous value. If no previous value was cached, the query is sent to all sink elements in the bin and the MAXIMUM of all values is returned. If the bin is a toplevel bin the value is cached. If no sinks are available in the bin, the query fails.

The query is sent to all sink elements in the bin and the MAXIMUM of all values is returned. If no sinks are available in the bin, the query fails.

the query is forwarded to all sink elements, the result of the first sink that answers the query successfully is returned. If no sink is in the bin, the query fails.

A `<gst-bin>` will by default forward any event sent to it to all sink elements. If all the sinks return TRUE, the bin will also return TRUE, else FALSE is returned. If no sinks are in the bin, the event handler will return TRUE.

Last reviewed on 2006-04-28 (0.10.6)

2.3 Usage

`<gst-bin>` [Class]

This `<gobject>` class defines the following properties:

`async-handling`
The bin will handle Asynchronous state changes

`element-added` (*arg0* `<gst-element>`) [Signal on `<gst-bin>`]

Will be emitted after the element was added to the bin.

`element-removed` (*arg0* `<gst-element>`) [Signal on `<gst-bin>`]

Will be emitted after the element was removed from the bin.

`gst-bin-new` (*name* `mchars`) ⇒ (*ret* `<gst-element>`) [Function]

Creates a new bin with the given name.

name the name of the new bin

ret	a new `<gst-bin>`

`gst-bin-add` (*self* `<gst-bin>`) (*element* `<gst-element>`) ⇒ (*ret* `bool`) [Function]
`add` [Method]

Adds the given element to the bin. Sets the element's parent, and thus takes ownership of the element. An element can only be added to one bin.

If the element's pads are linked to other pads, the pads will be unlinked before the element is added to the bin.

MT safe.

bin	a `<gst-bin>`
element	the `<gst-element>` to add
ret	TRUE if the element could be added, FALSE if the bin does not want to accept the element.

`gst-bin-remove` (*self* `<gst-bin>`) (*element* `<gst-element>`) [Function]
⇒ (*ret* `bool`)
`remove` [Method]

Removes the element from the bin, unparenting it as well. Unparenting the element means that the element will be dereferenced, so if the bin holds the only reference to the element, the element will be freed in the process of removing it from the bin. If you want the element to still exist after removing, you need to call `gst-object-ref` before removing it from the bin.

If the element's pads are linked to other pads, the pads will be unlinked before the element is removed from the bin.

MT safe.

bin	a `<gst-bin>`
element	the `<gst-element>` to remove
ret	TRUE if the element could be removed, FALSE if the bin does not want to remove the element.

`gst-bin-get-by-name` (*self* `<gst-bin>`) (*name* `mchars`) [Function]
⇒ (*ret* `<gst-element>`)
`get-by-name` [Method]

Gets the element with the given name from a bin. This function recurses into child bins.

Returns NULL if no element with the given name is found in the bin.

MT safe. Caller owns returned reference.

bin	a `<gst-bin>`
name	the element name to search for
ret	the `<gst-element>` with the given name, or NULL

`gst-bin-get-by-name-recurse-up` (*self* `<gst-bin>`) (*name* `mchars`) [Function]
⇒ (*ret* `<gst-element>`)

`get-by-name-recurse-up` [Method]

Gets the element with the given name from this bin. If the element is not found, a recursion is performed on the parent bin.

Returns NULL if: - no element with the given name is found in the bin

MT safe. Caller owns returned reference.

bin	a `<gst-bin>`
name	the element name to search for
ret	the `<gst-element>` with the given name, or NULL

`gst-bin-get-by-interface` (*self* `<gst-bin>`) (*interface* `<gtype>`) [Function]
⇒ (*ret* `<gst-element>`)

`get-by-interface` [Method]

Looks for an element inside the bin that implements the given interface. If such an element is found, it returns the element. You can cast this element to the given interface afterwards. If you want all elements that implement the interface, use `gst-bin-iterate-all-by-interface`. This function recurses into child bins.

MT safe. Caller owns returned reference.

bin	a `<gst-bin>`
iface	the `<g-type>` of an interface
ret	A `<gst-element>` inside the bin implementing the interface

`gst-bin-iterate-elements` (*self* `<gst-bin>`) [Function]
⇒ (*ret* `<gst-iterator*>`)

`iterate-elements` [Method]

Gets an iterator for the elements in this bin.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
ret	a `<gst-iterator>` of `<gst-element>`, or NULL

`gst-bin-iterate-recurse` (*self* `<gst-bin>`) [Function]
⇒ (*ret* `<gst-iterator*>`)

`iterate-recurse` [Method]

Gets an iterator for the elements in this bin. This iterator recurses into GstBin children.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
ret	a `<gst-iterator>` of `<gst-element>`, or NULL

`gst-bin-iterate-sinks` (*self* `<gst-bin>`) ⇒ (*ret* `<gst-iterator*>`) [Function]
`iterate-sinks` [Method]

Gets an iterator for all elements in the bin that have the `<gst-element-is-sink>` flag set.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
ret	a `<gst-iterator>` of `<gst-element>`, or NULL

`gst-bin-iterate-sorted` (*self* `<gst-bin>`) ⇒ (*ret* `<gst-iterator*>`) [Function]
`iterate-sorted` [Method]

Gets an iterator for the elements in this bin in topologically sorted order. This means that the elements are returned from the most downstream elements (sinks) to the sources.

This function is used internally to perform the state changes of the bin elements.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
ret	a `<gst-iterator>` of `<gst-element>`, or NULL

`gst-bin-iterate-sources` (*self* `<gst-bin>`) ⇒ (*ret* `<gst-iterator*>`) [Function]
`iterate-sources` [Method]

Gets an iterator for all elements in the bin that have no sinkpads and have the `<gst-element-is-sink>` flag unset.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
ret	a `<gst-iterator>` of `<gst-element>`, or NULL

`gst-bin-iterate-all-by-interface` (*self* `<gst-bin>`) (*interface* `<gtype>`) ⇒ (*ret* `<gst-iterator*>`) [Function]
`iterate-all-by-interface` [Method]

Looks for all elements inside the bin that implements the given interface. You can safely cast all returned elements to the given interface. The function recurses inside child bins. The iterator will yield a series of `<gst-element>` that should be unreffed after use.

Each element yielded by the iterator will have its refcount increased, so unref after use.

MT safe. Caller owns returned value.

bin	a `<gst-bin>`
iface	the `<g-type>` of an interface
ret	a `<gst-iterator>` of `<gst-element>` for all elements in the bin implementing the given interface, or NULL

`gst-bin-find-unconnected-pad` (*self* `<gst-bin>`) (*direction* `<gst-pad-direction>`) ⇒ (*ret* `<gst-pad>`) [Function]

`find-unconnected-pad` [Method]

Recursively looks for elements with an unconnected pad of the given direction within the specified bin and returns an unconnected pad if one is found, or NULL otherwise. If a pad is found, the caller owns a reference to it and should use `gst-object-unref` on the pad when it is not needed any longer.

bin	bin in which to look for elements with unconnected pads
direction	whether to look for an unconnected source or sink pad
ret	unconnected pad of the given direction, or NULL.

Since 0.10.3

3 GstBuffer

Data-passing buffer type, supporting sub-buffers.

3.1 Overview

Buffers are the basic unit of data transfer in GStreamer. The `<gst-buffer>` type provides all the state necessary to define a region of memory as part of a stream. Sub-buffers are also supported, allowing a smaller region of a buffer to become its own buffer, with mechanisms in place to ensure that neither memory space goes away prematurely.

Buffers are usually created with `gst-buffer-new`. After a buffer has been created one will typically allocate memory for it and set the size of the buffer data. The following example creates a buffer that can hold a given video frame with a given width, height and bits per plane.

```
GstBuffer *buffer;
gint size, width, height, bpp;
...
size = width * height * bpp;
buffer = gst_buffer_new ();
GST_BUFFER_SIZE (buffer) = size;
GST_BUFFER_MALLOCDATA (buffer) = g_malloc (size);
GST_BUFFER_DATA (buffer) = GST_BUFFER_MALLOCDATA (buffer);
...
```

Alternatively, use `gst-buffer-new-and-alloc` to create a buffer with preallocated data of a given size.

The data pointed to by the buffer can be retrieved with the `gst-buffer-data` macro. The size of the data can be found with `gst-buffer-size`. For buffers of size 0, the data pointer is undefined (usually NULL) and should never be used.

If an element knows what pad you will push the buffer out on, it should use `gst-pad-alloc-buffer` instead to create a buffer. This allows downstream elements to provide special buffers to write in, like hardware buffers.

A buffer has a pointer to a `<gst-caps>` describing the media type of the data in the buffer. Attach caps to the buffer with `gst-buffer-set-caps`; this is typically done before pushing out a buffer using `gst-pad-push` so that the downstream element knows the type of the buffer.

A buffer will usually have a timestamp, and a duration, but neither of these are guaranteed (they may be set to `<gst-clock-time-none>`). Whenever a meaningful value can be given for these, they should be set. The timestamp and duration are measured in nanoseconds (they are `<gst-clock-time>` values).

A buffer can also have one or both of a start and an end offset. These are media-type specific. For video buffers, the start offset will generally be the frame number. For audio buffers, it will be the number of samples produced so far. For compressed data, it could be the byte offset in a source or destination file. Likewise, the end offset will be the offset of the end of the buffer. These can only be meaningfully interpreted if you know the media type of the buffer (the `<gst-caps>` set on it). Either or both can be set to `<gst-buffer-offset-none>`.

`gst-buffer-ref` is used to increase the refcount of a buffer. This must be done when you want to keep a handle to the buffer after pushing it to the next element.

To efficiently create a smaller buffer out of an existing one, you can use `gst-buffer-create-sub`.

If a plug-in wants to modify the buffer data in-place, it should first obtain a buffer that is safe to modify by using `gst-buffer-make-writable`. This function is optimized so that a copy will only be made when it is necessary.

A plugin that only wishes to modify the metadata of a buffer, such as the offset, timestamp or caps, should use `gst-buffer-make-metadata-writable`, which will create a subbuffer of the original buffer to ensure the caller has sole ownership, and not copy the buffer data.

Several flags of the buffer can be set and unset with the `gst-buffer-flag-set` and `gst-buffer-flag-unset` macros. Use `gst-buffer-flag-is-set` to test if a certain `<gst-buffer-flag>` is set.

Buffers can be efficiently merged into a larger buffer with `gst-buffer-merge` and `gst-buffer-span` if the `gst-buffer-is-span-fast` function returns TRUE.

An element should either unref the buffer or push it out on a src pad using `gst-pad-push` (see `<gst-pad>`).

Buffers are usually freed by unreffing them with `gst-buffer-unref`. When the refcount drops to 0, any data pointed to by `gst-buffer-mallocdata` will also be freed.

Last reviewed on August 11th, 2006 (0.10.10)

3.2 Usage

`<gst-buffer>` [Class]

`gst-buffer-new` ⇒ (*ret* `<gst-buffer>`) [Function]

Creates a newly allocated buffer without any data.

MT safe.

ret the new `<gst-buffer>`.

`gst-buffer-make-metadata-writable` (*self* `<gst-buffer>`) [Function]
⇒ (*ret* `<gst-buffer>`)

`make-metadata-writable` [Method]

Similar to gst_buffer_make_writable, but does not ensure that the buffer data array is writable. Instead, this just ensures that the returned buffer is solely owned by the caller, by creating a subbuffer of the original buffer if necessary.

After calling this function, *buf* should not be referenced anymore. The result of this function has guaranteed writable metadata.

buf a `<gst-buffer>`

ret A new `<gst-buffer>` with writable metadata.

`gst-buffer-get-caps` (*self* `<gst-buffer>`) ⇒ (*ret* `<gst-caps>`) [Function]
`get-caps` [Method]

Gets the media type of the buffer. This can be NULL if there is no media type attached to this buffer.

Returns: a reference to the `<gst-caps>`. unref after usage.

buffer	a `<gst-buffer>`.
ret	NULL if there were no caps on this buffer.

`gst-buffer-set-caps` (*self* `<gst-buffer>`) (*caps* `<gst-caps>`) [Function]
`set-caps` [Method]

Sets the media type on the buffer. The refcount of the caps will be increased and any previous caps on the buffer will be unreffed.

buffer	a `<gst-buffer>`.
caps	a `<gst-caps>`.

`gst-buffer-create-sub` (*self* `<gst-buffer>`) (*offset* `unsigned-int`) (*size* `unsigned-int`) ⇒ (*ret* `<gst-buffer>`) [Function]
`create-sub` [Method]

Creates a sub-buffer from *parent* at *offset* and *size*. This sub-buffer uses the actual memory space of the parent buffer. This function will copy the offset and timestamp fields when the offset is 0. If not, they will be set to `<gst-clock-time-none>` and `<gst-buffer-offset-none>`. If *offset* equals 0 and *size* equals the total size of *buffer*, the duration and offset end fields are also copied. If not they will be set to `<gst-clock-time-none>` and `<gst-buffer-offset-none>`.

MT safe. Returns: the new `<gst-buffer>`.

parent	a `<gst-buffer>`.
offset	the offset into parent `<gst-buffer>` at which the new sub-buffer begins.
size	the size of the new `<gst-buffer>` sub-buffer, in bytes.
ret	NULL if the arguments were invalid.

`gst-buffer-is-span-fast` (*self* `<gst-buffer>`) (*buf2* `<gst-buffer>`) ⇒ (*ret* `bool`) [Function]
`is-span-fast` [Method]

Determines whether a `gst-buffer-span` can be done without copying the contents, that is, whether the data areas are contiguous sub-buffers of the same buffer.

MT safe.

buf1	the first `<gst-buffer>`.
buf2	the second `<gst-buffer>`.
ret	TRUE if the buffers are contiguous, FALSE if a copy would be required.

`gst-buffer-span` (*self* `<gst-buffer>`) (*offset* `unsigned-int32`) (*buf2* `<gst-buffer>`) (*len* `unsigned-int32`) ⇒ (*ret* `<gst-buffer>`) [Function]

`span` [Method]

Creates a new buffer that consists of part of buf1 and buf2. Logically, buf1 and buf2 are concatenated into a single larger buffer, and a new buffer is created at the given offset inside this space, with a given length.

If the two source buffers are children of the same larger buffer, and are contiguous, the new buffer will be a child of the shared parent, and thus no copying is necessary. you can use `gst-buffer-is-span-fast` to determine if a memcpy will be needed.

MT safe. Returns: the new `<gst-buffer>` that spans the two source buffers.

buf1	the first source `<gst-buffer>` to merge.
offset	the offset in the first buffer from where the new buffer should start.
buf2	the second source `<gst-buffer>` to merge.
len	the total length of the new buffer.
ret	NULL if the arguments are invalid.

`gst-buffer-stamp` (*self* `<gst-buffer>`) (*src* `<gst-buffer>`) [Function]

`stamp` [Method]

'`gst_buffer_stamp`' is deprecated and should not be used in newly-written code. use `gst-buffer-copy-metadata` instead, it provides more control.

Copies additional information (the timestamp, duration, and offset start and end) from one buffer to the other.

This function does not copy any buffer flags or caps and is equivalent to gst_buffer_copy_metadata(*dest*, *src*, GST_BUFFER_COPY_TIMESTAMPS).

dest	buffer to stamp
src	buffer to stamp from

`gst-buffer-join` (*self* `<gst-buffer>`) (*buf2* `<gst-buffer>`) ⇒ (*ret* `<gst-buffer>`) [Function]

`join` [Method]

Create a new buffer that is the concatenation of the two source buffers, and unrefs the original source buffers.

If the buffers point to contiguous areas of memory, the buffer is created without copying the data.

buf1	the first source `<gst-buffer>`.
buf2	the second source `<gst-buffer>`.
ret	the new `<gst-buffer>` which is the concatenation of the source buffers.

`gst-buffer-merge` (*self* `<gst-buffer>`) (*buf2* `<gst-buffer>`) ⇒ (*ret* `<gst-buffer>`) [Function]

`merge` [Method]

Create a new buffer that is the concatenation of the two source buffers. The original source buffers will not be modified or unref'd. Make sure you unref the source buffers if they are not used anymore afterwards.

If the buffers point to contiguous areas of memory, the buffer is created without copying the data.

buf1	the first source `<gst-buffer>` to merge.
buf2	the second source `<gst-buffer>` to merge.
ret	the new `<gst-buffer>` which is the concatenation of the source buffers.

4 GstBus

Asynchronous message bus subsystem

4.1 Overview

The `<gst-bus>` is an object responsible for delivering `<gst-messages>` in a first-in first-out way from the streaming threads to the application.

Since the application typically only wants to deal with delivery of these messages from one thread, the GstBus will marshall the messages between different threads. This is important since the actual streaming of media is done in another thread than the application.

The GstBus provides support for `<g-source>` based notifications. This makes it possible to handle the delivery in the glib mainloop.

The `<g-source>` callback function `gst-bus-async-signal-func` can be used to convert all bus messages into signal emissions.

A message is posted on the bus with the `gst-bus-post` method. With the `gst-bus-peek` and `gst-bus-pop` methods one can look at or retrieve a previously posted message.

The bus can be polled with the `gst-bus-poll` method. This methods blocks up to the specified timeout value until one of the specified messages types is posted on the bus. The application can then `-pop` the messages from the bus to handle them. Alternatively the application can register an asynchronous bus function using `gst-bus-add-watch-full` or `gst-bus-add-watch`. This function will install a `<g-source>` in the default glib main loop and will deliver messages a short while after they have been posted. Note that the main loop should be running for the asynchronous callbacks.

It is also possible to get messages from the bus without any thread marshalling with the `gst-bus-set-sync-handler` method. This makes it possible to react to a message in the same thread that posted the message on the bus. This should only be used if the application is able to deal with messages from different threads.

Every `<gst-pipeline>` has one bus.

Note that a `<gst-pipeline>` will set its bus into flushing state when changing from READY to NULL state.

Last reviewed on 2006-03-12 (0.10.5)

4.2 Usage

`<gst-bus>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`sync-message` (*arg0* `<gst-message>`) [Signal on `<gst-bus>`]

A message has been posted on the bus. This signal is emitted from the thread that posted the message so one has to be careful with locking.

This signal will not be emitted by default, you have to set up `gst-bus-sync-signal-handler` as a sync handler if you want this signal to be emitted when a message is posted on the bus, like this:

```
          gst_bus_set_sync_handler (bus, gst_bus_sync_signal_handler, yourdata);
```

`message` (*arg0* `<gst-message>`) [Signal on `<gst-bus>`]

A message has been posted on the bus. This signal is emitted from a GSource added to the mainloop. this signal will only be emitted when there is a mainloop running.

`gst-bus-new` ⇒ (*ret* `<gst-bus>`) [Function]

Creates a new `<gst-bus>` instance.

ret	a new `<gst-bus>` instance

`gst-bus-post` (*self* `<gst-bus>`) (*message* `<gst-message>`) ⇒ (*ret* `bool`) [Function]

`post` [Method]

Post a message on the given bus. Ownership of the message is taken by the bus.

bus	a `<gst-bus>` to post on
message	The `<gst-message>` to post
ret	TRUE if the message could be posted, FALSE if the bus is flushing. MT safe.

`gst-bus-have-pending` (*self* `<gst-bus>`) ⇒ (*ret* `bool`) [Function]

`have-pending` [Method]

Check if there are pending messages on the bus that should be handled.

bus	a `<gst-bus>` to check
ret	TRUE if there are messages on the bus to be handled, FALSE otherwise. MT safe.

`gst-bus-peek` (*self* `<gst-bus>`) ⇒ (*ret* `<gst-message>`) [Function]

`peek` [Method]

Peek the message on the top of the bus' queue. The message will remain on the bus' message queue. A reference is returned, and needs to be unreffed by the caller.

bus	a `<gst-bus>`
ret	The `<gst-message>` that is on the bus, or NULL if the bus is empty. MT safe.

`gst-bus-pop` (*self* `<gst-bus>`) ⇒ (*ret* `<gst-message>`) [Function]

`pop` [Method]

Get a message from the bus.

bus	a `<gst-bus>` to pop
ret	The `<gst-message>` that is on the bus, or NULL if the bus is empty. The message is taken from the bus and needs to be unreffed with `gst-message-unref` after usage. MT safe.

`gst-bus-set-flushing` (*self* `<gst-bus>`) (*flushing* `bool`) [Function]
`set-flushing` [Method]

If *flushing*, flush out and unref any messages queued in the bus. Releases references to the message origin objects. Will flush future messages until `gst-bus-set-flushing` sets *flushing* to `#f`.

MT safe.

bus	a `<gst-bus>`
flushing	whether or not to flush the bus

`gst-bus-set-sync-handler` (*self* `<gst-bus>`) (*func* `<gst-bus-sync-handler>`) (*data* `<gpointer>`) [Function]
`set-sync-handler` [Method]

Sets the synchronous handler on the bus. The function will be called every time a new message is posted on the bus. Note that the function will be called in the same thread context as the posting object. This function is usually only called by the creator of the bus. Applications should handle messages asynchronously using the gst_bus watch and poll functions.

You cannot replace an existing sync_handler. You can pass NULL to this function, which will clear the existing handler.

bus	a `<gst-bus>` to install the handler on
func	The handler function to install
data	User data that will be sent to the handler function.

`gst-bus-sync-signal-handler` (*self* `<gst-bus>`) (*message* `<gst-message>`) (*data* `<gpointer>`) ⇒ (*ret* `<gst-bus-sync-reply>`) [Function]
`sync-signal-handler` [Method]

A helper GstBusSyncHandler that can be used to convert all synchronous messages into signals.

bus	a `<gst-bus>`
message	the `<gst-message>` received
data	user data
ret	GST_BUS_PASS

`gst-bus-create-watch` (*self* `<gst-bus>`) ⇒ (*ret* `<g-source*>`) [Function]
`create-watch` [Method]

Create watch for this bus. The GSource will be dispatched whenever a message is on the bus. After the GSource is dispatched, the message is popped off the bus and unreffed.

bus	a `<gst-bus>` to create the watch for
ret	A `<g-source>` that can be added to a mainloop.

`gst-bus-add-watch-full` (*self* `<gst-bus>`) (*priority* `int`) (*func* `<gst-bus-func>`) (*user_data* `<gpointer>`) (*notify* `<g-destroy-notify>`) ⇒ (*ret* `unsigned-int`) [Function]
`add-watch-full` [Method]

Adds a bus watch to the default main context with the given *priority*. This function is used to receive asynchronous messages in the main loop.

When *func* is called, the message belongs to the caller; if you want to keep a copy of it, call `gst-message-ref` before leaving *func*.

The watch can be removed using `g-source-remove` or by returning FALSE from *func*.

bus	a `<gst-bus>` to create the watch for.
priority	The priority of the watch.
func	A function to call when a message is received.
user-data	user data passed to *func*.
notify	the function to call when the source is removed.
ret	The event source id. MT safe.

`gst-bus-add-watch` (*self* `<gst-bus>`) (*func* `<gst-bus-func>`) (*user_data* `<gpointer>`) ⇒ (*ret* `unsigned-int`) [Function]
`add-watch` [Method]

Adds a bus watch to the default main context with the default priority. This function is used to receive asynchronous messages in the main loop.

The watch can be removed using `g-source-remove` or by returning FALSE from *func*.

bus	a `<gst-bus>` to create the watch for
func	A function to call when a message is received.
user-data	user data passed to *func*.
ret	The event source id. MT safe.

`gst-bus-async-signal-func` (*self* `<gst-bus>`) (*message* `<gst-message>`) (*data* `<gpointer>`) ⇒ (*ret* `bool`) [Function]
`async-signal-func` [Method]

A helper `<gst-bus-func>` that can be used to convert all asynchronous messages into signals.

bus	a `<gst-bus>`
message	the `<gst-message>` received
data	user data
ret	TRUE

`gst-bus-add-signal-watch` (*self* `<gst-bus>`) [Function]
`add-signal-watch` [Method]

Adds a bus signal watch to the default main context with the default priority. After calling this statement, the bus will emit the "message" signal for each message posted on the bus.

This function may be called multiple times. To clean up, the caller is responsible for calling `gst-bus-remove-signal-watch` as many times as this function is called.

MT safe.

bus	a `<gst-bus>` on which you want to receive the "message" signal

`gst-bus-add-signal-watch-full` (*self* `<gst-bus>`) (*priority* `int`) [Function]
`add-signal-watch-full` [Method]

Adds a bus signal watch to the default main context with the given priority. After calling this statement, the bus will emit the "message" signal for each message posted on the bus when the main loop is running.

This function may be called multiple times. To clean up, the caller is responsible for calling `gst-bus-remove-signal-watch` as many times as this function is called.

MT safe.

bus	a `<gst-bus>` on which you want to receive the "message" signal
priority	The priority of the watch.

`gst-bus-remove-signal-watch` (*self* `<gst-bus>`) [Function]
`remove-signal-watch` [Method]

Removes a signal watch previously added with `gst-bus-add-signal-watch`.

MT safe.

bus	a `<gst-bus>` you previously added a signal watch to

`gst-bus-poll` (*self* `<gst-bus>`) (*events* `<gst-message-type>`) [Function]
(*timeout* `unsigned-long-long`) ⇒ (*ret* `<gst-message>`)
`poll` [Method]

Poll the bus for messages. Will block while waiting for messages to come. You can specify a maximum time to poll with the *timeout* parameter. If *timeout* is negative, this function will block indefinitely.

All messages not in *events* will be popped off the bus and will be ignored.

Because poll is implemented using the "message" signal enabled by `gst-bus-add-signal-watch`, calling `gst-bus-poll` will cause the "message" signal to be emitted for every message that poll sees. Thus a "message" signal handler will see the same messages that this function sees – neither will steal messages from the other.

This function will run a main loop from the default main context when polling.

bus	a `<gst-bus>`
events	a mask of `<gst-message-type>`, representing the set of message types to poll for.
timeout	the poll timeout, as a `<gst-clock-time-diff>`, or -1 to poll indefinitely.
ret	The message that was received, or NULL if the poll timed out. The message is taken from the bus and needs to be unreffed with `gst-message-unref` after usage.

5 GstCaps

Structure describing sets of media formats

5.1 Overview

Caps (capabilities) are lighweight refcounted objects describing media types. They are composed of an array of `<gst-structure>`.

Caps are exposed on `<gst-pad-template>` to describe all possible types a given pad can handle. They are also stored in the `<gst-registry>` along with a description of the `<gst-element>`.

Caps are exposed on the element pads using the `gst-pad-get-caps` pad function. This function describes the possible types that the pad can handle or produce at runtime.

Caps are also attached to buffers to describe to content of the data pointed to by the buffer with `gst-buffer-set-caps`. Caps attached to a `<gst-buffer>` allow for format negotiation upstream and downstream.

A `<gst-caps>` can be constructed with the following code fragment:

```
GstCaps *caps;
caps = gst_caps_new_simple ("video/x-raw-yuv",
     "format", GST_TYPE_FOURCC, GST_MAKE_FOURCC ('I', '4', '2', '0'),
     "framerate", GST_TYPE_FRACTION, 25, 1,
     "pixel-aspect-ratio", GST_TYPE_FRACTION, 1, 1,
     "width", G_TYPE_INT, 320,
     "height", G_TYPE_INT, 240,
     NULL);
```

A `<gst-caps>` is fixed when it has no properties with ranges or lists. Use `gst-caps-is-fixed` to test for fixed caps. Only fixed caps can be set on a `<gst-pad>` or `<gst-buffer>`.

Various methods exist to work with the media types such as subtracting or intersecting.

Last reviewed on 2007-02-13 (0.10.10)

5.2 Usage

`<gst-caps>` [Class]

`gst-caps-new-empty` ⇒ (*ret* `<gst-caps>`) [Function]

Creates a new `<gst-caps>` that is empty. That is, the returned `<gst-caps>` contains no media formats. Caller is responsible for unreffing the returned caps.

ret	the new `<gst-caps>`

`gst-caps-new-any` ⇒ (*ret* `<gst-caps>`) [Function]

Creates a new `<gst-caps>` that indicates that it is compatible with any media format.

ret	the new `<gst-caps>`

`gst-caps-copy-nth` (*self* `<gst-caps>`) (*nth* `unsigned-int`) [Function]
⇒ (*ret* `<gst-caps>`)

Creates a new `<gst-caps>` and appends a copy of the nth structure contained in *caps*.

caps the `<gst-caps>` to copy

nth the nth structure to copy

ret the new `<gst-caps>`

`gst-static-caps-get` (*self* `<gst-static-caps*>`) [Function]
⇒ (*ret* `<gst-caps>`)

Converts a `<gst-static-caps>` to a `<gst-caps>`.

static-caps
the `<gst-static-caps>` to convert

ret A pointer to the `<gst-caps>`. Unref after usage. Since the core holds an additional ref to the returned caps, use `gst-caps-make-writable` on the returned caps to modify it.

`gst-caps-append` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) [Function]

Appends the structures contained in *caps2* to *caps1*. The structures in *caps2* are not copied – they are transferred to *caps1*, and then *caps2* is freed. If either caps is ANY, the resulting caps will be ANY.

caps1 the `<gst-caps>` that will be appended to

caps2 the `<gst-caps>` to append

`gst-caps-merge` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) [Function]

Appends the structures contained in *caps2* to *caps1* if they are not yet expressed by *caps1*. The structures in *caps2* are not copied – they are transferred to *caps1*, and then *caps2* is freed. If either caps is ANY, the resulting caps will be ANY.

caps1 the `<gst-caps>` that will take the new entries

caps2 the `<gst-caps>` to merge in

Since 0.10.10

`gst-caps-append-structure` (*self* `<gst-caps>`) [Function]
(*structure* `<gst-structure>`)

Appends *structure* to *caps*. The structure is not copied; *caps* becomes the owner of *structure*.

caps the `<gst-caps>` that will be appended to

structure the `<gst-structure>` to append

`gst-caps-remove-structure` (*self* `<gst-caps>`) (*idx* `unsigned-int`) [Function]

removes the stucture with the given index from the list of structures contained in *caps*.

caps the `<gst-caps>` to remove from

idx Index of the structure to remove

gst-caps-merge-structure (*self* `<gst-caps>`) (*structure* `<gst-structure>`) [Function]

Appends *structure* to *caps* if its not already expressed by *caps*. The structure is not copied; *caps* becomes the owner of *structure*.

caps	the `<gst-caps>` that will the the new structure
structure	the `<gst-structure>` to merge

gst-caps-get-size (*self* `<gst-caps>`) ⇒ (*ret* `unsigned-int`) [Function]

Gets the number of structures contained in *caps*.

caps	a `<gst-caps>`
ret	the number of structures that *caps* contains

gst-caps-get-structure (*self* `<gst-caps>`) (*index* `unsigned-int`) ⇒ (*ret* `<gst-structure>`) [Function]

Finds the structure in *caps* that has the index *index*, and returns it.

WARNING: This function takes a const GstCaps *, but returns a non-const GstStructure *. This is for programming convenience – the caller should be aware that structures inside a constant `<gst-caps>` should not be modified.

caps	a `<gst-caps>`
index	the index of the structure
ret	a pointer to the `<gst-structure>` corresponding to *index*

gst-caps-is-any (*self* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Determines if *caps* represents any media format.

caps	the `<gst-caps>` to test
ret	TRUE if *caps* represents any format.

gst-caps-is-empty (*self* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Determines if *caps* represents no media formats.

caps	the `<gst-caps>` to test
ret	TRUE if *caps* represents no formats.

gst-caps-is-fixed (*self* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Fixed `<gst-caps>` describe exactly one format, that is, they have exactly one structure, and each field in the structure describes a fixed type. Examples of non-fixed types are GST_TYPE_INT_RANGE and GST_TYPE_LIST.

caps	the `<gst-caps>` to test
ret	TRUE if *caps* is fixed

gst-caps-is-equal (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Checks if the given caps represent the same set of caps.

> This function does not work reliably if optional properties for caps are included on one caps and omitted on the other.

This function deals correctly with passing NULL for any of the caps.

caps1 a `<gst-caps>`

caps2 another `<gst-caps>`

ret TRUE if both caps are equal.

`gst-caps-is-equal-fixed` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Tests if two `<gst-caps>` are equal. This function only works on fixed `<gst-caps>`.

caps1 the `<gst-caps>` to test

caps2 the `<gst-caps>` to test

ret TRUE if the arguments represent the same format

`gst-caps-is-always-compatible` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

A given `<gst-caps>` structure is always compatible with another if every media format that is in the first is also contained in the second. That is, *caps1* is a subset of *caps2*.

caps1 the `<gst-caps>` to test

caps2 the `<gst-caps>` to test

ret TRUE if *caps1* is a subset of *caps2*.

`gst-caps-is-subset` (*self* `<gst-caps>`) (*superset* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Checks if all caps represented by *subset* are also represented by *superset*.

> This function does not work reliably if optional properties for caps are included on one caps and omitted on the other.

subset a `<gst-caps>`

superset a potentially greater `<gst-caps>`

ret '`#t`' if *subset* is a subset of *superset*

`gst-caps-intersect` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) ⇒ (*ret* `<gst-caps>`) [Function]

Creates a new `<gst-caps>` that contains all the formats that are common to both *caps1* and *caps2*.

caps1 a `<gst-caps>` to intersect

caps2 a `<gst-caps>` to intersect

ret the new `<gst-caps>`

`gst-caps-union` (*self* `<gst-caps>`) (*caps2* `<gst-caps>`) ⇒ (*ret* `<gst-caps>`) [Function]

Creates a new `<gst-caps>` that contains all the formats that are in either *caps1* and *caps2*.

caps1	a `<gst-caps>` to union
caps2	a `<gst-caps>` to union
ret	the new `<gst-caps>`

`gst-caps-normalize` (*self* `<gst-caps>`) ⇒ (*ret* `<gst-caps>`) [Function]

Creates a new `<gst-caps>` that represents the same set of formats as *caps*, but contains no lists. Each list is expanded into separate *gst-structures*.

caps	a `<gst-caps>` to normalize
ret	the new `<gst-caps>`

`gst-caps-do-simplify` (*self* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]

Modifies the given *caps* inplace into a representation that represents the same set of formats, but in a simpler form. Component structures that are identical are merged. Component structures that have values that can be merged are also merged.

caps	a `<gst-caps>` to simplify
ret	TRUE, if the caps could be simplified

`gst-caps-save-thyself` (*self* `<gst-caps>`) (*parent* `<xml-node-ptr>`) ⇒ (*ret* `<xml-node-ptr>`) [Function]

Serializes a `<gst-caps>` to XML and adds it as a child node of *parent*.

caps	a `<gst-caps>` structure
parent	a XML parent node
ret	a XML node pointer

`gst-caps-load-thyself` (*parent* `<xml-node-ptr>`) ⇒ (*ret* `<gst-caps>`) [Function]

Creates a `<gst-caps>` from its XML serialization.

parent	a XML node
ret	a new `<gst-caps>` structure

`gst-caps-replace` (*caps* `<gst-caps**>`) (*newcaps* `<gst-caps>`) [Function]

Replaces *caps with *newcaps*. Unrefs the `<gst-caps>` in the location pointed to by *caps*, if applicable, then modifies *caps* to point to *newcaps*. An additional ref on *newcaps* is taken.

This function does not take any locks so you might want to lock the object owning *caps* pointer.

caps	a pointer to `<gst-caps>`
newcaps	a `<gst-caps>` to replace *caps

`gst-caps-to-string` (*self* `<gst-caps>`) ⇒ (*ret* `mchars`) [Function]

Converts *caps* to a string representation. This string representation can be converted back to a `<gst-caps>` by `gst-caps-from-string`.

For debugging purposes its easier to do something like this: This prints the caps in human readble form.

```
GST_LOG ("caps are %" GST_PTR_FORMAT, caps);
```

caps	a `<gst-caps>`
ret	a newly allocated string representing *caps*.

`gst-caps-from-string` (*string* `mchars`) ⇒ (*ret* `<gst-caps>`) [Function]

Converts *caps* from a string representation.

string	a string to convert to `<gst-caps>`
ret	a newly allocated `<gst-caps>`

`gst-caps-subtract` (*self* `<gst-caps>`) (*subtrahend* `<gst-caps>`) ⇒ (*ret* `<gst-caps>`) [Function]

Subtracts the *subtrahend* from the *minuend*.

This function does not work reliably if optional properties for caps are included on one caps and omitted on the other.

minuend	`<gst-caps>` to substract from
subtrahend	`<gst-caps>` to substract
ret	the resulting caps

`gst-caps-make-writable` (*self* `<gst-caps>`) ⇒ (*ret* `<gst-caps>`) [Function]

Returns a writable copy of *caps*.

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

In short, this function unrefs the caps in the argument and refs the caps that it returns. Don't access the argument after calling this function. See also: `gst-caps-ref`.

caps	the `<gst-caps>` to make writable
ret	the same `<gst-caps>` object.

`gst-caps-truncate` (*self* `<gst-caps>`) [Function]

Destructively discard all but the first structure from *caps*. Useful when fixating. *caps* must be writable.

caps	the `<gst-caps>` to truncate

6 GstChildProxy

Interface for multi child elements.

6.1 Overview

This interface abstracts handling of property sets for child elements. Imagine elements such as mixers or polyphonic generators. They all have multiple `<gst-pad>` or some kind of voice objects. The element acts as a parent for those child objects. Each child has the same properties.

By implementing this interface the child properties can be accessed from the parent element by using `gst-child-proxy-get` and `gst-child-proxy-set`.

Property names are written as "child-name::property-name". The whole naming scheme is recursive. Thus "child1::child2::property" is valid too, if "child1" also implements the `<gst-child-proxy>` interface.

6.2 Usage

`gst-child-proxy-get-children-count` (*self* `<gst-child-proxy*>`) ⇒ (*ret* `unsigned-int`) [Function]

Gets the number of child objects this parent contains.

parent	the parent object
ret	the number of child objects MT safe.

`gst-child-proxy-get-child-by-name` (*self* `<gst-child-proxy*>`) (*name* `mchars`) ⇒ (*ret* `<gst-object>`) [Function]

Looks up a child element by the given name.

Implementors can use `<gst-object>` together with `gst-object-get-name`

parent	the parent object to get the child from
name	the childs name
ret	the child object or '`#f`' if not found. Unref after usage. MT safe.

`gst-child-proxy-get-child-by-index` (*self* `<gst-child-proxy*>`) (*index* `unsigned-int`) ⇒ (*ret* `<gst-object>`) [Function]

Fetches a child by its number.

parent	the parent object to get the child from
index	the childs position in the child list
ret	the child object or '`#f`' if not found (index too high). Unref after usage. MT safe.

`gst-child-proxy-lookup` (*object* `<gst-object>`) (*name* `mchars`) (*target* `<gst-object**>`) (*pspec* `<g-param-spec**>`) ⇒ (*ret* `bool`) [Function]

Looks up which object and `<gparam>` would be effected by the given *name*.

object	object to lookup the property in

name	name of the property to look up
target	pointer to a `<gst-object>` that takes the real object to set property on
pspec	pointer to take the `<gparam>` describing the property
ret	TRUE if *target* and *pspec* could be found. FALSE otherwise. In that case the values for *pspec* and *target* are not modified. Unref *target* after usage. MT safe.

`gst-child-proxy-get-property` (*object* `<gst-object>`) (*name* `mchars`) (*value* `<gvalue>`) [Function]

Gets a single property using the GstChildProxy mechanism. You are responsible for for freeing it by calling `g-value-unset`

object	object to query
name	name of the property
value	a `<gvalue>` that should take the result.

`gst-child-proxy-set-property` (*object* `<gst-object>`) (*name* `mchars`) (*value* `<gvalue>`) [Function]

Sets a single property using the GstChildProxy mechanism.

object	the parent object
name	name of the property to set
value	new `<gvalue>` for the property

`gst-child-proxy-child-added` (*object* `<gst-object>`) (*child* `<gst-object>`) [Function]

Emits the "child-added" signal.

object	the parent object
child	the newly added child

`gst-child-proxy-child-removed` (*object* `<gst-object>`) (*child* `<gst-object>`) [Function]

Emits the "child-removed" signal.

object	the parent object
child	the newly added child

7 GstClock

Abstract class for global clocks

7.1 Overview

GStreamer uses a global clock to synchronize the plugins in a pipeline. Different clock implementations are possible by implementing this abstract base class.

The `<gst-clock>` returns a monotonically increasing time with the method `gst-clock-get-time`. Its accuracy and base time depend on the specific clock implementation but time is always expressed in nanoseconds. Since the baseline of the clock is undefined, the clock time returned is not meaningful in itself, what matters are the deltas between two clock times. The time returned by a clock is called the absolute time.

The pipeline uses the clock to calculate the stream time. Usually all renderers synchronize to the global clock using the buffer timestamps, the newsegment events and the element's base time, see `<gst-pipeline>`.

A clock implementation can support periodic and single shot clock notifications both synchronous and asynchronous.

One first needs to create a `<gst-clock-id>` for the periodic or single shot notification using `gst-clock-new-single-shot-id` or `gst-clock-new-periodic-id`.

To perform a blocking wait for the specific time of the `<gst-clock-id>` use the `gst-clock-id-wait`. To receive a callback when the specific time is reached in the clock use `gst-clock-id-wait-async`. Both these calls can be interrupted with the `gst-clock-id-unschedule` call. If the blocking wait is unscheduled a return value of GST_CLOCK_UNSCHEDULED is returned.

Periodic callbacks scheduled async will be repeadedly called automatically until it is unscheduled. To schedule a sync periodic callback, `gst-clock-id-wait` should be called repeadedly.

The async callbacks can happen from any thread, either provided by the core or from a streaming thread. The application should be prepared for this.

A `<gst-clock-id>` that has been unscheduled cannot be used again for any wait operation, a new `<gst-clock-id>` should be created and the old unscheduled one should be destroyed wirth `gst-clock-id-unref`.

It is possible to perform a blocking wait on the same `<gst-clock-id>` from multiple threads. However, registering the same `<gst-clock-id>` for multiple async notifications is not possible, the callback will only be called for the thread registering the entry last.

None of the wait operations unref the `<gst-clock-id>`, the owner is responsible for unreffing the ids itself. This holds for both periodic and single shot notifications. The reason being that the owner of the `<gst-clock-id>` has to keep a handle to the `<gst-clock-id>` to unblock the wait on FLUSHING events or state changes and if the entry would be unreffed automatically, the handle might become invalid without any notification.

These clock operations do not operate on the stream time, so the callbacks will also occur when not in PLAYING state as if the clock just keeps on running. Some clocks however do not progress when the element that provided the clock is not PLAYING.

When a clock has the GST_CLOCK_FLAG_CAN_SET_MASTER flag set, it can be slaved to another `<gst-clock>` with the `gst-clock-set-master`. The clock will then automatically be synchronized to this master clock by repeadedly sampling the master clock and the slave clock and recalibrating the slave clock with `gst-clock-set-calibration`. This feature is mostly useful for plugins that have an internal clock but must operate with another clock selected by the `<gst-pipeline>`. They can track the offset and rate difference of their internal clock relative to the master clock by using the `gst-clock-get-calibration` function.

The master/slave synchronisation can be tuned with the "timeout", "window-size" and "window-threshold" properties. The "timeout" property defines the interval to sample the master clock and run the calibration functions. "window-size" defines the number of samples to use when calibrating and "window-threshold" defines the minimum number of samples before the calibration is performed.

Last reviewed on 2006-08-11 (0.10.10)

7.2 Usage

`<gst-clock>` [Class]

This `<gobject>` class defines the following properties:

`stats`	Enable clock stats (unimplemented)
`window-size`	The size of the window used to calculate rate and offset
`window-threshold`	The threshold to start calculating rate and offset
`timeout`	The amount of time, in nanoseconds, to sample master and slave clocks

`gst-clock-add-observation` (*self* `<gst-clock>`) [Function]
(*slave* `unsigned-long-long`) (*master* `unsigned-long-long`) ⇒ (*ret* `bool`)
(*r_squared* `double`)

`add-observation` [Method]

The time *master* of the master clock and the time *slave* of the slave clock are added to the list of observations. If enough observations are available, a linear regression algorithm is run on the observations and *clock* is recalibrated.

If this functions returns '`#t`', *r-squared* will contain the correlation coefficient of the interpollation. A value of 1.0 means a perfect regression was performed. This value can be used to control the sampling frequency of the master and slave clocks.

clock	a `<gst-clock>`
slave	a time on the slave
master	a time on the master
r-squared	a pointer to hold the result
ret	TRUE if enough observations were added to run the regression algorithm. MT safe.

`gst-clock-set-master` (*self* `<gst-clock>`) (*master* `<gst-clock>`) ⇒ (*ret* `bool`) [Function]

`set-master` [Method]

Set *master* as the master clock for *clock*. *clock* will be automatically calibrated so that `gst-clock-get-time` reports the same time as the master clock.

A clock provider that slaves its clock to a master can get the current calibration values with `gst-clock-get-calibration`.

master can be NULL in which case *clock* will not be slaved anymore. It will however keep reporting its time adjusted with the last configured rate and time offsets.

clock	a `<gst-clock>`
master	a master `<gst-clock>`
ret	TRUE if the clock is capable of being slaved to a master clock. Trying to set a master on a clock without the GST_CLOCK_FLAG_CAN_SET_MASTER flag will make this function return FALSE. MT safe.

`gst-clock-get-master` (*self* `<gst-clock>`) ⇒ (*ret* `<gst-clock>`) [Function]

`get-master` [Method]

Get the master clock that *clock* is slaved to or NULL when the clock is not slaved to any master clock.

clock	a `<gst-clock>`
ret	a master `<gst-clock>` or NULL when this clock is not slaved to a master clock. Unref after usage. MT safe.

`gst-clock-set-resolution` (*self* `<gst-clock>`) (*resolution* `unsigned-long-long`) ⇒ (*ret* `unsigned-long-long`) [Function]

`set-resolution` [Method]

Set the accuracy of the clock. Some clocks have the possibility to operate with different accuracy at the expense of more resource usage. There is normally no need to change the default resolution of a clock. The resolution of a clock can only be changed if the clock has the `<gst-clock-flag-can-set-resolution>` flag set.

clock	a `<gst-clock>`
resolution	The resolution to set
ret	the new resolution of the clock.

`gst-clock-get-resolution` (*self* `<gst-clock>`) ⇒ (*ret* `unsigned-long-long`) [Function]

`get-resolution` [Method]

Get the accuracy of the clock. The accuracy of the clock is the granularity of the values returned by `gst-clock-get-time`.

clock	a `<gst-clock>`
ret	the resolution of the clock in units of `<gst-clock-time>`. MT safe.

`gst-clock-get-time` (*self* `<gst-clock>`) [Function]
⇒ (*ret* `unsigned-long-long`)

`get-time` [Method]

Gets the current time of the given clock. The time is always monotonically increasing and adjusted according to the current offset and rate.

clock	a `<gst-clock>` to query
ret	the time of the clock. Or GST_CLOCK_TIME_NONE when giving wrong input. MT safe.

`gst-clock-new-single-shot-id` (*self* `<gst-clock>`) [Function]
(*time* `unsigned-long-long`) ⇒ (*ret* `<gst-clock-id>`)

`new-single-shot-id` [Method]

Get a `<gst-clock-id>` from *clock* to trigger a single shot notification at the requested time. The single shot id should be unreffed after usage.

clock	The `<gst-clock-id>` to get a single shot notification from
time	the requested time
ret	A `<gst-clock-id>` that can be used to request the time notification. MT safe.

`gst-clock-new-periodic-id` (*self* `<gst-clock>`) [Function]
(*start_time* `unsigned-long-long`) (*interval* `unsigned-long-long`)
⇒ (*ret* `<gst-clock-id>`)

`new-periodic-id` [Method]

Get an ID from *clock* to trigger a periodic notification. The periodeic notifications will be start at time start_time and will then be fired with the given interval. *id* should be unreffed after usage.

clock	The `<gst-clock-id>` to get a periodic notification id from
start-time	the requested start time
interval	the requested interval
ret	A `<gst-clock-id>` that can be used to request the time notification. MT safe.

`gst-clock-get-internal-time` (*self* `<gst-clock>`) [Function]
⇒ (*ret* `unsigned-long-long`)

`get-internal-time` [Method]

Gets the current internal time of the given clock. The time is returned unadjusted for the offset and the rate.

clock	a `<gst-clock>` to query
ret	the internal time of the clock. Or GST_CLOCK_TIME_NONE when giving wrong input. MT safe.

`gst-clock-adjust-unlocked` (*self* `<gst-clock>`) [Function]
(*internal* `unsigned-long-long`) ⇒ (*ret* `unsigned-long-long`)
`adjust-unlocked` [Method]

Converts the given *internal* clock time to the external time, adjusting for the rate and reference time set with `gst-clock-set-calibration` and making sure that the returned time is increasing. This function should be called with the clock's OBJECT_LOCK held and is mainly used by clock subclasses.

This function is te reverse of `gst-clock-unadjust-unlocked`.

clock	a `<gst-clock>` to use
internal	a clock time
ret	the converted time of the clock.

`gst-clock-get-calibration` (*self* `<gst-clock>`) [Function]
(*internal* `<gst-clock-time*>`) (*external* `<gst-clock-time*>`)
(*rate_num* `<gst-clock-time*>`) (*rate_denom* `<gst-clock-time*>`)
`get-calibration` [Method]

Gets the internal rate and reference time of *clock*. See `gst-clock-set-calibration` for more information.

internal, *external*, *rate-num*, and *rate-denom* can be left NULL if the caller is not interested in the values.

MT safe.

clock	a `<gst-clock>`
internal	a location to store the internal time
external	a location to store the external time
rate-num	a location to store the rate numerator
rate-denom	a location to store the rate denominator

`gst-clock-set-calibration` (*self* `<gst-clock>`) [Function]
(*internal* `unsigned-long-long`) (*external* `unsigned-long-long`)
(*rate_num* `unsigned-long-long`) (*rate_denom* `unsigned-long-long`)
`set-calibration` [Method]

Adjusts the rate and time of *clock*. A rate of 1/1 is the normal speed of the clock. Values bigger than 1/1 make the clock go faster.

internal and *external* are calibration parameters that arrange that `gst-clock-get-time` should have been *external* at internal time *internal*. This internal time should not be in the future; that is, it should be less than the value of `gst-clock-get-internal-time` when this function is called.

Subsequent calls to `gst-clock-get-time` will return clock times computed as follows:

```
      time = (internal_time - @internal) * @rate_num / @rate_denom + @external
```

This formula is implemented in `gst-clock-adjust-unlocked`. Of course, it tries to do the integer arithmetic as precisely as possible.

Note that `gst-clock-get-time` always returns increasing values so when you move the clock backwards, `gst-clock-get-time` will report the previous value until the clock catches up.

MT safe.

clock	a `<gst-clock>` to calibrate
internal	a reference internal time
external	a reference external time
rate-num	the numerator of the rate of the clock relative to its internal time
rate-denom	the denominator of the rate of the clock

`gst-clock-id-get-time` (*id* `<gst-clock-id>`) ⇒ (*ret* `unsigned-long-long`) [Function]

Get the time of the clock ID

id	The `<gst-clock-id>` to query
ret	the time of the given clock id. MT safe.

`gst-clock-id-wait` (*id* `<gst-clock-id>`) (*jitter* `<gst-clock-time-diff*>`) ⇒ (*ret* `<gst-clock-return>`) [Function]

Perform a blocking wait on *id*. *id* should have been created with `gst-clock-new-single-shot-id` or `gst-clock-new-periodic-id` and should not have been unscheduled with a call to `gst-clock-id-unschedule`.

If the *jitter* argument is not NULL and this function returns `<gst-clock-ok>` or `<gst-clock-early>`, it will contain the difference against the clock and the time of *id* when this method was called. Positive values indicate how late *id* was relative to the clock (in which case this function will return `<gst-clock-early>`). Negative values indicate how much time was spent waiting on the clock before this function returned.

id	The `<gst-clock-id>` to wait on
jitter	A pointer that will contain the jitter, can be NULL.
ret	the result of the blocking wait. `<gst-clock-early>` will be returned if the current clock time is past the time of *id*, `<gst-clock-ok>` if *id* was scheduled in time. `<gst-clock-unscheduled>` if *id* was unscheduled with `gst-clock-id-unschedule`. MT safe.

`gst-clock-id-wait-async` (*id* `<gst-clock-id>`) (*callback* `scm`) ⇒ (*ret* `<gst-clock-return>`) [Function]

Register a callback on the given `<gst-clock-id>`*id* with the given function and user_data. When passing a `<gst-clock-id>` with an invalid time to this function, the callback will be called immediatly with a time set to GST_CLOCK_TIME_NONE. The callback will be called when the time of *id* has been reached.

id	a `<gst-clock-id>` to wait on

func	The callback function
user-data	User data passed in the calback
ret	the result of the non blocking wait. MT safe.

`gst-clock-id-unschedule` (*id* `<gst-clock-id>`) [Function]

Cancel an outstanding request with *id*. This can either be an outstanding async notification or a pending sync notification. After this call, *id* cannot be used anymore to receive sync or async notifications, you need to create a new `<gst-clock-id>`.

MT safe.

id	The id to unschedule

`gst-clock-id-compare-func` (*id1* `<gconstpointer>`) [Function]
(*id2* `<gconstpointer>`) ⇒ (*ret* `int`)

Compares the two `<gst-clock-id>` instances. This function can be used as a GCompareFunc when sorting ids.

id1	A `<gst-clock-id>`
id2	A `<gst-clock-id>` to compare with
ret	negative value if a `<` b; zero if a = b; positive value if a `>` b MT safe.

8 gstconfig

Build configuration options

8.1 Overview

This describes the configuration options for GStreamer. When building GStreamer there are a lot of parts (known internally as "subsystems") that can be disabled for various reasons. The most common reasons are speed and size, which is important because GStreamer is designed to run on embedded systems.

If a subsystem is disabled, most of this changes are done in an API compatible way, so you don't need to adapt your code in most cases. It is never done in an ABI compatible way though. So if you want to disable a suybsystem, you have to rebuild all programs depending on GStreamer, too.

If a subsystem is disabled in GStreamer, a value is defined in <gst/gst.h>. You can check this if you do subsystem-specific stuff.

```
#ifndef GST_DISABLE_GST_DEBUG
// do stuff specific to the debugging subsystem
#endif // GST_DISABLE_GST_DEBUG
```

8.2 Usage

9 GstElementFactory

Create GstElements from a factory

9.1 Overview

`<gst-element-factory>` is used to create instances of elements. A GstElementfactory can be added to a `<gst-plugin>` as it is also a `<gst-plugin-feature>`.

Use the `gst-element-factory-find` and `gst-element-factory-create` functions to create element instances or use `gst-element-factory-make` as a convenient shortcut.

The following code example shows you how to create a GstFileSrc element.

```
#include <gst/gst.h>
GstElement *src;
GstElementFactory *srcfactory;
gst_init(&argc,&argv);
srcfactory = gst_element_factory_find("filesrc");
g_return_if_fail(srcfactory != NULL);
src = gst_element_factory_create(srcfactory,"src");
g_return_if_fail(src != NULL);
...
```

Last reviewed on 2005-11-23 (0.9.5)

9.2 Usage

`<gst-element-factory>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-element-register` (*plugin* `<gst-plugin>`) (*name* `mchars`) (*rank* `unsigned-int`) (*type* `<gtype>`) ⇒ (*ret* `bool`) [Function]

Create a new elementfactory capable of instantiating objects of the *type* and add the factory to *plugin*.

plugin	`<gst-plugin>` to register the element with
name	name of elements of this type
rank	rank of element (higher rank means more importance when autoplugging)
type	GType of element to register
ret	TRUE, if the registering succeeded, FALSE on error

`gst-element-factory-find` (*name* `mchars`) ⇒ (*ret* `<gst-element-factory>`) [Function]

Search for an element factory of the given name. Refs the returned element factory; caller is responsible for unreffing.

name	name of factory to find
ret	`<gst-element-factory>` if found, NULL otherwise

`gst-element-factory-get-longname` (*self* `<gst-element-factory>`) [Function]
  ⇒ (*ret* `mchars`)

`get-longname` [Method]

Gets the longname for this factory

factory	a `<gst-element-factory>`
ret	the longname

`gst-element-factory-get-klass` (*self* `<gst-element-factory>`) [Function]
  ⇒ (*ret* `mchars`)

`get-klass` [Method]

Gets the class for this factory.

factory	a `<gst-element-factory>`
ret	the class

`gst-element-factory-get-description` [Function]
  (*self* `<gst-element-factory>`) ⇒ (*ret* `mchars`)

`get-description` [Method]

Gets the description for this factory.

factory	a `<gst-element-factory>`
ret	the description

`gst-element-factory-get-author` (*self* `<gst-element-factory>`) [Function]
  ⇒ (*ret* `mchars`)

`get-author` [Method]

Gets the author for this factory.

factory	a `<gst-element-factory>`
ret	the author

`gst-element-factory-get-uri-type` (*self* `<gst-element-factory>`) [Function]
  ⇒ (*ret* `int`)

`get-uri-type` [Method]

Gets the type of URIs the element supports or GST_URI_UNKNOWN if none.

factory	a `<gst-element-factory>`
ret	type of URIs this element supports

`gst-element-factory-create` (*self* `<gst-element-factory>`) [Function]
  (*name* `mchars`) ⇒ (*ret* `<gst-element>`)

`create` [Method]

Create a new element of the type defined by the given elementfactory. It will be given the name supplied, since all elements require a name as their first argument.

factory	factory to instantiate
name	name of new element
ret	new `<gst-element>` or NULL if the element couldn't be created

`gst-element-factory-make` (*factoryname* `mchars`) (*name* `mchars`) [Function]
⇒ (*ret* `<gst-element>`)

Create a new element of the type defined by the given element factory. If name is NULL, then the element will receive a guaranteed unique name, consisting of the element factory name and a number. If name is given, it will be given the name supplied.

factoryname	a named factory to instantiate
name	name of new element
ret	new `<gst-element>` or NULL if unable to create element

`gst-element-factory-can-sink-caps` [Function]
(*self* `<gst-element-factory>`) (*caps* `<gst-caps>`) ⇒ (*ret* `bool`)

`can-sink-caps` [Method]

Checks if the factory can sink the given capability.

factory	factory to query
caps	the caps to check
ret	true if it can sink the capabilities

`gst-element-factory-can-src-caps` (*self* `<gst-element-factory>`) [Function]
(*caps* `<gst-caps>`) ⇒ (*ret* `bool`)

`can-src-caps` [Method]

Checks if the factory can source the given capability.

factory	factory to query
caps	the caps to check
ret	true if it can src the capabilities

10 GstElement

Abstract base class for all pipeline elements

10.1 Overview

GstElement is the abstract base class needed to construct an element that can be used in a GStreamer pipeline. Please refer to the plugin writers guide for more information on creating `<gst-element>` subclasses.

The name of a `<gst-element>` can be get with `gst-element-get-name` and set with `gst-element-set-name`. For speed, `gst-element-name` can be used in the core when using the appropriate locking. Do not use this in plug-ins or applications in order to retain ABI compatibility.

All elements have pads (of the type `<gst-pad>`). These pads link to pads on other elements. `<gst-buffer>` flow between these linked pads. A `<gst-element>` has a `<g-list>` of `<gst-pad>` structures for all their input (or sink) and output (or source) pads. Core and plug-in writers can add and remove pads with `gst-element-add-pad` and `gst-element-remove-pad`.

A pad of an element can be retrieved by name with `gst-element-get-pad`. An iterator of all pads can be retrieved with `gst-element-iterate-pads`.

Elements can be linked through their pads. If the link is straightforward, use the `gst-element-link` convenience function to link two elements, or `gst-element-link-many` for more elements in a row. Use `gst-element-link-filtered` to link two elements constrained by a specified set of `<gst-caps>`. For finer control, use `gst-element-link-pads` and `gst-element-link-pads-filtered` to specify the pads to link on each element by name.

Each element has a state (see `<gst-state>`). You can get and set the state of an element with `gst-element-get-state` and `gst-element-set-state`. To get a string representation of a `<gst-state>`, use `gst-element-state-get-name`.

You can get and set a `<gst-clock>` on an element using `gst-element-get-clock` and `gst-element-set-clock`. Some elements can provide a clock for the pipeline if `gst-element-provides-clock` returns '`#t`'. With the `gst-element-provide-clock` method one can retrieve the clock provided by such an element. Not all elements require a clock to operate correctly. If `gst-element-requires-clock` returns '`#t`', a clock should be set on the element with `gst-element-set-clock`.

Note that clock slection and distribution is normally handled by the toplevel `<gst-pipeline>` so the clock functions are only to be used in very specific situations.

Last reviewed on 2006-03-12 (0.10.5)

10.2 Usage

`<gst-element>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`pad-added` (*arg0* `<gst-pad>`) [Signal on `<gst-element>`]

a new `<gst-pad>` has been added to the element.

`pad-removed` (*arg0* `<gst-pad>`) [Signal on `<gst-element>`]

a `<gst-pad>` has been removed from the element

`no-more-pads` [Signal on `<gst-element>`]

This signals that the element will not generate more dynamic pads.

`gst-element-class-add-pad-template` [Function]
(*klass* `<gst-element-class>`) (*templ* `<gst-pad-template>`)

Adds a padtemplate to an element class. This is mainly used in the _base_init functions of classes.

klass	the `<gst-element-class>` to add the pad template to.
templ	a `<gst-pad-template>` to add to the element class.

`gst-element-class-get-pad-template` [Function]
(*klass* `<gst-element-class>`) (*klass* `mchars`)
⇒ (*ret* `<gst-pad-template>`)

Retrieves a padtemplate from *element-class* with the given name.

> If you use this function in the `<g-instance-init-func>` of an object class that has subclasses, make sure to pass the g_class parameter of the `<g-instance-init-func>` here.

element-class	a `<gst-element-class>` to get the pad template of.
name	the name of the `<gst-pad-template>` to get.
ret	the `<gst-pad-template>` with the given name, or '`#f`' if none was found. No unreferencing is necessary.

`gst-element-class-set-details` (*klass* `<gst-element-class>`) [Function]
(*details* `<gst-element-details*>`)

Sets the detailed information for a `<gst-element-class>`.

> This function is for use in _base_init functions only.

The *details* are copied.

klass	class to set details for
details	details to set

`gst-element-add-pad` (*self* `<gst-element>`) (*pad* `<gst-pad>`) [Function]
⇒ (*ret* `bool`)

`add-pad` [Method]

Adds a pad (link point) to *element*. *pad*'s parent will be set to *element*; see `gst-object-set-parent` for refcounting information.

Pads are not automatically activated so elements should perform the needed steps to activate the pad in case this pad is added in the PAUSED or PLAYING state. See `gst-pad-set-active` for more information about activating pads.

The pad and the element should be unlocked when calling this function.

This function will emit the `<gst-element::pad-added>` signal on the element.

element	a `<gst-element>` to add the pad to.
pad	the `<gst-pad>` to add to the element.
ret	'`#t`' if the pad could be added. This function can fail when a pad with the same name already existed or the pad already had another parent. MT safe.

`gst-element-get-pad` (*self* `<gst-element>`) (*name* `mchars`) ⇒ (*ret* `<gst-pad>`) [Function]

`get-pad` [Method]

Retrieves a pad from *element* by name. Tries `gst-element-get-static-pad` first, then `gst-element-get-request-pad`.

> Usage of this function is not recommended as it is unclear if the reference to the result pad should be released with `gst-object-unref` in case of a static pad or `gst-element-release-request-pad` in case of a request pad.

element	a `<gst-element>`.
name	the name of the pad to retrieve.
ret	the `<gst-pad>` if found, otherwise '`#f`'. Unref or Release after usage, depending on the type of the pad.

`gst-element-create-all-pads` (*self* `<gst-element>`) [Function]

`create-all-pads` [Method]

Creates a pad for each pad template that is always available. This function is only useful during object intialization of subclasses of `<gst-element>`.

element	a `<gst-element>` to create pads for

`gst-element-get-compatible-pad` (*self* `<gst-element>`) (*pad* `<gst-pad>`) (*caps* `<gst-caps>`) ⇒ (*ret* `<gst-pad>`) [Function]

`get-compatible-pad` [Method]

Looks for an unlinked pad to which the given pad can link. It is not guaranteed that linking the pads will work, though it should work in most cases.

element	a `<gst-element>` in which the pad should be found.
pad	the `<gst-pad>` to find a compatible one for.
caps	the `<gst-caps>` to use as a filter.
ret	the `<gst-pad>` to which a link can be made, or '`#f`' if one cannot be found.

`gst-element-get-request-pad` (*self* `<gst-element>`) (*name* `mchars`) ⇒ (*ret* `<gst-pad>`) [Function]

`get-request-pad` [Method]

Retrieves a pad from the element by name. This version only retrieves request pads. The pad should be released with `gst-element-release-request-pad`.

element	a `<gst-element>` to find a request pad of.
name	the name of the request `<gst-pad>` to retrieve.
ret	requested `<gst-pad>` if found, otherwise '`#f`'. Release after usage.

`gst-element-get-static-pad` (*self* `<gst-element>`) (*name* `mchars`) ⇒ (*ret* `<gst-pad>`) [Function]

`get-static-pad` [Method]

Retrieves a pad from *element* by name. This version only retrieves already-existing (i.e. 'static') pads.

element	a `<gst-element>` to find a static pad of.
name	the name of the static `<gst-pad>` to retrieve.
ret	the requested `<gst-pad>` if found, otherwise '`#f`'. unref after usage. MT safe.

`gst-element-no-more-pads` (*self* `<gst-element>`) [Function]

`no-more-pads` [Method]

Use this function to signal that the element does not expect any more pads to show up in the current pipeline. This function should be called whenever pads have been added by the element itself. Elements with `<gst-pad-sometimes>` pad templates use this in combination with autopluggers to figure out that the element is done initializing its pads.

This function emits the `<gst-element::no-more-pads>` signal.

MT safe.

element	a `<gst-element>`

`gst-element-release-request-pad` (*self* `<gst-element>`) (*pad* `<gst-pad>`) [Function]

`release-request-pad` [Method]

Makes the element free the previously requested pad as obtained with `gst-element-get-request-pad`.

MT safe.

element	a `<gst-element>` to release the request pad of.
pad	the `<gst-pad>` to release.

`gst-element-remove-pad` (*self* `<gst-element>`) (*pad* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]

`remove-pad` [Method]

Removes *pad* from *element*. *pad* will be destroyed if it has not been referenced elsewhere using `gst-object-unparent`.

This function is used by plugin developers and should not be used by applications. Pads that were dynamically requested from elements with `gst-element-get-request-pad` should be released with the `gst-element-release-request-pad` function instead.

Pads are not automatically deactivated so elements should perform the needed steps to deactivate the pad in case this pad is removed in the PAUSED or PLAYING state. See `gst-pad-set-active` for more information about deactivating pads.

The pad and the element should be unlocked when calling this function.

This function will emit the `<gst-element::pad-removed>` signal on the element.

element	a `<gst-element>` to remove pad from.
pad	the `<gst-pad>` to remove from the element.
ret	'`#t`' if the pad could be removed. Can return '`#f`' if the pad does not belong to the provided element. MT safe.

`gst-element-iterate-pads` (*self* `<gst-element>`) [Function]
⇒ (*ret* `<gst-iterator*>`)
`iterate-pads` [Method]

Retrieves an iterattor of *element*'s pads. The iterator should be freed after usage.

element	a `<gst-element>` to iterate pads of.
ret	the `<gst-iterator>` of `<gst-pad>`. Unref each pad after use. MT safe.

`gst-element-iterate-sink-pads` (*self* `<gst-element>`) [Function]
⇒ (*ret* `<gst-iterator*>`)
`iterate-sink-pads` [Method]

Retrieves an iterator of *element*'s sink pads.

element	a `<gst-element>`.
ret	the `<gst-iterator>` of `<gst-pad>`. Unref each pad after use. MT safe.

`gst-element-iterate-src-pads` (*self* `<gst-element>`) [Function]
⇒ (*ret* `<gst-iterator*>`)
`iterate-src-pads` [Method]

Retrieves an iterator of *element*'s source pads.

element	a `<gst-element>`.
ret	the `<gst-iterator>` of `<gst-pad>`. Unref each pad after use. MT safe.

`gst-element-link` (*self* `<gst-element>`) (*dest* `<gst-element>`) [Function]
⇒ (*ret* `bool`)
`link` [Method]

Links *src* to *dest*. The link must be from source to destination; the other direction will not be tried. The function looks for existing pads that aren't linked yet. It will request new pads if necessary. Such pads need to be released manualy when unlinking. If multiple links are possible, only one is established.

Make sure you have added your elements to a bin or pipeline with `gst-bin-add` before trying to link them.

src	a `<gst-element>` containing the source pad.
dest	the `<gst-element>` containing the destination pad.
ret	TRUE if the elements could be linked, FALSE otherwise.

`gst-element-unlink` (*self* `<gst-element>`) (*dest* `<gst-element>`) [Function]
`unlink` [Method]

Unlinks all source pads of the source element with all sink pads of the sink element to which they are linked.

If the link has been made using `gst-element-link`, it could have created an request-pad, which has to be released using `gst-element-release-request-pad`.

src	the source `<gst-element>` to unlink.
dest	the sink `<gst-element>` to unlink.

`gst-element-link-pads` (*self* `<gst-element>`) (*srcpadname* `mchars`) (*dest* `<gst-element>`) (*destpadname* `mchars`) ⇒ (*ret* `bool`) [Function]
`link-pads` [Method]

Links the two named pads of the source and destination elements. Side effect is that if one of the pads has no parent, it becomes a child of the parent of the other element. If they have different parents, the link fails.

src	a `<gst-element>` containing the source pad.
srcpadname	the name of the `<gst-pad>` in source element or NULL for any pad.
dest	the `<gst-element>` containing the destination pad.
destpadname	the name of the `<gst-pad>` in destination element, or NULL for any pad.
ret	TRUE if the pads could be linked, FALSE otherwise.

`gst-element-unlink-pads` (*self* `<gst-element>`) (*srcpadname* `mchars`) (*dest* `<gst-element>`) (*destpadname* `mchars`) [Function]
`unlink-pads` [Method]

Unlinks the two named pads of the source and destination elements.

src	a `<gst-element>` containing the source pad.
srcpadname	the name of the `<gst-pad>` in source element.
dest	a `<gst-element>` containing the destination pad.
destpadname	the name of the `<gst-pad>` in destination element.

`gst-element-link-pads-filtered` (*self* `<gst-element>`) (*srcpadname* `mchars`) (*dest* `<gst-element>`) (*destpadname* `mchars`) (*filter* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]
`link-pads-filtered` [Method]

Links the two named pads of the source and destination elements. Side effect is that if one of the pads has no parent, it becomes a child of the parent of the other element. If they have different parents, the link fails. If *caps* is not `#f`, makes sure that the caps of the link is a subset of *caps*.

src a `<gst-element>` containing the source pad.

srcpadname
the name of the `<gst-pad>` in source element or NULL for any pad.

dest the `<gst-element>` containing the destination pad.

destpadname
the name of the `<gst-pad>` in destination element or NULL for any pad.

filter the `<gst-caps>` to filter the link, or `#f` for no filter.

ret TRUE if the pads could be linked, FALSE otherwise.

`gst-element-link-filtered` (*self* `<gst-element>`) [Function]
(*dest* `<gst-element>`) (*filter* `<gst-caps>`) ⇒ (*ret* `bool`)
`link-filtered` [Method]

Links *src* to *dest* using the given caps as filtercaps. The link must be from source to destination; the other direction will not be tried. The function looks for existing pads that aren't linked yet. It will request new pads if necessary. If multiple links are possible, only one is established.

Make sure you have added your elements to a bin or pipeline with `gst-bin-add` before trying to link them.

src a `<gst-element>` containing the source pad.

dest the `<gst-element>` containing the destination pad.

filter the `<gst-caps>` to filter the link, or `#f` for no filter.

ret TRUE if the pads could be linked, FALSE otherwise.

`gst-element-set-base-time` (*self* `<gst-element>`) [Function]
(*time* `unsigned-long-long`)
`set-base-time` [Method]

Set the base time of an element. See `gst-element-get-base-time`.

MT safe.

element a `<gst-element>`.

time the base time to set.

`gst-element-get-base-time` (*self* `<gst-element>`) [Function]
⇒ (*ret* `unsigned-long-long`)
`get-base-time` [Method]

Returns the base time of the element. The base time is the absolute time of the clock when this element was last put to PLAYING. Subtracting the base time from the clock time gives the stream time of the element.

element a `<gst-element>`.

ret the base time of the element. MT safe.

`gst-element-set-bus` (*self* `<gst-element>`) (*bus* `<gst-bus>`) [Function]
`set-bus` [Method]

Sets the bus of the element. Increases the refcount on the bus. For internal use only, unless you're testing elements.

MT safe.

element	a `<gst-element>` to set the bus of.
bus	the `<gst-bus>` to set.

`gst-element-get-bus` (*self* `<gst-element>`) ⇒ (*ret* `<gst-bus>`) [Function]
`get-bus` [Method]

Returns the bus of the element. Note that only a `<gst-pipeline>` will provide a bus for the application.

element	a `<gst-element>` to get the bus of.
ret	the element's `<gst-bus>`. unref after usage. MT safe.

`gst-element-get-factory` (*self* `<gst-element>`) [Function]
⇒ (*ret* `<gst-element-factory>`)
`get-factory` [Method]

Retrieves the factory that was used to create this element.

element	a `<gst-element>` to request the element factory of.
ret	the `<gst-element-factory>` used for creating this element. no refcounting is needed.

`gst-element-set-index` (*self* `<gst-element>`) (*index* `<gst-index>`) [Function]
`set-index` [Method]

Set *index* on the element. The refcount of the index will be increased, any previously set index is unreffed.

MT safe.

element	a `<gst-element>`.
index	a `<gst-index>`.

`gst-element-get-index` (*self* `<gst-element>`) ⇒ (*ret* `<gst-index>`) [Function]
`get-index` [Method]

Gets the index from the element.

element	a `<gst-element>`.
ret	a `<gst-index>` or '`#f`' when no index was set on the element. unref after usage. MT safe.

`gst-element-is-indexable` (*self* `<gst-element>`) ⇒ (*ret* `bool`) [Function]
`is-indexable` [Method]

Queries if the element can be indexed.

element	a `<gst-element>`.
ret	TRUE if the element can be indexed. MT safe.

`gst-element-requires-clock` (*self* `<gst-element>`) ⇒ (*ret* `bool`) [Function]
`requires-clock` [Method]

Query if the element requires a clock.

element	a `<gst-element>` to query
ret	'`#t`' if the element requires a clock MT safe.

`gst-element-set-clock` (*self* `<gst-element>`) (*clock* `<gst-clock>`) ⇒ (*ret* `bool`) [Function]
`set-clock` [Method]

Sets the clock for the element. This function increases the refcount on the clock. Any previously set clock on the object is unreffed.

element	a `<gst-element>` to set the clock for.
clock	the `<gst-clock>` to set for the element.
ret	'`#t`' if the element accepted the clock. An element can refuse a clock when it, for example, is not able to slave its internal clock to the *clock* or when it requires a specific clock to operate. MT safe.

`gst-element-get-clock` (*self* `<gst-element>`) ⇒ (*ret* `<gst-clock>`) [Function]
`get-clock` [Method]

Gets the currently configured clock of the element. This is the clock as was last set with `gst-element-set-clock`.

element	a `<gst-element>` to get the clock of.
ret	the `<gst-clock>` of the element. unref after usage. MT safe.

`gst-element-provides-clock` (*self* `<gst-element>`) ⇒ (*ret* `bool`) [Function]
`provides-clock` [Method]

Query if the element provides a clock. A `<gst-clock>` provided by an element can be used as the global `<gst-clock>` for the pipeline. An element that can provide a clock is only required to do so in the PAUSED state, this means when it is fully negotiated and has allocated the resources to operate the clock.

element	a `<gst-element>` to query
ret	'`#t`' if the element provides a clock MT safe.

`gst-element-provide-clock` (*self* `<gst-element>`) ⇒ (*ret* `<gst-clock>`) [Function]
`provide-clock` [Method]

Get the clock provided by the given element.

> An element is only required to provide a clock in the PAUSED state. Some elements can provide a clock in other states.

element	a `<gst-element>` to query
ret	the GstClock provided by the element or '`#f`' if no clock could be provided. Unref after usage. MT safe.

`gst-element-set-state` (*self* `<gst-element>`) (*state* `<gst-state>`) ⇒ (*ret* `<gst-state-change-return>`) [Function]

`set-state` [Method]

Sets the state of the element. This function will try to set the requested state by going through all the intermediary states and calling the class's state change function for each.

This function can return `<gst-state-change-async>`, in which case the element will perform the remainder of the state change asynchronously in another thread. An application can use `gst-element-get-state` to wait for the completion of the state change or it can wait for a state change message on the bus.

element	a `<gst-element>` to change state of.
state	the element's new `<gst-state>`.
ret	Result of the state change using `<gst-state-change-return>`. MT safe.

`gst-element-get-state` (*self* `<gst-element>`) (*state* `<gst-state*>`) (*pending* `<gst-state*>`) (*timeout* `unsigned-long-long`) ⇒ (*ret* `<gst-state-change-return>`) [Function]

`get-state` [Method]

Gets the state of the element.

For elements that performed an ASYNC state change, as reported by `gst-element-set-state`, this function will block up to the specified timeout value for the state change to complete. If the element completes the state change or goes into an error, this function returns immediately with a return value of '`GST_STATE_CHANGE_SUCCESS`' or '`GST_STATE_CHANGE_FAILURE`' respectively.

For elements that did not return '`GST_STATE_CHANGE_ASYNC`', this function returns the current and pending state immediately.

This function returns '`GST_STATE_CHANGE_NO_PREROLL`' if the element successfully changed its state but is not able to provide data yet. This mostly happens for live sources that only produce data in the PLAYING state. While the state change return is equivalent to '`GST_STATE_CHANGE_SUCCESS`', it is returned to the application to signal that some sink elements might not be able to complete their state change because an element is not producing data to complete the preroll. When setting the element to playing, the preroll will complete and playback will start.

element	a `<gst-element>` to get the state of.
state	a pointer to `<gst-state>` to hold the state. Can be '`#f`'.
pending	a pointer to `<gst-state>` to hold the pending state. Can be '`#f`'.
timeout	a `<gst-clock-time>` to specify the timeout for an async state change or '`GST_CLOCK_TIME_NONE`' for infinite timeout.
ret	'`GST_STATE_CHANGE_SUCCESS`' if the element has no more pending state and the last state change succeeded, '`GST_STATE_CHANGE_ASYNC`' if the element is still performing a state change or '`GST_STATE_CHANGE_FAILURE`' if the last state change failed. MT safe.

`gst-element-set-locked-state` (*self* `<gst-element>`) (*locked_state* `bool`) ⇒ (*ret* `bool`) [Function]
`set-locked-state` [Method]

Locks the state of an element, so state changes of the parent don't affect this element anymore.

MT safe.

element a `<gst-element>`

locked-state
TRUE to lock the element's state

ret TRUE if the state was changed, FALSE if bad parameters were given or the elements state-locking needed no change.

`gst-element-is-locked-state` (*self* `<gst-element>`) ⇒ (*ret* `bool`) [Function]
`is-locked-state` [Method]

Checks if the state of an element is locked. If the state of an element is locked, state changes of the parent don't affect the element. This way you can leave currently unused elements inside bins. Just lock their state before changing the state from `<gst-state-null>`.

MT safe.

element a `<gst-element>`.

ret TRUE, if the element's state is locked.

`gst-element-abort-state` (*self* `<gst-element>`) [Function]
`abort-state` [Method]

Abort the state change of the element. This function is used by elements that do asynchronous state changes and find out something is wrong.

This function should be called with the STATE_LOCK held.

MT safe.

element a `<gst-element>` to abort the state of.

`gst-element-continue-state` (*self* `<gst-element>`) (*ret* `<gst-state-change-return>`) ⇒ (*ret* `<gst-state-change-return>`) [Function]
`continue-state` [Method]

Commit the state change of the element and proceed to the next pending state if any. This function is used by elements that do asynchronous state changes. The core will normally call this method automatically when an element returned '`GST_STATE_CHANGE_SUCCESS`' from the state change function.

If after calling this method the element still has not reached the pending state, the next state change is performed.

This method is used internally and should normally not be called by plugins or applications.

element a `<gst-element>` to continue the state change of.

ret The previous state return value

ret The result of the commit state change. MT safe.

`gst-element-lost-state` (*self* `<gst-element>`) [Function]
`lost-state` [Method]

Brings the element to the lost state. The current state of the element is copied to the pending state so that any call to `gst-element-get-state` will return '`GST_STATE_CHANGE_ASYNC`'.

An ASYNC_START message is posted with an indication to distribute a new base_time to the element. If the element was PLAYING, it will go to PAUSED. The element will be restored to its PLAYING state by the parent pipeline when it prerolls again.

This is mostly used for elements that lost their preroll buffer in the '`GST_STATE_PAUSED`' or '`GST_STATE_PLAYING`' state after a flush, they will go to their pending state again when a new preroll buffer is queued. This function can only be called when the element is currently not in error or an async state change.

This function is used internally and should normally not be called from plugins or applications.

MT safe.

element a `<gst-element>` the state is lost of

`gst-element-state-get-name` (*state* `<gst-state>`) ⇒ (*ret* `mchars`) [Function]

Gets a string representing the given state.

state a `<gst-state>` to get the name of.

ret a string with the name of the state.

`gst-element-sync-state-with-parent` (*self* `<gst-element>`) [Function]
⇒ (*ret* `bool`)
`sync-state-with-parent` [Method]

Tries to change the state of the element to the same as its parent. If this function returns FALSE, the state of element is undefined.

element a `<gst-element>`.

ret TRUE, if the element's state could be synced to the parent's state. MT safe.

`gst-element-found-tags` (*self* `<gst-element>`) [Function]
(*list* `<gst-tag-list*>`)
`found-tags` [Method]

Posts a message to the bus that new tags were found, and pushes an event to all sourcepads. Takes ownership of the *list*.

This is a utility method for elements. Applications should use the `<gst-tag-setter>` interface.

element element for which we found the tags.

list list of tags.

`gst-element-found-tags-for-pad` (*self* `<gst-element>`) (*pad* `<gst-pad>`) (*list* `<gst-tag-list*>`) [Function]

`found-tags-for-pad` [Method]

Posts a message to the bus that new tags were found and pushes the tags as event. Takes ownership of the *list*.

This is a utility method for elements. Applications should use the `<gst-tag-setter>` interface.

element	element for which to post taglist to bus.
pad	pad on which to push tag-event.
list	the taglist to post on the bus and create event from.

`gst-element-message-full` (*self* `<gst-element>`) (*type* `<gst-message-type>`) (*domain* `unsigned-int`) (*code* `int`) (*text* `mchars`) (*debug* `mchars`) (*file* `mchars`) (*function* `mchars`) (*line* `int`) [Function]

`message-full` [Method]

Post an error, warning or info message on the bus from inside an element.

type must be of `<gst-message-error>`, `<gst-message-warning>` or `<gst-message-info>`.

MT safe.

element	a `<gst-element>` to send message from
type	the `<gst-message-type>`
domain	the GStreamer GError domain this message belongs to
code	the GError code belonging to the domain
text	an allocated text string to be used as a replacement for the default message connected to code, or '`#f`'
debug	an allocated debug message to be used as a replacement for the default debugging information, or '`#f`'
file	the source code file where the error was generated
function	the source code function where the error was generated
line	the source code line where the error was generated

`gst-element-post-message` (*self* `<gst-element>`) (*message* `<gst-message>`) ⇒ (*ret* `bool`) [Function]

`post-message` [Method]

Post a message on the element's `<gst-bus>`. This function takes ownership of the message; if you want to access the message after this call, you should add an additional reference before calling.

element	a `<gst-element>` posting the message
message	a `<gst-message>` to post
ret	'`#t`' if the message was successfully posted. The function returns '`#f`' if the element did not have a bus. MT safe.

`gst-element-get-query-types` (*self* `<gst-element>`) [Function]
  ⇒ (*ret* `<gst-query-type*>`)
`get-query-types` [Method]

Get an array of query types from the element. If the element doesn't implement a query types function, the query will be forwarded to the peer of a random linked sink pad.

element	a `<gst-element>` to query
ret	An array of `<gst-query-type>` elements that should not be freed or modified. MT safe.

`gst-element-query` (*self* `<gst-element>`) (*query* `<gst-query>`) [Function]
  ⇒ (*ret* `bool`)
`query` [Method]

Performs a query on the given element.

For elements that don't implement a query handler, this function forwards the query to a random srcpad or to the peer of a random linked sinkpad of this element.

element	a `<gst-element>` to perform the query on.
query	the `<gst-query>`.
ret	TRUE if the query could be performed. MT safe.

`gst-element-query-convert` (*self* `<gst-element>`) [Function]
  (*src_format* `<gst-format>`) (*src_val* `int64`) (*dest_format* `<gst-format*>`)
  ⇒ (*ret* `bool`) (*dest_val* `int64`)
`query-convert` [Method]

Queries an element to convert *src-val* in *src-format* to *dest-format*.

element	a `<gst-element>` to invoke the convert query on.
src-format	a `<gst-format>` to convert from.
src-val	a value to convert.
dest-format	a pointer to the `<gst-format>` to convert to.
dest-val	a pointer to the result.
ret	TRUE if the query could be performed.

`gst-element-query-position` (*self* `<gst-element>`) [Function]
  (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*cur* `int64`)
`query-position` [Method]

Queries an element for the stream position.

element	a `<gst-element>` to invoke the position query on.
format	a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.
cur	A location in which to store the current position, or NULL.
ret	TRUE if the query could be performed.

`gst-element-query-duration` (*self* `<gst-element>`) (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*duration* `int64`) [Function]
`query-duration` [Method]

Queries an element for the total stream duration.

element	a `<gst-element>` to invoke the duration query on.
format	a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.
duration	A location in which to store the total duration, or NULL.
ret	TRUE if the query could be performed.

`gst-element-send-event` (*self* `<gst-element>`) (*event* `<gst-event>`) ⇒ (*ret* `bool`) [Function]
`send-event` [Method]

Sends an event to an element. If the element doesn't implement an event handler, the event will be pushed on a random linked sink pad for upstream events or a random linked source pad for downstream events.

This function takes owership of the provided event so you should `gst-event-ref` it if you want to reuse the event after this call.

element	a `<gst-element>` to send the event to.
event	the `<gst-event>` to send to the element.
ret	'`#t`' if the event was handled. MT safe.

`gst-element-seek-simple` (*self* `<gst-element>`) (*format* `<gst-format>`) (*seek_flags* `<gst-seek-flags>`) (*seek_pos* `int64`) ⇒ (*ret* `bool`) [Function]
`seek-simple` [Method]

Simple API to perform a seek on the given element, meaning it just seeks to the given position relative to the start of the stream. For more complex operations like segment seeks (e.g. for looping) or changing the playback rate or seeking relative to the last configured playback segment you should use `gst-element-seek`.

In a completely prerolled PAUSED or PLAYING pipeline, seeking is always guaranteed to return '`#t`' on a seekable media type or '`#f`' when the media type is certainly not seekable (such as a live stream).

Some elements allow for seeking in the READY state, in this case they will store the seek event and execute it when they are put to PAUSED. If the element supports seek in READY, it will always return '`#t`' when it receives the event in the READY state.

element	a `<gst-element>` to seek on
format	a `<gst-format>` to execute the seek in, such as `<gst-format-time>`
seek-flags	seek options; playback applications will usually want to use GST_SEEK_FLAG_FLUSH `|` GST_SEEK_FLAG_KEY_UNIT here

seek-pos position to seek to (relative to the start); if you are doing a seek in `<gst-format-time>` this value is in nanoseconds - multiply with `<gst-second>` to convert seconds to nanoseconds or with `<gst-msecond>` to convert milliseconds to nanoseconds.

ret '`#t`' if the seek operation succeeded (the seek might not always be executed instantly though)

Since 0.10.7

`gst-element-seek` (*self* `<gst-element>`) (*rate* `double`) (*format* `<gst-format>`) (*flags* `<gst-seek-flags>`) (*cur_type* `<gst-seek-type>`) (*cur* `int64`) (*stop_type* `<gst-seek-type>`) (*stop* `int64`) ⇒ (*ret* `bool`) [Function]

`seek` [Method]

Sends a seek event to an element. See `gst-event-new-seek` for the details of the parameters. The seek event is sent to the element using `gst-element-send-event`.

element a `<gst-element>` to send the event to.

rate The new playback rate

format The format of the seek values

flags The optional seek flags.

cur-type The type and flags for the new current position

cur The value of the new current position

stop-type The type and flags for the new stop position

stop The value of the new stop position

ret '`#t`' if the event was handled. MT safe.

11 GstGError

Categorized error messages

11.1 Overview

GStreamer elements can throw non-fatal warnings and fatal errors. Higher-level elements and applications can programatically filter the ones they are interested in or can recover from, and have a default handler handle the rest of them.

The rest of this section will use the term *error* to mean both (non-fatal) warnings and (fatal) errors; they are treated similarly.

Errors from elements are the combination of a `<g-error>` and a debug string. The `<g-error>` contains: - a domain type: CORE, LIBRARY, RESOURCE or STREAM - a code: an enum value specific to the domain - a translated, human-readable message - a non-translated additional debug string, which also contains - file and line information

Elements do not have the context required to decide what to do with errors. As such, they should only inform about errors, and stop their processing. In short, an element doesn't know what it is being used for.

It is the application or compound element using the given element that has more context about the use of the element. Errors can be received by listening to the `<gst-bus>` of the element/pipeline for `<gst-message>` objects with the type '`GST_MESSAGE_ERROR`' or '`GST_MESSAGE_WARNING`'. The thrown errors should be inspected, and filtered if appropriate.

An application is expected to, by default, present the user with a dialog box (or an equivalent) showing the error message. The dialog should also allow a way to get at the additional debug information, so the user can provide bug reporting information.

A compound element is expected to forward errors by default higher up the hierarchy; this is done by default in the same way as for other types of `<gst-message>`.

When applications or compound elements trigger errors that they can recover from, they can filter out these errors and take appropriate action. For example, an application that gets an error from xvimagesink that indicates all XVideo ports are taken, the application can attempt to use another sink instead.

Elements throw errors using the `<gst-element-error>` convenience macro:

```
GST_ELEMENT_ERROR (src, RESOURCE, NOT_FOUND,
  (_("No file name specified for reading.")), (NULL));
```

Things to keep in mind:

- Don't go off inventing new error codes. The ones currently provided should be enough. If you find your type of error does not fit the current codes, you should use FAILED.
- Don't provide a message if the default one suffices. this keeps messages more uniform. Use (NULL) - not forgetting the parentheses.
- If you do supply a custom message, it should be marked for translation. The message should start with a capital and end with a period. The message should describe the error in short, in a human-readable form, and without any complex technical terms. A

user interface will present this message as the first thing a user sees. Details, technical info, ... should go in the debug string.

- The debug string can be as you like. Again, use (NULL) if there's nothing to add - file and line number will still be passed. `<gst-error-system>` can be used as a shortcut to give debug information on a system call error.

Last reviewed on 2006-09-15 (0.10.10)

11.2 Usage

`gst-error-get-message` (*domain* `unsigned-int`) (*code* `int`) [Function]
⇒ (*ret* `mchars`)

Get a string describing the error message in the current locale.

domain	the GStreamer error domain this error belongs to.
code	the error code belonging to the domain.
ret	a newly allocated string describing the error message in the current locale.

12 GstEvent

Structure describing events that are passed up and down a pipeline

12.1 Overview

The event class provides factory methods to construct and functions query (parse) 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` and elements will use `gst-pad-send-event` or `gst-pad-push-event`. The event should be unreffed with `gst-event-unref` if it has not been sent.

Events that have been received can be parsed with their respective gst_event_parse_*() functions.

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` is used to create a seek event. It takes the needed parameters to specity 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");
...
```

Last reviewed on 2006-09-6 (0.10.10)

12.2 Usage

`<gst-event>` [Class]

`gst-event-get-structure` (*self* `<gst-event>`) [Function]
⇒ (*ret* `<gst-structure>`)

get-structure [Method]

Access the structure of the event.

event	The `<gst-event>`.
ret	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-new-buffer-size (*format* `<gst-format>`) (*minsize* `int64`) (*maxsize* `int64`) (*async* `bool`) ⇒ (*ret* `<gst-event>`) [Function]

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 prefered.

format	buffer format
minsize	minimum buffer size
maxsize	maximum buffer size
async	thread behavior
ret	a new `<gst-event>`

gst-event-new-eos ⇒ (*ret* `<gst-event>`) [Function]

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-unexpected>` as a `<gst-flow-return>` 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>` 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.

ret	The new EOS event.

gst-event-new-flush-start ⇒ (*ret* `<gst-event>`) [Function]

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-wrong-state>` when used for data flow with `gst-pad-push`, `gst-pad-chain`, `gst-pad-alloc-buffer`, `gst-pad-get-range` and `gst-pad-pull-range`. Any event (except a `<gst-event-flush-stop>`) received on a flushing pad will return '`#f`' 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.

ret	A new flush start event.

gst-event-new-flush-stop ⇒ (*ret* `<gst-event>`) [Function]

Allocate a new flush stop event. The flush stop event can be sent upstream and downstream and travels out-of-bounds 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 preceeding FLUSH_START event stopped the dataflow.

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

ret	A new flush stop event.

gst-event-new-navigation (*structure* `<gst-structure>`) ⇒ (*ret* `<gst-event>`) [Function]

Create a new navigation event from the given description.

structure	description of the event
ret	a new `<gst-event>`

gst-event-new-new-segment (*update* `bool`) (*rate* `double`) (*format* `<gst-format>`) (*start* `int64`) (*stop* `int64`) (*position* `int64`) ⇒ (*ret* `<gst-event>`) [Function]

Allocate a new newsegment event with the given format/values tripplets

This method calls `gst-event-new-new-segment-full` passing a default value of 1.0 for applied_rate

update	is this segment an update to a previous one
rate	a new rate for playback
format	The format of the segment values
start	the start value of the segment
stop	the stop value of the segment
position	stream position
ret	A new newsegment event.

gst-event-new-new-segment-full (*update* `bool`) (*rate* `double`) (*applied_rate* `double`) (*format* `<gst-format>`) (*start* `int64`) (*stop* `int64`) (*position* `int64`) ⇒ (*ret* `<gst-event>`) [Function]

Allocate a new newsegment event with the given format/values triplets.

The newsegment 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 position 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. *position* represents the stream_time of a buffer carrying a timestamp of *start*. *position* 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 newsegment event with *rate* of 1.0 and *applied-rate* of 2.0

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

position + (TIMESTAMP(buf) - start) * ABS (rate * applied_rate)

update	Whether this segment is an update to a previous one
rate	A new rate for playback
applied-rate	The rate factor which has already been applied
format	The format of the segment values
start	The start value of the segment
stop	The stop value of the segment
position	stream position
ret	A new newsegment event.

Since 0.10.6

`gst-event-new-qos` (*proportion* `double`) (*diff* `unsigned-long-long`) (*timestamp* `unsigned-long-long`) ⇒ (*ret* `<gst-event>`) [Function]

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. Typically sinks generate these events for each buffer they receive.

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.

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 possitive *diff*, all buffers with timestamp <= *timestamp* + *diff* will certainly arrive late in the sink as well.

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

proportion	the proportion of the qos message
diff	The time difference of the last Clock sync
timestamp	The timestamp of the buffer
ret	A new QOS event.

`gst-event-new-seek` (*rate* `double`) (*format* `<gst-format>`) (*flags* `<gst-seek-flags>`) (*cur_type* `<gst-seek-type>`) (*cur* `int64`) (*stop_type* `<gst-seek-type>`) (*stop* `int64`) ⇒ (*ret* `<gst-event>`) [Function]

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>`.

start-type and *stop-type* specify how to adjust the currently configured start and stop fields in *segment*. Adjustments can be made relative or absolute to the last configured values. A type of `<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 if updated, it must be different from -1 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>` and update the playback segment current position with a `<gst-seek-type-set>` to the desired position.

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
ret	A new seek event.

`gst-event-new-tag` (*taglist* `<gst-tag-list*>`) ⇒ (*ret* `<gst-event>`) [Function]

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

taglist	metadata list
ret	a new `<gst-event>`

`gst-event-parse-buffer-size` (*self* `<gst-event>`) [Function]
(*format* `<gst-format*>`) ⇒ (*minsize* `int64`) (*maxsize* `int64`) (*async* `bool`)
`parse-buffer-size` [Method]

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

event	The event to query
format	A pointer to store the format in
minsize	A pointer to store the minsize in
maxsize	A pointer to store the maxsize in
async	A pointer to store the async-flag in

`gst-event-parse-new-segment` (*self* `<gst-event>`) [Function]
(*format* `<gst-format*>`) ⇒ (*update* `bool`) (*rate* `double`) (*start* `int64`)
(*stop* `int64`) (*position* `int64`)
`parse-new-segment` [Method]

Get the update flag, rate, format, start, stop and position in the newsegment event. In general, `gst-event-parse-new-segment-full` should be used instead of this, to also retrieve the applied_rate value of the segment. See `gst-event-new-new-segment-full` for a full description of the newsegment event.

event	The event to query
update	A pointer to the update flag of the segment
rate	A pointer to the rate of the segment
format	A pointer to the format of the newsegment values
start	A pointer to store the start value in
stop	A pointer to store the stop value in
position	A pointer to store the stream time in

`gst-event-parse-new-segment-full` (*self* `<gst-event>`) [Function]
(*format* `<gst-format*>`) ⇒ (*update* `bool`) (*rate* `double`)
(*applied_rate* `double`) (*start* `int64`) (*stop* `int64`) (*position* `int64`)
`parse-new-segment-full` [Method]

Get the update, rate, applied_rate, format, start, stop and position in the newsegment event. See `gst-event-new-new-segment-full` for a full description of the newsegment event.

event	The event to query
update	A pointer to the update flag of the segment
rate	A pointer to the rate of the segment

applied-rate
A pointer to the applied_rate of the segment

format A pointer to the format of the newsegment values

start A pointer to store the start value in

stop A pointer to store the stop value in

position A pointer to store the stream time in

Since 0.10.6

`gst-event-parse-qos` (*self* `<gst-event>`) (*diff* `<gst-clock-time-diff*>`) (*timestamp* `<gst-clock-time*>`) ⇒ (*proportion* `double`) [Function]

`parse-qos` [Method]

Get the proportion, diff and timestamp in the qos event. See `gst-event-new-qos` for more information about the different QoS values.

event The event to query

proportion
A pointer to store the proportion in

diff A pointer to store the diff in

timestamp
A pointer to store the timestamp in

`gst-event-parse-seek` (*self* `<gst-event>`) (*format* `<gst-format*>`) (*flags* `<gst-seek-flags*>`) (*cur_type* `<gst-seek-type*>`) (*stop_type* `<gst-seek-type*>`) ⇒ (*rate* `double`) (*cur* `int64`) (*stop* `int64`) [Function]

`parse-seek` [Method]

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

event a seek event

rate result location for the rate

format result location for the stream format

flags result location for the `<gst-seek-flags>`

start-type result location for the `<gst-seek-type>` of the start position

start result location for the start postion expressed in *format*

stop-type result location for the `<gst-seek-type>` of the stop position

stop result location for the stop postion expressed in *format*

`gst-event-parse-tag` (*self* `<gst-event>`) (*taglist* `<gst-tag-list**>`) [Function]

`parse-tag` [Method]

Parses a tag *event* and stores the results in the given *taglist* location.

event a tag event

taglist pointer to metadata list

`gst-event-type-get-flags` (*self* `<gst-event-type*>`) [Function]
⇒ (*ret* `<gst-event-type-flags>`)

Gets the `<gst-event-type-flags>` associated with *type*.

type	a `<gst-event-type>`
ret	a `<gst-event-type-flags>`.

`gst-event-type-get-name` (*self* `<gst-event-type*>`) [Function]
⇒ (*ret* `mchars`)

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

type	the event type
ret	a reference to the static name of the event.

`gst-event-type-to-quark` (*self* `<gst-event-type*>`) [Function]
⇒ (*ret* `unsigned-int`)

Get the unique quark for the given event type.

type	the event type
ret	the quark associated with the event type

13 GstFilter

A utility function to filter GLists.

13.1 Overview

```
GList *node;
GstObject *result = NULL;

node = gst_filter_run (list, (GstFilterFunc) my_filter, TRUE, NULL);
if (node) {
  result = GST_OBJECT (node->data);
  gst_object_ref (result);
  gst_list_free (node);
}
```

13.2 Usage

14 GstFormat

Dynamically register new data formats

14.1 Overview

GstFormats functions are used to register a new format to the gstreamer core. Formats can be used to perform seeking or conversions/query operations.

14.2 Usage

`gst-format-get-name` (*self* `<gst-format*>`) ⇒ (*ret* `mchars`) [Function]

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

format a `<gst-format>`

ret a reference to the static name of the format or NULL if the format is unknown.

`gst-format-to-quark` (*self* `<gst-format*>`) ⇒ (*ret* `unsigned-int`) [Function]

Get the unique quark for the given format.

format a `<gst-format>`

ret the quark associated with the format or 0 if the format is unknown.

`gst-format-register` (*nick* `mchars`) (*description* `mchars`) ⇒ (*ret* `<gst-format>`) [Function]

Create a new GstFormat based on the nick or return an already registered format with that nick.

nick The nick of the new format

description The description of the new format

ret A new GstFormat or an already registered format with the same nick. MT safe.

`gst-format-get-by-nick` (*nick* `mchars`) ⇒ (*ret* `<gst-format>`) [Function]

Return the format registered with the given nick.

nick The nick of the format

ret The format with *nick* or GST_FORMAT_UNDEFINED if the format was not registered.

`gst-formats-contains` (*self* `<gst-format*>`) (*format* `<gst-format>`) ⇒ (*ret* `bool`) [Function]

See if the given format is inside the format array.

formats The format array to search

format the format to find

ret TRUE if the format is found inside the array

`gst-format-get-details` (*format* `<gst-format>`) [Function]
⇒ (*ret* `<gst-format-definition*>`)

Get details about the given format.

format	The format to get details of
ret	The `<gst-format-definition>` for *format* or NULL on failure. MT safe.

`gst-format-iterate-definitions` ⇒ (*ret* `<gst-iterator*>`) [Function]

Iterate all the registered formats. The format definition is read only.

ret	A GstIterator of `<gst-format-definition>`.

15 GstGhostPad

Pseudo link pads

15.1 Overview

GhostPads are useful when organizing pipelines with `<gst-bin>` like elements. The idea here is to create hierarchical element graphs. The bin element contains a sub-graph. Now one would like to treat the bin-element like other `<gst-elements>`. This is where GhostPads come into play. A GhostPad acts as a proxy for another pad. Thus the bin can have sink and source ghost-pads that are associated with sink and source pads of the child elements.

If the target pad is known at creation time, `gst-ghost-pad-new` is the function to use to get a ghost-pad. Otherwise one can use `gst-ghost-pad-new-no-target` to create the ghost-pad and use `gst-ghost-pad-set-target` to establish the association later on.

Note that GhostPads add overhead to the data processing of a pipeline.

Last reviewed on 2005-11-18 (0.9.5)

15.2 Usage

`gst-ghost-pad-new` (*name* `mchars`) (*target* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-pad>`)

Create a new ghostpad with *target* as the target. The direction will be taken from the target pad. *target* must be unlinked.

Will ref the target.

name	the name of the new pad, or NULL to assign a default name.
target	the pad to ghost.
ret	a new `<gst-pad>`, or NULL in case of an error.

`gst-ghost-pad-new-no-target` (*name* `mchars`) [Function]
(*dir* `<gst-pad-direction>`) ⇒ (*ret* `<gst-pad>`)

Create a new ghostpad without a target with the given direction. A target can be set on the ghostpad later with the `gst-ghost-pad-set-target` function.

The created ghostpad will not have a padtemplate.

name	the name of the new pad, or NULL to assign a default name.
dir	the direction of the ghostpad
ret	a new `<gst-pad>`, or NULL in case of an error.

`gst-ghost-pad-new-from-template` (*name* `mchars`) [Function]
(*target* `<gst-pad>`) (*templ* `<gst-pad-template>`) ⇒ (*ret* `<gst-pad>`)

Create a new ghostpad with *target* as the target. The direction will be taken from the target pad. The template used on the ghostpad will be *template*.

Will ref the target.

name	the name of the new pad, or NULL to assign a default name.

target	the pad to ghost.
templ	the `<gst-pad-template>` to use on the ghostpad.
ret	a new `<gst-pad>`, or NULL in case of an error.

Since 0.10.10

`gst-ghost-pad-set-target` (*self* `<gst-ghost-pad*>`) (*newtarget* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]

Set the new target of the ghostpad *gpad*. Any existing target is unlinked and links to the new target are established.

gpad	the `<gst-ghostpad>`
newtarget	the new pad target
ret	TRUE if the new target could be set. This function can return FALSE when the internal pads could not be linked.

`gst-ghost-pad-get-target` (*self* `<gst-ghost-pad*>`) ⇒ (*ret* `<gst-pad>`) [Function]

Get the target pad of `<gpad>`. Unref target pad after usage.

gpad	the `<gst-ghostpad>`
ret	the target `<gst-pad>`, can be NULL if the ghostpad has no target set. Unref target pad after usage.

16 GstImplementsInterface

Core interface implemented by instances that allows runtime querying of interface availability

16.1 Overview

Provides interface functionality on per instance basis and not per class basis, which is the case for gobject.

16.2 Usage

`gst-element-implements-interface` (*self* `<gst-element>`) (*iface_type* `<gtype>`) ⇒ (*ret* `bool`) [Function]

`implements-interface` [Method]

Test whether the given element implements a certain interface of type iface_type, and test whether it is supported for this specific instance.

element	`<gst-element>` to check for the implementation of the interface
iface-type	(final) type of the interface which we want to be implemented
ret	whether or not the element implements the interface.

`gst-implements-interface-cast` (*from* `<gpointer>`) (*type* `<gtype>`) ⇒ (*ret* `<gpointer>`) [Function]

cast a given object to an interface type, and check whether this interface is supported for this specific instance.

from	the object (any sort) from which to cast to the interface
type	the interface type to cast to
ret	a gpointer to the interface type

`gst-implements-interface-check` (*from* `<gpointer>`) (*type* `<gtype>`) ⇒ (*ret* `bool`) [Function]

check a given object for an interface implementation, and check whether this interface is supported for this specific instance.

from	the object (any sort) from which to check from for the interface
type	the interface type to check for
ret	whether or not the object implements the given interface

17 GstIndexFactory

Create GstIndexes from a factory

17.1 Overview

GstIndexFactory is used to dynamically create GstIndex implementations.

17.2 Usage

`<gst-index-factory>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-index-factory-new` (*name* `mchars`) (*longdesc* `mchars`) (*type* `<gtype>`) ⇒ (*ret* `<gst-index-factory>`) [Function]

Create a new indexfactory with the given parameters

name	name of indexfactory to create
longdesc	long description of indexfactory to create
type	the GType of the GstIndex element of this factory
ret	a new `<gst-index-factory>`.

`gst-index-factory-destroy` (*self* `<gst-index-factory>`) [Function]
`destroy` [Method]

Removes the index from the global list.

factory	factory to destroy

`gst-index-factory-find` (*name* `mchars`) ⇒ (*ret* `<gst-index-factory>`) [Function]

Search for an indexfactory of the given name.

name	name of indexfactory to find
ret	`<gst-index-factory>` if found, NULL otherwise

`gst-index-factory-create` (*self* `<gst-index-factory>`) ⇒ (*ret* `<gst-index>`) [Function]
`create` [Method]

Create a new `<gst-index>` instance from the given indexfactory.

factory	the factory used to create the instance
ret	A new `<gst-index>` instance.

`gst-index-factory-make` (*name* `mchars`) ⇒ (*ret* `<gst-index>`) [Function]

Create a new `<gst-index>` instance from the indexfactory with the given name.

name	the name of the factory used to create the instance
ret	A new `<gst-index>` instance.

18 GstIndex

Generate indexes on objects

18.1 Overview

GstIndex is used to generate a stream index of one or more elements in a pipeline.

18.2 Usage

`<gst-index>` [Class]

This `<gobject>` class defines the following properties:

`resolver` Select a predefined object to string mapper

`entry-added` (*arg0* `<gst-index-entry>`) [Signal on `<gst-index>`]

Is emitted when a new entry is added to the index.

`gst-index-new` ⇒ (*ret* `<gst-index>`) [Function]

Create a new tileindex object

ret a new index object

`gst-index-commit` (*self* `<gst-index>`) (*id* `int`) [Function]
`commit` [Method]

Tell the index that the writer with the given id is done with this index and is not going to write any more entries to it.

index the index to commit

id the writer that commited the index

`gst-index-get-group` (*self* `<gst-index>`) ⇒ (*ret* `int`) [Function]
`get-group` [Method]

Get the id of the current group.

index the index to get the current group from

ret the id of the current group.

`gst-index-new-group` (*self* `<gst-index>`) ⇒ (*ret* `int`) [Function]
`new-group` [Method]

Create a new group for the given index. It will be set as the current group.

index the index to create the new group in

ret the id of the newly created group.

`gst-index-set-group` (*self* `<gst-index>`) (*groupnum* `int`) [Function]
⇒ (*ret* `bool`)
`set-group` [Method]

Set the current groupnumber to the given argument.

index the index to set the new group in

groupnum the groupnumber to set

ret TRUE if the operation succeeded, FALSE if the group did not exist.

`gst-index-set-certainty` (*self* `<gst-index>`) (*certainty* `<gst-index-certainty>`) [Function]
`set-certainty` [Method]

Set the certainty of the given index.

index	the index to set the certainty on
certainty	the certainty to set

`gst-index-get-certainty` (*self* `<gst-index>`) ⇒ (*ret* `<gst-index-certainty>`) [Function]
`get-certainty` [Method]

Get the certainty of the given index.

index	the index to get the certainty of
ret	the certainty of the index.

`gst-index-set-filter-full` (*self* `<gst-index>`) (*filter* `<gst-index-filter>`) (*user_data* `<gpointer>`) (*user_data_destroy* `<g-destroy-notify>`) [Function]
`set-filter-full` [Method]

Lets the app register a custom filter function so that it can select what entries should be stored in the index.

index	the index to register the filter on
filter	the filter to register
user-data	data passed to the filter function
user-data-destroy	function to call when *user-data* is unset

`gst-index-get-writer-id` (*self* `<gst-index>`) (*writer* `<gst-object>`) ⇒ (*ret* `bool`) (*id* `int`) [Function]
`get-writer-id` [Method]

Before entries can be added to the index, a writer should obtain a unique id. The methods to add new entries to the index require this id as an argument.

The application can implement a custom function to map the writer object to a string. That string will be used to register or look up an id in the index.

index	the index to get a unique write id for
writer	the GstObject to allocate an id for
id	a pointer to a gint to hold the id
ret	TRUE if the writer would be mapped to an id.

`gst-index-add-format` (*self* `<gst-index>`) (*id* `int`) (*format* `<gst-format>`) ⇒ (*ret* `<gst-index-entry*>`) [Function]
`add-format` [Method]

Adds a format entry into the index. This function is used to map dynamic GstFormat ids to their original format key.

index	the index to add the entry to
id	the id of the index writer
format	the format to add to the index
ret	a pointer to the newly added entry in the index.

`gst-index-add-associationv` (*self* `<gst-index>`) (*id* `int`) (*flags* `<gst-assoc-flags>`) (*n* `int`) (*list* `<gst-index-association*>`) ⇒ (*ret* `<gst-index-entry*>`) [Function]
`add-associationv` [Method]

Associate given format/value pairs with each other.

index	the index to add the entry to
id	the id of the index writer
flags	optinal flags for this entry
n	number of associations
list	list of associations
ret	a pointer to the newly added entry in the index.

`gst-index-add-object` (*self* `<gst-index>`) (*id* `int`) (*key* `mchars`) (*type* `<gtype>`) (*object* `<gpointer>`) ⇒ (*ret* `<gst-index-entry*>`) [Function]
`add-object` [Method]

Add the given object to the index with the given key.

This function is not yet implemented.

index	the index to add the object to
id	the id of the index writer
key	a key for the object
type	the GType of the object
object	a pointer to the object to add
ret	a pointer to the newly added entry in the index.

`gst-index-add-id` (*self* `<gst-index>`) (*id* `int`) (*description* `mchars`) ⇒ (*ret* `<gst-index-entry*>`) [Function]
`add-id` [Method]

Add an id entry into the index.

index	the index to add the entry to
id	the id of the index writer
description	the description of the index writer
ret	a pointer to the newly added entry in the index.

`gst-index-get-assoc-entry` (*self* `<gst-index>`) (*id* `int`) (*method* `<gst-index-lookup-method>`) (*flags* `<gst-assoc-flags>`) (*format* `<gst-format>`) (*value* `int64`) ⇒ (*ret* `<gst-index-entry*>`) [Function]

`get-assoc-entry` [Method]

Finds the given format/value in the index

index	the index to search
id	the id of the index writer
method	The lookup method to use
flags	Flags for the entry
format	the format of the value
value	the value to find
ret	the entry associated with the value or NULL if the value was not found.

`gst-index-get-assoc-entry-full` (*self* `<gst-index>`) (*id* `int`) (*method* `<gst-index-lookup-method>`) (*flags* `<gst-assoc-flags>`) (*format* `<gst-format>`) (*value* `int64`) (*func* `<g-compare-data-func>`) (*user_data* `<gpointer>`) ⇒ (*ret* `<gst-index-entry*>`) [Function]

`get-assoc-entry-full` [Method]

Finds the given format/value in the index with the given compare function and user_data.

index	the index to search
id	the id of the index writer
method	The lookup method to use
flags	Flags for the entry
format	the format of the value
value	the value to find
func	the function used to compare entries
user-data	user data passed to the compare function
ret	the entry associated with the value or NULL if the value was not found.

`gst-index-entry-assoc-map` (*self* `<gst-index-entry*>`) (*format* `<gst-format>`) ⇒ (*ret* `bool`) (*value* `int64`) [Function]

Gets alternative formats associated with the indexentry.

entry	the index to search
format	the format of the value the find
value	a pointer to store the value
ret	TRUE if there was a value associated with the given format.

19 GstInfo

Debugging and logging facilities

19.1 Overview

GStreamer's debugging subsystem is an easy way to get information about what the application is doing. It is not meant for programming errors. Use GLib methods (g_warning and friends) for that.

The debugging subsystem works only after GStreamer has been initialized - for example by calling `gst-init`.

The debugging subsystem is used to log informational messages while the application runs. Each messages has some properties attached to it. Among these properties are the debugging category, the severity (called "level" here) and an optional `<gobject>` it belongs to. Each of these messages is sent to all registered debugging handlers, which then handle the messages. GStreamer attaches a default handler on startup, which outputs requested messages to stderr.

Messages are output by using shortcut macros like `<gst-debug>`, `<gst-cat-error-object>` or similar. These all expand to calling `gst-debug-log` with the right parameters. The only thing a developer will probably want to do is define his own categories. This is easily done with 3 lines. At the top of your code, declare the variables and set the default category. After that you only need to initialize the category. Initialization must be done before the category is used first. Plugins do this in their plugin_init function, libraries and applications should do that during their initialization.

```
GST_DEBUG_CATEGORY_STATIC (my_category);     // define category (statically)
#define GST_CAT_DEFAULT my_category     // set as default

GST_DEBUG_CATEGORY_INIT (my_category, "my category",
                         0, "This is my very own");
```

The whole debugging subsystem can be disabled at build time with passing the –disable-gst-debug switch to configure. If this is done, every function, macro and even structs described in this file evaluate to default values or nothing at all. So don't take addresses of these functions or use other tricks. If you must do that for some reason, there is still an option. If the debugging subsystem was compiled out, `<gst-disable-gst-debug>` is defined in <gst/gst.h>, so you can check that before doing your trick. Disabling the debugging subsystem will give you a slight (read: unnoticeable) speed increase and will reduce the size of your compiled code. The GStreamer library itself becomes around 10% smaller.

Please note that there are naming conventions for the names of debugging categories. These are explained at `gst-debug-category-init`.

19.2 Usage

`gst-debug-message-get` (*self* `<gst-debug-message*>`) [Function]
⇒ (*ret* `mchars`)

Gets the string representation of a `<gst-debug-message>`. This function is used in debug handlers to extract the message.

message	a debug message
ret	the string representation of a `<gst-debug-message>`.

`gst-debug-log-default` (*category* `<gst-debug-category*>`) (*level* `<gst-debug-level>`) (*file* `mchars`) (*function* `mchars`) (*line* `int`) (*object* `<gobject>`) (*message* `<gst-debug-message*>`) (*unused* `<gpointer>`) [Function]

The default logging handler used by GStreamer. Logging functions get called whenever a macro like GST_DEBUG or similar is used. This function outputs the message and additional info using the glib error handler. You can add other handlers by using `gst-debug-add-log-function`. And you can remove this handler by calling gst_debug_remove_log_function(gst_debug_log_default);

category	category to log
level	level of the message
file	the file that emitted the message, usually the __FILE__ identifier
function	the function that emitted the message
line	the line from that the message was emitted, usually __LINE__
object	the object this message relates to or NULL if none
message	the actual message
unused	an unused variable, reserved for some user_data.

`gst-debug-level-get-name` (*level* `<gst-debug-level>`) ⇒ (*ret* `mchars`) [Function]

Get the string representation of a debugging level

level	the level to get the name for
ret	the name

`gst-debug-set-active` (*active* `bool`) [Function]

If activated, debugging messages are sent to the debugging handlers. It makes sense to deactivate it for speed issues.

This function is not threadsafe. It makes sense to only call it during initialization.

active	Whether to use debugging output or not

`gst-debug-is-active` ⇒ (*ret* `bool`) [Function]

Checks if debugging output is activated.

ret	TRUE, if debugging is activated

`gst-debug-set-colored` (*colored* `bool`) [Function]

Sets or unsets the use of coloured debugging output.

colored	Whether to use colored output or not

`gst-debug-is-colored` ⇒ (*ret* `bool`) [Function]

Checks if the debugging output should be colored.

ret TRUE, if the debug output should be colored.

`gst-debug-set-default-threshold` (*level* `<gst-debug-level>`) [Function]

Sets the default threshold to the given level and updates all categories to use this threshold.

level level to set

`gst-debug-get-default-threshold` ⇒ (*ret* `<gst-debug-level>`) [Function]

Returns the default threshold that is used for new categories.

ret the default threshold level

`gst-debug-set-threshold-for-name` (*name* `mchars`) (*level* `<gst-debug-level>`) [Function]

Sets all categories which match the given glob style pattern to the given level.

name name of the categories to set

level level to set them to

`gst-debug-unset-threshold-for-name` (*name* `mchars`) [Function]

Resets all categories with the given name back to the default level.

name name of the categories to set

`gst-debug-category-set-threshold` (*self* `<gst-debug-category*>`) (*level* `<gst-debug-level>`) [Function]

Sets the threshold of the category to the given level. Debug information will only be output if the threshold is lower or equal to the level of the debugging message.

Do not use this function in production code, because other functions may change the threshold of categories as side effect. It is however a nice function to use when debugging (even from gdb).

category a `<gst-debug-category>` to set threshold of.

level the `<gst-debug-level>` threshold to set.

`gst-debug-category-reset-threshold` (*self* `<gst-debug-category*>`) [Function]

Resets the threshold of the category to the default level. Debug information will only be output if the threshold is lower or equal to the level of the debugging message. Use this function to set the threshold back to where it was after using `gst-debug-category-set-threshold`.

category a `<gst-debug-category>` to reset threshold of.

`gst-debug-category-get-threshold` (*self* `<gst-debug-category*>`) ⇒ (*ret* `<gst-debug-level>`) [Function]

Returns the threshold of a `<gst-debug-category>`.

category a `<gst-debug-category>` to get threshold of.

ret the `<gst-debug-level>` that is used as threshold.

`gst-debug-category-get-name` (*self* `<gst-debug-category*>`) ⇒ (*ret* `mchars`) [Function]

Returns the name of a debug category.

category	a `<gst-debug-category>` to get name of.
ret	the name of the category.

`gst-debug-category-get-color` (*self* `<gst-debug-category*>`) ⇒ (*ret* `unsigned-int`) [Function]

Returns the color of a debug category used when printing output in this category.

category	a `<gst-debug-category>` to get the color of.
ret	the color of the category.

`gst-debug-category-get-description` (*self* `<gst-debug-category*>`) ⇒ (*ret* `mchars`) [Function]

Returns the description of a debug category.

category	a `<gst-debug-category>` to get the description of.
ret	the description of the category.

`gst-debug-get-all-categories` ⇒ (*ret* `gslist-of`) [Function]

Returns a snapshot of a all categories that are currently in use . This list may change anytime. The caller has to free the list after use.

ret	the list of categories

`gst-debug-construct-term-color` (*colorinfo* `unsigned-int`) ⇒ (*ret* `mchars`) [Function]

Constructs a string that can be used for getting the desired color in color terminals. You need to free the string after use.

colorinfo	the color info
ret	a string containing the color definition

20 GstIterator

Object to retrieve multiple elements in a threadsafe way.

20.1 Overview

A GstIterator is used to retrieve multiple objects from another object in a threadsafe way.

Various GStreamer objects provide access to their internal structures using an iterator.

The basic use pattern of an iterator is as follows:

```
it = _get_iterator(object);
done = FALSE;
while (!done) {
  switch (gst_iterator_next (it, &item)) {
    case GST_ITERATOR_OK:
      ... use/change item here...
      gst_object_unref (item);
      break;
    case GST_ITERATOR_RESYNC:
      ...rollback changes to items...
      gst_iterator_resync (it);
      break;
    case GST_ITERATOR_ERROR:
      ...wrong parameter were given...
      done = TRUE;
      break;
    case GST_ITERATOR_DONE:
      done = TRUE;
      break;
  }
}
gst_iterator_free (it);
```

Last reviewed on 2005-11-09 (0.9.4)

20.2 Usage

`gst-iterator-new` (*size* `unsigned-int`) (*type* `<gtype>`) (*lock* `<g-mutex*>`) (*next* `<gst-iterator-next-function>`) (*item* `<gst-iterator-item-function>`) (*resync* `<gst-iterator-resync-function>`) (*free* `<gst-iterator-free-function>`) ⇒ (*ret* `<gst-iterator*>`) (*master_cookie* `unsigned-int32`) [Function]

Create a new iterator. This function is mainly used for objects implementing the next/resync/free function to iterate a data structure.

For each item retrieved, the *item* function is called with the lock held. The *free* function is called when the iterator is freed.

size	the size of the iterator structure
type	`<g-type>` of children
lock	pointer to a `<g-mutex>`.
master-cookie	pointer to a guint32 to protect the iterated object.
next	function to get next item
item	function to call on each item retrieved
resync	function to resync the iterator
free	function to free the iterator
ret	the new `<gst-iterator>`. MT safe.

`gst-iterator-new-list` (*type* `<gtype>`) (*lock* `<g-mutex*>`) (*list* `<g-list**>`) (*owner* `<gpointer>`) (*item* `<gst-iterator-item-function>`) (*free* `<gst-iterator-dispose-function>`) ⇒ (*ret* `<gst-iterator*>`) (*master_cookie* `unsigned-int32`) [Function]

Create a new iterator designed for iterating *list*.

type	`<g-type>` of elements
lock	pointer to a `<g-mutex>` protecting the list.
master-cookie	pointer to a guint32 to protect the list.
list	pointer to the list
owner	object owning the list
item	function to call for each item
free	function to call when the iterator is freed
ret	the new `<gst-iterator>` for *list*. MT safe.

`gst-iterator-next` (*self* `<gst-iterator*>`) (*elem* `<gpointer*>`) ⇒ (*ret* `<gst-iterator-result>`) [Function]

Get the next item from the iterator. For iterators that return refcounted objects, the returned object will have its refcount increased and should therefore be unreffed after usage.

it	The `<gst-iterator>` to iterate
elem	pointer to hold next element
ret	The result of the iteration. Unref after usage if this is a refcounted object. MT safe.

`gst-iterator-resync` (*self* `<gst-iterator*>`) [Function]

Resync the iterator. this function is mostly called after `gst-iterator-next` returned '`GST_ITERATOR_RESYNC`'.

MT safe.

it	The `<gst-iterator>` to resync

`gst-iterator-push` (*self* `<gst-iterator*>`) [Function]
(*other* `<gst-iterator*>`)

Pushes *other* iterator onto *it*. All calls performed on *it* are forwarded tot *other*. If *other* returns `<gst-iterator-done>`, it is popped again and calls are handled by *it* again.

This function is mainly used by objects implementing the iterator next function to recurse into substructures.

MT safe.

it	The `<gst-iterator>` to use
other	The `<gst-iterator>` to push

`gst-iterator-filter` (*self* `<gst-iterator*>`) [Function]
(*func* `<g-compare-func>`) (*user_data* `<gpointer>`)
⇒ (*ret* `<gst-iterator*>`)

Create a new iterator from an existing iterator. The new iterator will only return those elements that match the given compare function *func*. *func* should return 0 for elements that should be included in the iterator.

When this iterator is freed, *it* will also be freed.

it	The `<gst-iterator>` to filter
func	the compare function to select elements
user-data	user data passed to the compare function
ret	a new `<gst-iterator>`. MT safe.

`gst-iterator-fold` (*self* `<gst-iterator*>`) [Function]
(*func* `<gst-iterator-fold-function>`) (*ret* `<gvalue>`)
(*user_data* `<gpointer>`) ⇒ (*ret* `<gst-iterator-result>`)

Folds *func* over the elements of *iter*. That is to say, *proc* will be called as *proc* (object, *ret*, *user-data*) for each object in *iter*. The normal use of this procedure is to accumulate the results of operating on the objects in *ret*.

This procedure can be used (and is used internally) to implement the foreach and find_custom operations.

The fold will proceed as long as *func* returns TRUE. When the iterator has no more arguments, `<gst-iterator-done>` will be returned. If *func* returns FALSE, the fold will stop, and `<gst-iterator-ok>` will be returned. Errors or resyncs will cause fold to return `<gst-iterator-error>` or `<gst-iterator-resync>` as appropriate.

The iterator will not be freed.

it	The `<gst-iterator>` to fold over

func	the fold function
ret	the seed value passed to the fold function
user-data	user data passed to the fold function
ret	A `<gst-iterator-result>`, as described above. MT safe.

`gst-iterator-foreach` (*self* `<gst-iterator*>`) (*func* `<g-func>`) (*user_data* `<gpointer>`) ⇒ (*ret* `<gst-iterator-result>`) [Function]

Iterate over all element of *it* and call the given function *func* for each element.

it	The `<gst-iterator>` to iterate
func	the function to call for each element.
user-data	user data passed to the function
ret	the result call to `gst-iterator-fold`. The iterator will not be freed. MT safe.

`gst-iterator-find-custom` (*self* `<gst-iterator*>`) (*func* `<g-compare-func>`) (*user_data* `<gpointer>`) ⇒ (*ret* `<gpointer>`) [Function]

Find the first element in *it* that matches the compare function *func*. *func* should return 0 when the element is found.

The iterator will not be freed.

This function will return NULL if an error or resync happened to the iterator.

it	The `<gst-iterator>` to iterate
func	the compare function to use
user-data	user data passed to the compare function
ret	The element in the iterator that matches the compare function or NULL when no element matched. MT safe.

21 GstMessage

Lightweight objects to signal the application of pipeline events

21.1 Overview

Messages are implemented as a subclass of `<gst-mini-object>` with a generic `<gst-structure>` as the content. This allows for writing custom messages without requiring an API change while allowing a wide range of different types of messages.

Messages are posted by objects in the pipeline and are passed to the application using the `<gst-bus>`.

The basic use pattern of posting a message on a `<gst-bus>` is as follows:

```
    gst_bus_post (bus, gst_message_new_eos());
```

A `<gst-element>` usually posts messages on the bus provided by the parent container using `gst-element-post-message`.

Last reviewed on 2005-11-09 (0.9.4)

21.2 Usage

`<gst-message>` [Class]

`gst-message-type-to-quark` (*self* `<gst-message-type*>`) [Function]
⇒ (*ret* `unsigned-int`)

Get the unique quark for the given message type.

type the message type

ret the quark associated with the message type

`gst-message-type-get-name` (*self* `<gst-message-type*>`) [Function]
⇒ (*ret* `mchars`)

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

type the message type

ret a reference to the static name of the message.

`gst-message-get-structure` (*self* `<gst-message>`) [Function]
⇒ (*ret* `<gst-structure>`)

`get-structure` [Method]

Access the structure of the message.

message The `<gst-message>`.

ret The structure of the message. The structure is still owned by the message, which means that you should not free it and that the pointer becomes invalid when you free the message. MT safe.

`gst-message-new-application` (*src* `<gst-object>`) (*structure* `<gst-structure>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new application-typed message. GStreamer will never create these messages; they are a gift from us to you. Enjoy.

src	The object originating the message.
structure	The structure for the message. The message will take ownership of the structure.
ret	The new application message. MT safe.

`gst-message-new-clock-provide` (*src* `<gst-object>`) (*clock* `<gst-clock>`) (*ready* `bool`) ⇒ (*ret* `<gst-message>`) [Function]

Create a clock provide message. This message is posted whenever an element is ready to provide a clock or lost its ability to provide a clock (maybe because it paused or became EOS).

This message is mainly used internally to manage the clock selection.

src	The object originating the message.
clock	The clock it provides
ready	TRUE if the sender can provide a clock
ret	The new provide clock message. MT safe.

`gst-message-new-clock-lost` (*src* `<gst-object>`) (*clock* `<gst-clock>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a clock lost message. This message is posted whenever the clock is not valid anymore.

If this message is posted by the pipeline, the pipeline will select a new clock again when it goes to PLAYING. It might therefore be needed to set the pipeline to PAUSED and PLAYING again.

src	The object originating the message.
clock	the clock that was lost
ret	The new clock lost message. MT safe.

`gst-message-new-custom` (*type* `<gst-message-type>`) (*src* `<gst-object>`) (*structure* `<gst-structure>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new custom-typed message. This can be used for anything not handled by other message-specific functions to pass a message to the app. The structure field can be NULL.

type	The `<gst-message-type>` to distinguish messages
src	The object originating the message.
structure	The structure for the message. The message will take ownership of the structure.
ret	The new message. MT safe.

`gst-message-new-element` (*src* `<gst-object>`) (*structure* `<gst-structure>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new element-specific message. This is meant as a generic way of allowing one-way communication from an element to an application, for example "the firewire cable was unplugged". The format of the message should be documented in the element's documentation. The structure field can be NULL.

src	The object originating the message.
structure	The structure for the message. The message will take ownership of the structure.
ret	The new element message. MT safe.

`gst-message-new-error` (*src* `<gst-object>`) (*error* `<g-error*>`) (*debug* `mchars`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new error message. The message will copy *error* and *debug*. This message is posted by element when a fatal event occured. The pipeline will probably (partially) stop. The application receiving this message should stop the pipeline.

src	The object originating the message.
error	The GError for this message.
debug	A debugging string for something or other.
ret	The new error message. MT safe.

`gst-message-new-new-clock` (*src* `<gst-object>`) (*clock* `<gst-clock>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new clock message. This message is posted whenever the pipeline selectes a new clock for the pipeline.

src	The object originating the message.
clock	the new selected clock
ret	The new new clock message. MT safe.

`gst-message-new-segment-done` (*src* `<gst-object>`) (*format* `<gst-format>`) (*position* `int64`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new segment done message. This message is posted by elements that finish playback of a segment as a result of a segment seek. This message is received by the application after all elements that posted a segment_start have posted the segment_done.

src	The object originating the message.
format	The format of the position being done
position	The position of the segment being done
ret	The new segment done message. MT safe.

`gst-message-new-segment-start` (*src* `<gst-object>`) (*format* `<gst-format>`) (*position* `int64`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new segment message. This message is posted by elements that start playback of a segment as a result of a segment seek. This message is not received by the application but is used for maintenance reasons in container elements.

src	The object originating the message.
format	The format of the position being played
position	The position of the segment being played
ret	The new segment start message. MT safe.

`gst-message-new-state-changed` (*src* `<gst-object>`) (*oldstate* `<gst-state>`) (*newstate* `<gst-state>`) (*pending* `<gst-state>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a state change message. This message is posted whenever an element changed its state.

src	the object originating the message
oldstate	the previous state
newstate	the new (current) state
pending	the pending (target) state
ret	The new state change message. MT safe.

`gst-message-new-tag` (*src* `<gst-object>`) (*tag_list* `<gst-tag-list*>`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new tag message. The message will take ownership of the tag list. The message is posted by elements that discovered a new taglist.

src	The object originating the message.
tag-list	The tag list for the message.
ret	The new tag message. MT safe.

`gst-message-new-warning` (*src* `<gst-object>`) (*error* `<g-error*>`) (*debug* `mchars`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new warning message. The message will make copies of *error* and *debug*.

src	The object originating the message.
error	The GError for this message.
debug	A debugging string for something or other.
ret	The new warning message. MT safe.

`gst-message-new-duration` (*src* `<gst-object>`) (*format* `<gst-format>`) (*duration* `int64`) ⇒ (*ret* `<gst-message>`) [Function]

Create a new duration message. This message is posted by elements that know the duration of a stream in a specific format. This message is received by bins and is used to calculate the total duration of a pipeline. Elements may post a duration message with a duration of GST_CLOCK_TIME_NONE to indicate that the duration has changed and the cached duration should be discarded. The new duration can then be retrieved via a query.

src	The object originating the message.
format	The format of the duration
duration	The new duration
ret	The new duration message. MT safe.

`gst-message-new-state-dirty` (*src* `<gst-object>`) [Function]
⇒ (*ret* `<gst-message>`)

Create a state dirty message. This message is posted whenever an element changed its state asynchronously and is used internally to update the states of container objects.

src	the object originating the message
ret	The new state dirty message. MT safe.

22 GstMiniObject

Lightweight base class for the GStreamer object hierarchy

22.1 Overview

`<gst-mini-object>` is a baseclass like `<gobject>`, but has been stripped down of features to be fast and small. It offers sub-classing and ref-counting in the same way as `<gobject>` does. It has no properties and no signal-support though.

Last reviewed on 2005-11-23 (0.9.5)

22.2 Usage

23 GstObject

Base class for the GStreamer object hierarchy

23.1 Overview

`<gst-object>` provides a root for the object hierarchy tree filed in by the GStreamer library. It is currently a thin wrapper on top of `<gobject>`. It is an abstract class that is not very usable on its own.

`<gst-object>` gives us basic refcounting, parenting functionality and locking. Most of the function are just extended for special GStreamer needs and can be found under the same name in the base class of `<gst-object>` which is `<gobject>` (e.g. `g-object-ref` becomes `gst-object-ref`).

The most interesting difference between `<gst-object>` and `<gobject>` is the "floating" reference count. A `<gobject>` is created with a reference count of 1, owned by the creator of the `<gobject>`. (The owner of a reference is the code section that has the right to call `gst-object-unref` in order to remove that reference.) A `<gst-object>` is created with a reference count of 1 also, but it isn't owned by anyone; Instead, the initial reference count of a `<gst-object>` is "floating". The floating reference can be removed by anyone at any time, by calling `gst-object-sink`. `gst-object-sink` does nothing if an object is already sunk (has no floating reference).

When you add a `<gst-element>` to its parent container, the parent container will do this: This means that the container now owns a reference to the child element (since it called `gst-object-ref`), and the child element has no floating reference.

```
gst_object_ref (GST_OBJECT (child_element));
gst_object_sink (GST_OBJECT (child_element));
```

The purpose of the floating reference is to keep the child element alive until you add it to a parent container, which then manages the lifetime of the object itself:

```
element = gst_element_factory_make (factoryname, name);
// element has one floating reference to keep it alive
gst_bin_add (GST_BIN (bin), element);
// element has one non-floating reference owned by the container
```

Another effect of this is, that calling `gst-object-unref` on a bin object, will also destoy all the `<gst-element>` objects in it. The same is true for calling `gst-bin-remove`.

Special care has to be taken for all methods that `gst-object-sink` an object since if the caller of those functions had a floating reference to the object, the object reference is now invalid.

In contrast to `<gobject>` instances, `<gst-object>` adds a name property. The functions `gst-object-set-name` and `gst-object-get-name` are used to set/get the name of the object.

Last reviewed on 2005-11-09 (0.9.4)

23.2 Usage

`<gst-object>` [Class]

This `<gobject>` class defines the following properties:

`name` The name of the object

`parent-set` (*arg0* `<gobject>`) [Signal on `<gst-object>`]

Emitted when the parent of an object is set.

`parent-unset` (*arg0* `<gobject>`) [Signal on `<gst-object>`]

Emitted when the parent of an object is unset.

`object-saved` (*arg0* `<gpointer>`) [Signal on `<gst-object>`]

Trigered whenever a new object is saved to XML. You can connect to this signal to insert custom XML tags into the core XML.

`deep-notify` (*arg0* `<gst-object>`) (*arg1* `<gparam>`) [Signal on `<gst-object>`]

The deep notify signal is used to be notified of property changes. It is typically attached to the toplevel bin to receive notifications from all the elements contained in that bin.

`gst-object-set-name` (*self* `<gst-object>`) (*name* `mchars`) [Function]
⇒ (*ret* `bool`)

`set-name` [Method]

Sets the name of *object*, or gives *object* a guaranteed unique name (if *name* is NULL). This function makes a copy of the provided name, so the caller retains ownership of the name it sent.

object a `<gst-object>`

name new name of object

ret TRUE if the name could be set. Since Objects that have a parent cannot be renamed, this function returns FALSE in those cases. MT safe. This function grabs and releases *object*'s LOCK.

`gst-object-get-name` (*self* `<gst-object>`) ⇒ (*ret* `mchars`) [Function]

`get-name` [Method]

Returns a copy of the name of *object*. Caller should `g-free` the return value after usage. For a nameless object, this returns NULL, which you can safely `g-free` as well.

object a `<gst-object>`

ret the name of *object*. `g-free` after usage. MT safe. This function grabs and releases *object*'s LOCK.

`gst-object-set-parent` (*self* `<gst-object>`) (*parent* `<gst-object>`) [Function]
⇒ (*ret* `bool`)

`set-parent` [Method]

Sets the parent of *object* to *parent*. The object's reference count will be incremented, and any floating reference will be removed (see `gst-object-sink`).

This function causes the parent-set signal to be emitted when the parent was successfully set.

object a `<gst-object>`

parent new parent of object

ret TRUE if *parent* could be set or FALSE when *object* already had a parent or *object* and *parent* are the same. MT safe. Grabs and releases *object*'s LOCK.

`gst-object-get-parent` (*self* `<gst-object>`) ⇒ (*ret* `<gst-object>`) [Function]
`get-parent` [Method]

Returns the parent of *object*. This function increases the refcount of the parent object so you should `gst-object-unref` it after usage.

object a `<gst-object>`

ret parent of *object*, this can be NULL if *object* has no parent. unref after usage. MT safe. Grabs and releases *object*'s LOCK.

`gst-object-unparent` (*self* `<gst-object>`) [Function]
`unparent` [Method]

Clear the parent of *object*, removing the associated reference. This function decreases the refcount of *object*.

MT safe. Grabs and releases *object*'s lock.

object a `<gst-object>` to unparent

`gst-object-get-name-prefix` (*self* `<gst-object>`) ⇒ (*ret* `mchars`) [Function]
`get-name-prefix` [Method]

Returns a copy of the name prefix of *object*. Caller should `g-free` the return value after usage. For a prefixless object, this returns NULL, which you can safely `g-free` as well.

object a `<gst-object>`

ret the name prefix of *object*. `g-free` after usage. MT safe. This function grabs and releases *object*'s LOCK.

`gst-object-set-name-prefix` (*self* `<gst-object>`) (*name_prefix* `mchars`) [Function]
`set-name-prefix` [Method]

Sets the name prefix of *object* to *name-prefix*. This function makes a copy of the provided name prefix, so the caller retains ownership of the name prefix it sent.

MT safe. This function grabs and releases *object*'s LOCK.

object a `<gst-object>`

name-prefix
new name prefix of *object*

`gst-object-default-error` (*self* `<gst-object>`) (*error* `<g-error*>`) (*debug* `mchars`) [Function]

`default-error` [Method]

A default error function.

The default handler will simply print the error string using g_print.

source	the `<gst-object>` that initiated the error.
error	the GError.
debug	an additional debug information string, or NULL.

`gst-object-check-uniqueness` (*list* `glist-of`) (*name* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks to see if there is any object named *name* in *list*. This function does not do any locking of any kind. You might want to protect the provided list with the lock of the owner of the list. This function will lock each `<gst-object>` in the list to compare the name, so be carefull when passing a list with a locked object.

list	a list of `<gst-object>` to check through
name	the name to search for
ret	TRUE if a `<gst-object>` named *name* does not appear in *list*, FALSE if it does. MT safe. Grabs and releases the LOCK of each object in the list.

`gst-object-has-ancestor` (*self* `<gst-object>`) (*ancestor* `<gst-object>`) ⇒ (*ret* `bool`) [Function]

`has-ancestor` [Method]

Check if *object* has an ancestor *ancestor* somewhere up in the hierarchy.

object	a `<gst-object>` to check
ancestor	a `<gst-object>` to check as ancestor
ret	TRUE if *ancestor* is an ancestor of *object*. MT safe. Grabs and releases *object*'s locks.

`gst-object-save-thyself` (*self* `<gst-object>`) (*parent* `<xml-node-ptr>`) ⇒ (*ret* `<xml-node-ptr>`) [Function]

`save-thyself` [Method]

Saves *object* into the parent XML node.

object	a `<gst-object>` to save
parent	The parent XML node to save *object* into
ret	the new xmlNodePtr with the saved object

`gst-object-restore-thyself` (*self* `<gst-object>`) (*self* `<xml-node-ptr>`) [Function]

`restore-thyself` [Method]

Restores *object* with the data from the parent XML node.

object a `<gst-object>` to load into

self The XML node to load *object* from

`gst-object-get-path-string` (*self* `<gst-object>`) ⇒ (*ret* `mchars`) [Function]
`get-path-string` [Method]

Generates a string describing the path of *object* in the object hierarchy. Only useful (or used) for debugging.

object a `<gst-object>`

ret a string describing the path of *object*. You must `g-free` the string after usage. MT safe. Grabs and releases the `<gst-object>`'s LOCK for all objects in the hierarchy.

24 GstPadTemplate

Describe the media type of a pad.

24.1 Overview

Padtemplates describe the possible media types a pad or an elementfactory can handle. This allows for both inspection of handled types before loading the element plugin as well as identifying pads on elements that are not yet created (request or sometimes pads).

Pad and PadTemplates have `<gst-caps>` attached to it to describe the media type they are capable of dealing with. `gst-pad-template-get-caps` or `gst-pad-template-caps` are used to get the caps of a padtemplate. It's not possible to modify the caps of a padtemplate after creation.

PadTemplates have a `<gst-pad-presence>` property which identifies the lifetime of the pad and that can be retrieved with `gst-pad-template-presence`. Also the direction of the pad can be retrieved from the `<gst-pad-template>` with `gst-pad-template-direction`.

The `gst-pad-template-name-template` is important for GST_PAD_REQUEST pads because it has to be used as the name in the `gst-element-request-pad-by-name` call to instantiate a pad from this template.

Padtemplates can be created with `gst-pad-template-new` or with `gst-static-pad-template-get`, which creates a `<gst-pad-template>` from a `<gst-static-pad-template>` that can be filled with the convenient `gst-static-pad-template` macro.

A padtemplate can be used to create a pad (see `gst-pad-new-from-template` or `gst-pad-new-from-static-template`) or to add to an element class (see `gst-element-class-add-pad-template`).

The following code example shows the code to create a pad from a padtemplate.

```
GstStaticPadTemplate my_template =
GST_STATIC_PAD_TEMPLATE (
  "sink",          // the name of the pad
  GST_PAD_SINK,    // the direction of the pad
  GST_PAD_ALWAYS,  // when this pad will be present
  GST_STATIC_CAPS (        // the capabilities of the padtemplate
    "audio/x-raw-int, "
      "channels = (int) [ 1, 6 ]"
  )
)
void
my_method (void)
{
  GstPad *pad;
  pad = gst_pad_new_from_static_template (&my_template, "sink");
  ...
}
```

The following example shows you how to add the padtemplate to an element class, this is usually done in the base_init of the class:

```
static void
my_element_base_init (gpointer g_class)
{
  GstElementClass *gstelement_class = GST_ELEMENT_CLASS (g_class);

  gst_element_class_add_pad_template (gstelement_class,
      gst_static_pad_template_get (&my_template));
}
```

Last reviewed on 2006-02-14 (0.10.3)

24.2 Usage

`<gst-pad-template>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`pad-created` (*arg0* `<gst-pad>`) [Signal on `<gst-pad-template>`]

This signal is fired when an element creates a pad from this template.

`gst-static-pad-template-get` (*self* `<gst-static-pad-template*>`) [Function] ⇒ (*ret* `<gst-pad-template>`)

Converts a `<gst-static-pad-template>` into a `<gst-pad-template>`.

pad-template
the static pad template

ret a new `<gst-pad-template>`.

`gst-static-pad-template-get-caps` [Function] (*self* `<gst-static-pad-template*>`) ⇒ (*ret* `<gst-caps>`)

Gets the capabilities of the static pad template.

templ a `<gst-static-pad-template>` to get capabilities of.

ret the `<gst-caps>` of the static pad template. If you need to keep a reference to the caps, take a ref (see `gst-caps-ref`).

`gst-pad-template-new` (*name_template* `mchars`) [Function] (*direction* `<gst-pad-direction>`) (*presence* `<gst-pad-presence>`) (*caps* `<gst-caps>`) ⇒ (*ret* `<gst-pad-template>`)

Creates a new pad template with a name according to the given template and with the given arguments. This functions takes ownership of the provided caps, so be sure to not use them afterwards.

name-template
the name template.

direction the `<gst-pad-direction>` of the template.

presence the `<gst-pad-presence>` of the pad.

caps a `<gst-caps>` set for the template. The caps are taken ownership of.

ret a new `<gst-pad-template>`.

`gst-pad-template-get-caps` (*self* `<gst-pad-template>`) [Function]
⇒ (*ret* `<gst-caps>`)

`get-caps` [Method]

Gets the capabilities of the pad template.

templ a `<gst-pad-template>` to get capabilities of.

ret the `<gst-caps>` of the pad template. If you need to keep a reference to the caps, take a ref (see `gst-caps-ref`).

25 GstPad

Object contained by elements that allows links to other elements

25.1 Overview

A `<gst-element>` is linked to other elements via "pads", which are extremely light-weight generic link points. After two pads are retrieved from an element with `gst-element-get-pad`, the pads can be link with `gst-pad-link`. (For quick links, you can also use `gst-element-link`, which will make the obvious link for you if it's straightforward.)

Pads are typically created from a `<gst-pad-template>` with `gst-pad-new-from-template`.

Pads have `<gst-caps>` attached to it to describe the media type they are capable of dealing with. `gst-pad-get-caps` and `gst-pad-set-caps` are used to manipulate the caps of the pads. Pads created from a pad template cannot set capabilities that are incompatible with the pad template capabilities.

Pads without pad templates can be created with `gst-pad-new`, which takes a direction and a name as an argument. If the name is NULL, then a guaranteed unique name will be assigned to it.

`gst-pad-get-parent` will retrieve the `<gst-element>` that owns the pad.

A `<gst-element>` creating a pad will typically use the various gst_pad_set_*`-function` calls to register callbacks for various events on the pads.

GstElements will use `gst-pad-push` and `gst-pad-pull-range` to push out or pull in a buffer.

To send a `<gst-event>` on a pad, use `gst-pad-send-event` and `gst-pad-push-event`.

Last reviewed on 2006-07-06 (0.10.9)

25.2 Usage

`<gst-pad>` [Class]

This `<gobject>` class defines the following properties:

`caps`	The capabilities of the pad
`direction`	The direction of the pad
`template`	The GstPadTemplate of this pad

`linked` (*arg0* `<gst-pad>`) [Signal on `<gst-pad>`]

Signals that a pad has been linked to the peer pad.

`unlinked` (*arg0* `<gst-pad>`) [Signal on `<gst-pad>`]

Signals that a pad has been unlinked from the peer pad.

`request-link` [Signal on `<gst-pad>`]

Signals that a pad connection has been requested.

`have-data` (*arg0* `<gst-mini-object>`) ⇒ `<gboolean>` [Signal on `<gst-pad>`]

Signals that new data is available on the pad. This signal is used internally for implementing pad probes. See gst_pad_add_*_probe functions.

`gst-pad-get-direction` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-pad-direction>`)
`get-direction` [Method]

Gets the direction of the pad. The direction of the pad is decided at construction time so this function does not take the LOCK.

pad	a `<gst-pad>` to get the direction of.
ret	the `<gst-pad-direction>` of the pad. MT safe.

`gst-pad-get-parent-element` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-element>`)
`get-parent-element` [Method]

Gets the parent of *pad*, cast to a `<gst-element>`. If a *pad* has no parent or its parent is not an element, return NULL.

pad	a pad
ret	The parent of the pad. The caller has a reference on the parent, so unref when you're finished with it. MT safe.

`gst-pad-get-pad-template` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-pad-template>`)
`get-pad-template` [Method]

Gets the template for *pad*.

pad	a `<gst-pad>`.
ret	the `<gst-pad-template>` from which this pad was instantiated, or '`#f`' if this pad has no template. FIXME: currently returns an unrefcounted padtemplate.

`gst-pad-link` (*self* `<gst-pad>`) (*sinkpad* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-pad-link-return>`)
`link` [Method]

Links the source pad and the sink pad.

srcpad	the source `<gst-pad>` to link.
sinkpad	the sink `<gst-pad>` to link.
ret	A result code indicating if the connection worked or what went wrong. MT Safe.

`gst-pad-unlink` (*self* `<gst-pad>`) (*sinkpad* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`unlink` [Method]

Unlinks the source pad from the sink pad. Will emit the "unlinked" signal on both pads.

srcpad	the source `<gst-pad>` to unlink.

sinkpad	the sink `<gst-pad>` to unlink.
ret	TRUE if the pads were unlinked. This function returns FALSE if the pads were not linked together. MT safe.

`gst-pad-is-linked` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`is-linked` [Method]

Checks if a *pad* is linked to another pad or not.

pad	pad to check
ret	TRUE if the pad is linked, FALSE otherwise. MT safe.

`gst-pad-can-link` (*self* `<gst-pad>`) (*sinkpad* `<gst-pad>`) [Function]
⇒ (*ret* `bool`)
`can-link` [Method]

Checks if the source pad and the sink pad can be linked. Both *srcpad* and *sinkpad* must be unlinked.

srcpad	the source `<gst-pad>` to link.
sinkpad	the sink `<gst-pad>` to link.
ret	TRUE if the pads can be linked, FALSE otherwise.

`gst-pad-get-caps` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-caps>`) [Function]
`get-caps` [Method]

Gets the capabilities this pad can produce or consume. Note that this method doesn't necessarily return the caps set by `gst-pad-set-caps` - use `<gst-pad-caps>` for that instead. gst_pad_get_caps returns all possible caps a pad can operate with, using the pad's get_caps function; this returns the pad template caps if not explicitly set.

pad	a `<gst-pad>` to get the capabilities of.
ret	a newly allocated copy of the `<gst-caps>` of this pad. MT safe.

`gst-pad-get-allowed-caps` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-caps>`) [Function]
`get-allowed-caps` [Method]

Gets the capabilities of the allowed media types that can flow through *pad* and its peer.

The allowed capabilities is calculated as the intersection of the results of calling `gst-pad-get-caps` on *pad* and its peer. The caller owns a reference on the resulting caps.

pad	a `<gst-pad>`.
ret	the allowed `<gst-caps>` of the pad link. Unref the caps when you no longer need it. This function returns NULL when *pad* has no peer. MT safe.

`gst-pad-get-negotiated-caps` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-caps>`)
`get-negotiated-caps` [Method]

Gets the capabilities of the media type that currently flows through *pad* and its peer.

This function can be used on both src and sinkpads. Note that srcpads are always negotiated before sinkpads so it is possible that the negotiated caps on the srcpad do not match the negotiated caps of the peer.

pad	a `<gst-pad>`.
ret	the negotiated `<gst-caps>` of the pad link. Unref the caps when you no longer need it. This function returns NULL when the *pad* has no peer or is not negotiated yet. MT safe.

`gst-pad-get-pad-template-caps` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-caps>`) [Function]
`get-pad-template-caps` [Method]

Gets the capabilities for *pad*'s template.

pad	a `<gst-pad>` to get the template capabilities from.
ret	the `<gst-caps>` of this pad template. If you intend to keep a reference on the caps, make a copy (see `gst-caps-copy`).

`gst-pad-set-caps` (*self* `<gst-pad>`) (*caps* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]
`set-caps` [Method]

Sets the capabilities of this pad. The caps must be fixed. Any previous caps on the pad will be unreffed. This function refs the caps so you should unref if as soon as you don't need it anymore. It is possible to set NULL caps, which will make the pad unnegotiated again.

pad	a `<gst-pad>` to set the capabilities of.
caps	a `<gst-caps>` to set.
ret	TRUE if the caps could be set. FALSE if the caps were not fixed or bad parameters were provided to this function. MT safe.

`gst-pad-get-peer` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-pad>`) [Function]
`get-peer` [Method]

Gets the peer of *pad*. This function refs the peer pad so you need to unref it after use.

pad	a `<gst-pad>` to get the peer of.
ret	the peer `<gst-pad>`. Unref after usage. MT safe.

`gst-pad-peer-get-caps` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-caps>`) [Function]
`peer-get-caps` [Method]

Gets the capabilities of the peer connected to this pad.

pad	a `<gst-pad>` to get the peer capabilities of.
ret	the `<gst-caps>` of the peer pad. This function returns a new caps, so use gst_caps_unref to get rid of it. this function returns NULL if there is no peer pad.

`gst-pad-use-fixed-caps` (*self* `<gst-pad>`) [Function]
`use-fixed-caps` [Method]

A helper function you can use that sets the *gst-pad-get-fixed-caps-func* as the getcaps function for the pad. This way the function will always return the negotiated caps or in case the pad is not negotiated, the padtemplate caps.

Use this function on a pad that, once `-set-caps` has been called on it, cannot be renegotiated to something else.

pad	the pad to use

`gst-pad-is-active` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`is-active` [Method]

Query if a pad is active

pad	the `<gst-pad>` to query
ret	TRUE if the pad is active. MT safe.

`gst-pad-set-blocked` (*self* `<gst-pad>`) (*blocked* `bool`) ⇒ (*ret* `bool`) [Function]
`set-blocked` [Method]

Blocks or unblocks the dataflow on a pad. This function is a shortcut for `gst-pad-set-blocked-async` with a NULL callback.

pad	the `<gst-pad>` to block or unblock
blocked	boolean indicating we should block or unblock
ret	TRUE if the pad could be blocked. This function can fail if the wrong parameters were passed or the pad was already in the requested state. MT safe.

`gst-pad-set-blocked-async` (*self* `<gst-pad>`) (*blocked* `bool`) [Function]
(*callback* `<gst-pad-block-callback>`) (*user_data* `<gpointer>`)
⇒ (*ret* `bool`)
`set-blocked-async` [Method]

Blocks or unblocks the dataflow on a pad. The provided callback is called when the operation succeeds; this happens right before the next attempt at pushing a buffer on the pad.

This can take a while as the pad can only become blocked when real dataflow is happening. When the pipeline is stalled, for example in PAUSED, this can take an indeterminate amount of time. You can pass NULL as the callback to make this call block. Be careful with this blocking call as it might not return for reasons stated above.

pad	the `<gst-pad>` to block or unblock
blocked	boolean indicating whether the pad should be blocked or unblocked
callback	`<gst-pad-block-callback>` that will be called when the operation succeeds
user-data	user data passed to the callback

ret	TRUE if the pad could be blocked. This function can fail if the wrong parameters were passed or the pad was already in the requested state. MT safe.

`gst-pad-is-blocked` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`is-blocked` [Method]

Checks if the pad is blocked or not. This function returns the last requested state of the pad. It is not certain that the pad is actually blocking at this point (see `gst-pad-is-blocking`).

pad	the `<gst-pad>` to query
ret	TRUE if the pad is blocked. MT safe.

`gst-pad-add-data-probe` (*self* `<gst-pad>`) (*handler* `<g-callback>`) (*data* `<gpointer>`) ⇒ (*ret* `unsigned-long`) [Function]
`add-data-probe` [Method]

Adds a "data probe" to a pad. This function will be called whenever data passes through a pad. In this case data means both events and buffers. The probe will be called with the data as an argument, meaning *handler* should have the same callback signature as the 'have-data' signal of `<gst-pad>`. Note that the data will have a reference count greater than 1, so it will be immutable – you must not change it.

For source pads, the probe will be called after the blocking function, if any (see `gst-pad-set-blocked-async`), but before looking up the peer to chain to. For sink pads, the probe function will be called before configuring the sink with new caps, if any, and before calling the pad's chain function.

Your data probe should return TRUE to let the data continue to flow, or FALSE to drop it. Dropping data is rarely useful, but occasionally comes in handy with events.

Although probes are implemented internally by connecting *handler* to the have-data signal on the pad, if you want to remove a probe it is insufficient to only call g_signal_handler_disconnect on the returned handler id. To remove a probe, use the appropriate function, such as `gst-pad-remove-data-probe`.

pad	pad to add the data probe handler to
handler	function to call when data is passed over pad
data	data to pass along with the handler
ret	The handler id.

`gst-pad-add-buffer-probe` (*self* `<gst-pad>`) (*handler* `<g-callback>`) (*data* `<gpointer>`) ⇒ (*ret* `unsigned-long`) [Function]
`add-buffer-probe` [Method]

Adds a probe that will be called for all buffers passing through a pad. See `gst-pad-add-data-probe` for more information.

pad	pad to add the buffer probe handler to
handler	function to call when data is passed over pad
data	data to pass along with the handler
ret	The handler id

`gst-pad-add-event-probe` (*self* `<gst-pad>`) (*handler* `<g-callback>`) (*data* `<gpointer>`) ⇒ (*ret* `unsigned-long`) [Function]

`add-event-probe` [Method]

Adds a probe that will be called for all events passing through a pad. See `gst-pad-add-data-probe` for more information.

pad	pad to add the event probe handler to
handler	function to call when data is passed over pad
data	data to pass along with the handler
ret	The handler id

`gst-pad-remove-data-probe` (*self* `<gst-pad>`) (*handler_id* `unsigned-int`) [Function]

`remove-data-probe` [Method]

Removes a data probe from *pad*.

pad	pad to remove the data probe handler from
handler-id	handler id returned from gst_pad_add_data_probe

`gst-pad-remove-buffer-probe` (*self* `<gst-pad>`) (*handler_id* `unsigned-int`) [Function]

`remove-buffer-probe` [Method]

Removes a buffer probe from *pad*.

pad	pad to remove the buffer probe handler from
handler-id	handler id returned from gst_pad_add_buffer_probe

`gst-pad-remove-event-probe` (*self* `<gst-pad>`) (*handler_id* `unsigned-int`) [Function]

`remove-event-probe` [Method]

Removes an event probe from *pad*.

pad	pad to remove the event probe handler from
handler-id	handler id returned from gst_pad_add_event_probe

`gst-pad-new` (*name* `mchars`) (*direction* `<gst-pad-direction>`) ⇒ (*ret* `<gst-pad>`) [Function]

Creates a new pad with the given name in the given direction. If name is NULL, a guaranteed unique name (across all pads) will be assigned. This function makes a copy of the name so you can safely free the name.

name	the name of the new pad.
direction	the `<gst-pad-direction>` of the pad.
ret	a new `<gst-pad>`, or NULL in case of an error. MT safe.

`gst-pad-new-from-template` (*templ* `<gst-pad-template>`) (*name* `mchars`) ⇒ (*ret* `<gst-pad>`) [Function]

Creates a new pad with the given name from the given template. If name is NULL, a guaranteed unique name (across all pads) will be assigned. This function makes a copy of the name so you can safely free the name.

templ	the pad template to use
name	the name of the element
ret	a new `<gst-pad>`, or NULL in case of an error.

`gst-pad-new-from-static-template` (*templ* `<gst-static-pad-template*>`) (*name* `mchars`) ⇒ (*ret* `<gst-pad>`) [Function]

Creates a new pad with the given name from the given static template. If name is NULL, a guaranteed unique name (across all pads) will be assigned. This function makes a copy of the name so you can safely free the name.

templ	the `<gst-static-pad-template>` to use
name	the name of the element
ret	a new `<gst-pad>`, or NULL in case of an error.

`gst-pad-alloc-buffer` (*self* `<gst-pad>`) (*offset* `unsigned-int64`) (*size* `int`) (*caps* `<gst-caps>`) (*buf* `<gst-buffer**>`) ⇒ (*ret* `<gst-flow-return>`) [Function]

`alloc-buffer` [Method]

Allocates a new, empty buffer optimized to push to pad *pad*. This function only works if *pad* is a source pad and has a peer.

A new, empty `<gst-buffer>` will be put in the *buf* argument. You need to check the caps of the buffer after performing this function and renegotiate to the format if needed.

pad	a source `<gst-pad>`
offset	the offset of the new buffer in the stream
size	the size of the new buffer
caps	the caps of the new buffer
buf	a newly allocated buffer
ret	a result code indicating success of the operation. Any result code other than `<gst-flow-ok>` is an error and *buf* should not be used. An error can occur if the pad is not connected or when the downstream peer elements cannot provide an acceptable buffer. MT safe.

`gst-pad-alloc-buffer-and-set-caps` (*self* `<gst-pad>`) (*offset* `unsigned-int64`) (*size* `int`) (*caps* `<gst-caps>`) (*buf* `<gst-buffer**>`) ⇒ (*ret* `<gst-flow-return>`) [Function]

`alloc-buffer-and-set-caps` [Method]

In addition to the function `gst-pad-alloc-buffer`, this function automatically calls `gst-pad-set-caps` when the caps of the newly allocated buffer are different from the *pad* caps.

pad	a source `<gst-pad>`
offset	the offset of the new buffer in the stream
size	the size of the new buffer
caps	the caps of the new buffer
buf	a newly allocated buffer
ret	a result code indicating success of the operation. Any result code other than `<gst-flow-ok>` is an error and *buf* should not be used. An error can occur if the pad is not connected or when the downstream peer elements cannot provide an acceptable buffer. MT safe.

`gst-pad-set-chain-function` (*self* `<gst-pad>`) (*chain-function* `scm`) [Function]

`set-chain-function` [Method]

Sets the given chain function for the pad. The chain function is called to process a `<gst-buffer>` input buffer. see `<gst-pad-chain-function>` for more details.

pad	a sink `<gst-pad>`.
chain	the `<gst-pad-chain-function>` to set.

`gst-pad-get-range` (*self* `<gst-pad>`) (*offset* `unsigned-int64`) (*size* `unsigned-int`) (*buffer* `<gst-buffer**>`) ⇒ (*ret* `<gst-flow-return>`) [Function]

`get-range` [Method]

When *pad* is flushing this function returns `<gst-flow-wrong-state>` immediatly.

Calls the getrange function of *pad*, see `<gst-pad-get-range-function>` for a description of a getrange function. If *pad* has no getrange function installed (see `gst-pad-set-getrange-function`) this function returns `<gst-flow-not-supported>`.

buffer's caps must either be unset or the same as what is already configured on *pad*. Renegotiation within a running pull-mode pipeline is not supported.

This is a lowlevel function. Usualy `gst-pad-pull-range` is used.

pad	a src `<gst-pad>`, returns `<gst-flow-error>` if not.
offset	The start offset of the buffer
size	The length of the buffer
buffer	a pointer to hold the `<gst-buffer>`, returns `<gst-flow-error>` if '`#f`'.
ret	a `<gst-flow-return>` from the pad. MT safe.

gst-pad-set-getrange-function (*self* `<gst-pad>`) (*get-function* `scm`) [Function]
set-getrange-function [Method]

Sets the given getrange function for the pad. The getrange function is called to produce a new `<gst-buffer>` to start the processing pipeline. see `<gst-pad-get-range-function>` for a description of the getrange function.

pad	a source `<gst-pad>`.
get	the `<gst-pad-get-range-function>` to set.

gst-pad-accept-caps (*self* `<gst-pad>`) (*caps* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]
accept-caps [Method]

Check if the given pad accepts the caps.

pad	a `<gst-pad>` to check
caps	a `<gst-caps>` to check on the pad
ret	TRUE if the pad can accept the caps.

gst-pad-proxy-getcaps (*self* `<gst-pad>`) ⇒ (*ret* `<gst-caps>`) [Function]
proxy-getcaps [Method]

Calls `gst-pad-get-allowed-caps` for every other pad belonging to the same element as *pad*, and returns the intersection of the results.

This function is useful as a default getcaps function for an element that can handle any stream format, but requires all its pads to have the same caps. Two such elements are tee and aggregator.

pad	a `<gst-pad>` to proxy.
ret	the intersection of the other pads' allowed caps.

gst-pad-set-setcaps-function (*self* `<gst-pad>`) (*setcaps-function* `scm`) [Function]
set-setcaps-function [Method]

Sets the given setcaps function for the pad. The setcaps function will be called whenever a buffer with a new media type is pushed or pulled from the pad. The pad/element needs to update its internal structures to process the new media type. If this new type is not acceptable, the setcaps function should return FALSE.

pad	a `<gst-pad>`.
setcaps	the `<gst-pad-set-caps-function>` to set.

gst-pad-proxy-setcaps (*self* `<gst-pad>`) (*caps* `<gst-caps>`) ⇒ (*ret* `bool`) [Function]
proxy-setcaps [Method]

Calls `gst-pad-set-caps` for every other pad belonging to the same element as *pad*. If `gst-pad-set-caps` fails on any pad, the proxy setcaps fails. May be used only during negotiation.

pad	a `<gst-pad>` to proxy from

caps	the `<gst-caps>` to link with
ret	TRUE if sucessful

`gst-pad-fixate-caps` (*self* `<gst-pad>`) (*caps* `<gst-caps>`) [Function]
`fixate-caps` [Method]

Fixate a caps on the given pad. Modifies the caps in place, so you should make sure that the caps are actually writable (see `gst-caps-make-writable`).

pad	a `<gst-pad>` to fixate
caps	the `<gst-caps>` to fixate

`gst-pad-get-fixed-caps-func` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `<gst-caps>`)
`get-fixed-caps-func` [Method]

A helper function you can use as a GetCaps function that will return the currently negotiated caps or the padtemplate when NULL.

pad	the pad to use
ret	The currently negotiated caps or the padtemplate.

`gst-pad-peer-accept-caps` (*self* `<gst-pad>`) (*caps* `<gst-caps>`) [Function]
⇒ (*ret* `bool`)
`peer-accept-caps` [Method]

Check if the peer of *pad* accepts *caps*. If *pad* has no peer, this function returns TRUE.

pad	a `<gst-pad>` to check the peer of
caps	a `<gst-caps>` to check on the pad
ret	TRUE if the peer of *pad* can accept the caps or *pad* has no peer.

`gst-pad-push` (*self* `<gst-pad>`) (*buffer* `<gst-buffer>`) [Function]
⇒ (*ret* `<gst-flow-return>`)
`push` [Method]

Pushes a buffer to the peer of *pad*.

This function will call an installed pad block before triggering any installed pad probes.

If the caps on *buffer* are different from the currently configured caps on *pad*, this function will call any installed setcaps function on *pad* (see `gst-pad-set-setcaps-function`). In case of failure to renegotiate the new format, this function returns `<gst-flow-not-negotiated>`.

The function proceeds calling `gst-pad-chain` on the peer pad and returns the value from that function. If *pad* has no peer, `<gst-flow-not-linked>` will be returned.

In all cases, success or failure, the caller loses its reference to *buffer* after calling this function.

pad	a source `<gst-pad>`, returns `<gst-flow-error>` if not.
buffer	the `<gst-buffer>` to push returns GST_FLOW_ERROR if not.
ret	a `<gst-flow-return>` from the peer pad. MT safe.

`gst-pad-push-event` (*self* `<gst-pad>`) (*event* `<gst-event>`) [Function]
⇒ (*ret* `bool`)

`push-event` [Method]

Sends the event to the peer of the given pad. This function is mainly used by elements to send events to their peer elements.

This function takes owership of the provided event so you should `gst-event-ref` it if you want to reuse the event after this call.

pad	a `<gst-pad>` to push the event to.
event	the `<gst-event>` to send to the pad.
ret	TRUE if the event was handled. MT safe.

`gst-pad-check-pull-range` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]

`check-pull-range` [Method]

Checks if a `gst-pad-pull-range` can be performed on the peer source pad. This function is used by plugins that want to check if they can use random access on the peer source pad.

The peer sourcepad can implement a custom `<gst-pad-check-get-range-function>` if it needs to perform some logic to determine if pull_range is possible.

pad	a sink `<gst-pad>`.
ret	a gboolean with the result. MT safe.

`gst-pad-pull-range` (*self* `<gst-pad>`) (*offset* `unsigned-int64`) [Function]
(*size* `unsigned-int`) (*buffer* `<gst-buffer**>`)
⇒ (*ret* `<gst-flow-return>`)

`pull-range` [Method]

Pulls a *buffer* from the peer pad.

This function will first trigger the pad block signal if it was installed.

When *pad* is not linked `<gst-flow-not-linked>` is returned else this function returns the result of `gst-pad-get-range` on the peer pad. See `gst-pad-get-range` for a list of return values and for the semantics of the arguments of this function.

buffer's caps must either be unset or the same as what is already configured on *pad*. Renegotiation within a running pull-mode pipeline is not supported.

pad	a sink `<gst-pad>`, returns GST_FLOW_ERROR if not.
offset	The start offset of the buffer
size	The length of the buffer
buffer	a pointer to hold the `<gst-buffer>`, returns GST_FLOW_ERROR if '`#f`'.
ret	a `<gst-flow-return>` from the peer pad. When this function returns `<gst-flow-ok>`, *buffer* will contain a valid `<gst-buffer>` that should be freed with `gst-buffer-unref` after usage. *buffer* may not be used or freed when any other return value than `<gst-flow-ok>` is returned. MT safe.

`gst-pad-activate-pull` (*self* `<gst-pad>`) (*active* `bool`) ⇒ (*ret* `bool`) [Function]
`activate-pull` [Method]

Activates or deactivates the given pad in pull mode via dispatching to the pad's activatepullfunc. For use from within pad activation functions only. When called on sink pads, will first proxy the call to the peer pad, which is expected to activate its internally linked pads from within its activate_pull function.

If you don't know what this is, you probably don't want to call it.

pad	the `<gst-pad>` to activate or deactivate.
active	whether or not the pad should be active.
ret	TRUE if the operation was successful. MT safe.

`gst-pad-activate-push` (*self* `<gst-pad>`) (*active* `bool`) ⇒ (*ret* `bool`) [Function]
`activate-push` [Method]

Activates or deactivates the given pad in push mode via dispatching to the pad's activatepushfunc. For use from within pad activation functions only.

If you don't know what this is, you probably don't want to call it.

pad	the `<gst-pad>` to activate or deactivate.
active	whether the pad should be active or not.
ret	'`#t`' if the operation was successful. MT safe.

`gst-pad-send-event` (*self* `<gst-pad>`) (*event* `<gst-event>`) [Function]
⇒ (*ret* `bool`)
`send-event` [Method]

Sends the event to the pad. This function can be used by applications to send events in the pipeline.

If *pad* is a source pad, *event* should be an upstream event. If *pad* is a sink pad, *event* should be a downstream event. For example, you would not send a `<gst-event-eos>` on a src pad; EOS events only propagate downstream. Furthermore, some downstream events have to be serialized with data flow, like EOS, while some can travel out-of-band, like `<gst-event-flush-start>`. If the event needs to be serialized with data flow, this function will take the pad's stream lock while calling its event function.

To find out whether an event type is upstream, downstream, or downstream and serialized, see `<gst-event-type-flags>`, `gst-event-type-get-flags`, `<gst-event-is-upstream>`, `<gst-event-is-downstream>`, and `<gst-event-is-serialized>`. Note that in practice that an application or plugin doesn't need to bother itself with this information; the core handles all necessary locks and checks.

This function takes owership of the provided event so you should `gst-event-ref` it if you want to reuse the event after this call.

pad	a `<gst-pad>` to send the event to.
event	the `<gst-event>` to send to the pad.
ret	TRUE if the event was handled.

`gst-pad-event-default` (*self* `<gst-pad>`) (*event* `<gst-event>`) ⇒ (*ret* `bool`) [Function]
`event-default` [Method]

Invokes the default event handler for the given pad. End-of-stream and discontinuity events are handled specially, and then the event is sent to all pads internally linked to *pad*. Note that if there are many possible sink pads that are internally linked to *pad*, only one will be sent an event. Multi-sinkpad elements should implement custom event handlers.

pad	a `<gst-pad>` to call the default event handler on.
event	the `<gst-event>` to handle.
ret	TRUE if the event was sent succesfully.

`gst-pad-query` (*self* `<gst-pad>`) (*query* `<gst-query>`) ⇒ (*ret* `bool`) [Function]
`query` [Method]

Dispatches a query to a pad. The query should have been allocated by the caller via one of the type-specific allocation functions in gstquery.h. The element is responsible for filling the query with an appropriate response, which should then be parsed with a type-specific query parsing function.

Again, the caller is responsible for both the allocation and deallocation of the query structure.

pad	a `<gst-pad>` to invoke the default query on.
query	the `<gst-query>` to perform.
ret	TRUE if the query could be performed.

`gst-pad-query-default` (*self* `<gst-pad>`) (*query* `<gst-query>`) ⇒ (*ret* `bool`) [Function]
`query-default` [Method]

Invokes the default query handler for the given pad. The query is sent to all pads internally linked to *pad*. Note that if there are many possible sink pads that are internally linked to *pad*, only one will be sent the query. Multi-sinkpad elements should implement custom query handlers.

pad	a `<gst-pad>` to call the default query handler on.
query	the `<gst-query>` to handle.
ret	TRUE if the query was performed succesfully.

`gst-pad-query-position` (*self* `<gst-pad>`) (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*cur* `int64`) [Function]
`query-position` [Method]

Queries a pad for the stream position.

pad	a `<gst-pad>` to invoke the position query on.
format	a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.

cur A location in which to store the current position, or NULL.

ret TRUE if the query could be performed.

`gst-pad-query-duration` (*self* `<gst-pad>`) (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*duration* `int64`) [Function]

`query-duration` [Method]

Queries a pad for the total stream duration.

pad a `<gst-pad>` to invoke the duration query on.

format a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.

duration A location in which to store the total duration, or NULL.

ret TRUE if the query could be performed.

`gst-pad-query-convert` (*self* `<gst-pad>`) (*src_format* `<gst-format>`) (*src_val* `int64`) (*dest_format* `<gst-format*>`) ⇒ (*ret* `bool`) (*dest_val* `int64`) [Function]

`query-convert` [Method]

Queries a pad to convert *src-val* in *src-format* to *dest-format*.

pad a `<gst-pad>` to invoke the convert query on.

src-format a `<gst-format>` to convert from.

src-val a value to convert.

dest-format
a pointer to the `<gst-format>` to convert to.

dest-val a pointer to the result.

ret TRUE if the query could be performed.

`gst-pad-query-peer-position` (*self* `<gst-pad>`) (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*cur* `int64`) [Function]

`query-peer-position` [Method]

Queries the peer of a given sink pad for the stream position.

pad a `<gst-pad>` on whose peer to invoke the position query on. Must be a sink pad.

format a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.

cur A location in which to store the current position, or NULL.

ret TRUE if the query could be performed.

`gst-pad-query-peer-duration` (*self* `<gst-pad>`) (*format* `<gst-format*>`) ⇒ (*ret* `bool`) (*duration* `int64`) [Function]

`query-peer-duration` [Method]

Queries the peer pad of a given sink pad for the total stream duration.

pad a `<gst-pad>` on whose peer pad to invoke the duration query on. Must be a sink pad.

format a pointer to the `<gst-format>` asked for. On return contains the `<gst-format>` used.

duration A location in which to store the total duration, or NULL.

ret TRUE if the query could be performed.

`gst-pad-query-peer-convert` (*self* `<gst-pad>`) (*src_format* `<gst-format>`) (*src_val* `int64`) (*dest_format* `<gst-format*>`) ⇒ (*ret* `bool`) (*dest_val* `int64`) [Function]

`query-peer-convert` [Method]

Queries the peer pad of a given sink pad to convert *src-val* in *src-format* to *dest-format*.

pad a `<gst-pad>`, on whose peer pad to invoke the convert query on. Must be a sink pad.

src-format a `<gst-format>` to convert from.

src-val a value to convert.

dest-format a pointer to the `<gst-format>` to convert to.

dest-val a pointer to the result.

ret TRUE if the query could be performed.

`gst-pad-get-query-types` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-query-type*>`) [Function]

`get-query-types` [Method]

Get an array of supported queries that can be performed on this pad.

pad a `<gst-pad>`.

ret a zero-terminated array of `<gst-query-type>`.

`gst-pad-get-query-types-default` (*self* `<gst-pad>`) ⇒ (*ret* `<gst-query-type*>`) [Function]

`get-query-types-default` [Method]

Invoke the default dispatcher for the query types on the pad.

pad a `<gst-pad>`.

ret an zero-terminated array of `<gst-query-type>`, or NULL if none of the internally-linked pads has a query types function.

`gst-pad-get-internal-links` (*self* `<gst-pad>`) ⇒ (*ret* `glist-of`) [Function]

`get-internal-links` [Method]

Gets a list of pads to which the given pad is linked to inside of the parent element. The caller must free this list after use.

pad the `<gst-pad>` to get the internal links of.

ret a newly allocated `<g-list>` of pads. Not MT safe.

`gst-pad-get-internal-links-default` (*self* `<gst-pad>`) [Function]
⇒ (*ret* `glist-of`)
`get-internal-links-default` [Method]

Gets a list of pads to which the given pad is linked to inside of the parent element. This is the default handler, and thus returns a list of all of the pads inside the parent element with opposite direction. The caller must free this list after use.

pad	the `<gst-pad>` to get the internal links of.
ret	a newly allocated `<g-list>` of pads, or NULL if the pad has no parent. Not MT safe.

`gst-pad-load-and-link` (*self* `<xml-node-ptr>`) [Function]
(*parent* `<gst-object>`)

Reads the pad definition from the XML node and links the given pad in the element to a pad of an element up in the hierarchy.

self	an `<xml-node-ptr>` to read the description from.
parent	the `<gst-object>` element that owns the pad.

`gst-pad-dispatcher` (*self* `<gst-pad>`) [Function]
(*dispatch* `<gst-pad-dispatcher-function>`) (*data* `<gpointer>`)
⇒ (*ret* `bool`)
`dispatcher` [Method]

Invokes the given dispatcher function on each respective peer of all pads that are internally linked to the given pad. The GstPadDispatcherFunction should return TRUE when no further pads need to be processed.

pad	a `<gst-pad>` to dispatch.
dispatch	the `<gst-dispatcher-function>` to call.
data	gpointer user data passed to the dispatcher function.
ret	TRUE if one of the dispatcher functions returned TRUE.

`gst-pad-chain` (*self* `<gst-pad>`) (*buffer* `<gst-buffer>`) [Function]
⇒ (*ret* `<gst-flow-return>`)
`chain` [Method]

Chain a buffer to *pad*.

The function returns `<gst-flow-wrong-state>` if the pad was flushing.

If the caps on *buffer* are different from the current caps on *pad*, this function will call any setcaps function (see `gst-pad-set-setcaps-function`) installed on *pad*. If the new caps are not acceptable for *pad*, this function returns `<gst-flow-not-negotiated>`.

The function proceeds calling the chain function installed on *pad* (see `gst-pad-set-chain-function`) and the return value of that function is returned to the caller. `<gst-flow-not-supported>` is returned if *pad* has no chain function.

In all cases, success or failure, the caller loses its reference to *buffer* after calling this function.

pad	a sink `<gst-pad>`, returns GST_FLOW_ERROR if not.
buffer	the `<gst-buffer>` to send, return GST_FLOW_ERROR if not.
ret	a `<gst-flow-return>` from the pad. MT safe.

`gst-pad-start-task` (*self* `<gst-pad>`) (*func* `<gst-task-function>`) (*data* `<gpointer>`) ⇒ (*ret* `bool`) [Function]
`start-task` [Method]

Starts a task that repeatedly calls *func* with *data*. This function is mostly used in pad activation functions to start the dataflow. The `<gst-pad-stream-lock>` of *pad* will automatically be acquired before *func* is called.

pad	the `<gst-pad>` to start the task of
func	the task function to call
data	data passed to the task function
ret	a '`#t`' if the task could be started.

`gst-pad-pause-task` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`pause-task` [Method]

Pause the task of *pad*. This function will also wait until the function executed by the task is finished if this function is not called from the task function.

pad	the `<gst-pad>` to pause the task of
ret	a TRUE if the task could be paused or FALSE when the pad has no task.

`gst-pad-stop-task` (*self* `<gst-pad>`) ⇒ (*ret* `bool`) [Function]
`stop-task` [Method]

Stop the task of *pad*. This function will also make sure that the function executed by the task will effectively stop if not called from the GstTaskFunction.

This function will deadlock if called from the GstTaskFunction of the task. Use `gst-task-pause` instead.

Regardless of whether the pad has a task, the stream lock is acquired and released so as to ensure that streaming through this pad has finished.

pad	the `<gst-pad>` to stop the task of
ret	a TRUE if the task could be stopped or FALSE on error.

`gst-pad-set-active` (*self* `<gst-pad>`) (*active* `bool`) ⇒ (*ret* `bool`) [Function]
`set-active` [Method]

Activates or deactivates the given pad. Normally called from within core state change functions.

If *active*, makes sure the pad is active. If it is already active, either in push or pull mode, just return. Otherwise dispatches to the pad's activate function to perform the actual activation.

If not *active*, checks the pad's current mode and calls `gst-pad-activate-push` or `gst-pad-activate-pull`, as appropriate, with a FALSE argument.

pad	the `<gst-pad>` to activate or deactivate.

active	whether or not the pad should be active.
ret	`#t` if the operation was successful. MT safe.

26 GstParse

Get a pipeline from a text pipeline description

26.1 Overview

These function allow to create a pipeline based on the syntax used in the gst-launch utillity.

26.2 Usage

`gst-parse-error-quark` ⇒ (*ret* `unsigned-int`) [Function]

Get the error quark used by the parsing subsystem.

ret the quark of the parse errors.

`gst-parse-launch` (*pipeline_description* `mchars`) ⇒ (*ret* `<gst-element>`) [Function]

Create a new pipeline based on command line syntax. Please note that you might get a return value that is not '`#f`' even though the *error* is set. In this case there was a recoverable parsing error and you can try to play the pipeline.

pipeline-description
the command line describing the pipeline

error the error message in case of an erroneous pipeline.

ret a new element on success, '`#f`' on failure. If more than one toplevel element is specified by the *pipeline-description*, all elements are put into a `<gst-pipeline>`, which than is returned.

`gst-parse-bin-from-description` (*bin_description* `mchars`) (*ghost_unconnected_pads* `bool`) ⇒ (*ret* `<gst-element>`) [Function]

This is a convenience wrapper around `gst-parse-launch` to create a `<gst-bin>` from a gst-launch-style pipeline description. See `gst-parse-launch` and the gst-launch man page for details about the syntax. Ghost pads on the bin for unconnected source or sink pads within the bin can automatically be created (but only a maximum of one ghost pad for each direction will be created; if you expect multiple unconnected source pads or multiple unconnected sink pads and want them all ghosted, you will have to create the ghost pads yourself).

bin-description
command line describing the bin

ghost-unconnected-pads
whether to automatically create ghost pads for unconnected source or sink pads within the bin

err where to store the error message in case of an error, or NULL

ret a newly-created bin, or NULL if an error occurred.

Since 0.10.3

27 GstPipeline

Top-level bin with clocking and bus management functionality.

27.1 Overview

A `<gst-pipeline>` is a special `<gst-bin>` used as the toplevel container for the filter graph. The `<gst-pipeline>` will manage the selection and distribution of a global `<gst-clock>` as well as provide a `<gst-bus>` to the application. It will also implement a default behaviour for managing seek events (see `gst-element-seek`).

`gst-pipeline-new` is used to create a pipeline. when you are done with the pipeline, use `gst-object-unref` to free its resources including all added `<gst-element>` objects (if not otherwise referenced).

Elements are added and removed from the pipeline using the `<gst-bin>` methods like `gst-bin-add` and `gst-bin-remove` (see `<gst-bin>`).

Before changing the state of the `<gst-pipeline>` (see `<gst-element>`) a `<gst-bus>` can be retrieved with `gst-pipeline-get-bus`. This bus can then be used to receive `<gst-message>` from the elements in the pipeline.

By default, a `<gst-pipeline>` will automatically flush the pending `<gst-bus>` messages when going to the NULL state to ensure that no circular references exist when no messages are read from the `<gst-bus>`. This behaviour can be changed with `gst-pipeline-set-auto-flush-bus`.

When the `<gst-pipeline>` performs the PAUSED to PLAYING state change it will select a clock for the elements. The clock selection algorithm will by default select a clock provided by an element that is most upstream (closest to the source). For live pipelines (ones that return `<gst-state-change-no-preroll>` from the `gst-element-set-state` call) this will select the clock provided by the live source. For normal pipelines this will select a clock provided by the sinks (most likely the audio sink). If no element provides a clock, a default `<gst-system-clock>` is used.

The clock selection can be controlled with the `gst-pipeline-use-clock` method, which will enforce a given clock on the pipeline. With `gst-pipeline-auto-clock` the default clock selection algorithm can be restored.

A `<gst-pipeline>` maintains a stream time for the elements. The stream time is defined as the difference between the current clock time and the base time. When the pipeline goes to READY or a flushing seek is performed on it, the stream time is reset to 0. When the pipeline is set from PLAYING to PAUSED, the current clock time is sampled and used to configure the base time for the elements when the pipeline is set to PLAYING again. This default behaviour can be changed with the `gst-pipeline-set-new-stream-time` method.

When sending a flushing seek event to a GstPipeline (see `gst-element-seek`), it will make sure that the pipeline is properly PAUSED and resumed as well as set the new stream time to 0 when the seek succeeded.

Last reviewed on 2006-03-12 (0.10.5)

27.2 Usage

`<gst-pipeline>` [Class]

This `<gobject>` class defines the following properties:

`delay` Expected delay needed for elements to spin up to PLAYING in nanoseconds

`auto-flush-bus`
Whether to automatically flush the pipeline's bus when going from READY into NULL state

`gst-pipeline-new` (*name* `mchars`) ⇒ (*ret* `<gst-element>`) [Function]

Create a new pipeline with the given name.

name name of new pipeline

ret newly created GstPipeline MT safe.

`gst-pipeline-get-bus` (*self* `<gst-pipeline>`) ⇒ (*ret* `<gst-bus>`) [Function]
`get-bus` [Method]

Gets the `<gst-bus>` of *pipeline*.

pipeline a `<gst-pipeline>`

ret a `<gst-bus>`, unref after usage. MT safe.

`gst-pipeline-set-clock` (*self* `<gst-pipeline>`) [Function]
(*clock* `<gst-clock>`) ⇒ (*ret* `bool`)
`set-clock` [Method]

Set the clock for *pipeline*. The clock will be distributed to all the elements managed by the pipeline.

pipeline a `<gst-pipeline>`

clock the clock to set

ret TRUE if the clock could be set on the pipeline. FALSE if some element did not accept the clock. MT safe.

`gst-pipeline-get-clock` (*self* `<gst-pipeline>`) [Function]
⇒ (*ret* `<gst-clock>`)
`get-clock` [Method]

Gets the current clock used by *pipeline*.

pipeline a `<gst-pipeline>`

ret a `<gst-clock>`, unref after usage.

`gst-pipeline-use-clock` (*self* `<gst-pipeline>`) [Function]
(*clock* `<gst-clock>`)
`use-clock` [Method]

Force *pipeline* to use the given *clock*. The pipeline will always use the given clock even if new clock providers are added to this pipeline.

If *clock* is NULL all clocking will be disabled which will make the pipeline run as fast as possible.

MT safe.

pipeline	a `<gst-pipeline>`
clock	the clock to use

`gst-pipeline-auto-clock` (*self* `<gst-pipeline>`) [Function]
`auto-clock` [Method]

Let *pipeline* select a clock automatically. This is the default behaviour.

Use this function if you previous forced a fixed clock with `gst-pipeline-use-clock` and want to restore the default pipeline clock selection algorithm.

MT safe.

pipeline	a `<gst-pipeline>`

`gst-pipeline-set-new-stream-time` (*self* `<gst-pipeline>`) [Function]
(*time* `unsigned-long-long`)
`set-new-stream-time` [Method]

Set the new stream time of *pipeline* to *time*. The stream time is used to set the base time on the elements (see `gst-element-set-base-time`) in the PAUSED->PLAYING state transition.

Setting *time* to `<gst-clock-time-none>` will disable the pipeline's management of element base time. The application will then be responsible for performing base time distribution. This is sometimes useful if you want to synchronize capture from multiple pipelines, and you can also ensure that the pipelines have the same clock.

MT safe.

pipeline	a `<gst-pipeline>`
time	the new stream time to set

`gst-pipeline-get-last-stream-time` (*self* `<gst-pipeline>`) [Function]
⇒ (*ret* `unsigned-long-long`)
`get-last-stream-time` [Method]

Gets the last stream time of *pipeline*. If the pipeline is PLAYING, the returned time is the stream time used to configure the element's base time in the PAUSED->PLAYING state. If the pipeline is PAUSED, the returned time is the stream time when the pipeline was paused.

This function returns `<gst-clock-time-none>` if the pipeline was configured to not handle the management of the element's base time (see `gst-pipeline-set-new-stream-time`).

pipeline	a `<gst-pipeline>`
ret	a `<gst-clock-time>`. MT safe.

`gst-pipeline-set-auto-flush-bus` (*self* `<gst-pipeline>`) (*auto_flush* `bool`) [Function]

`set-auto-flush-bus` [Method]

Usually, when a pipeline goes from READY to NULL state, it automatically flushes all pending messages on the bus, which is done for refcounting purposes, to break circular references.

This means that applications that update state using (async) bus messages (e.g. do certain things when a pipeline goes from PAUSED to READY) might not get to see messages when the pipeline is shut down, because they might be flushed before they can be dispatched in the main thread. This behaviour can be disabled using this function.

It is important that all messages on the bus are handled when the automatic flushing is disabled else memory leaks will be introduced.

MT safe.

pipeline	a `<gst-pipeline>`
auto-flush	whether or not to automatically flush the bus when the pipeline goes from READY to NULL state

Since 0.10.4

`gst-pipeline-get-auto-flush-bus` (*self* `<gst-pipeline>`) ⇒ (*ret* `bool`) [Function]

`get-auto-flush-bus` [Method]

Check if *pipeline* will automatically flush messages when going to the NULL state.

pipeline	a `<gst-pipeline>`
ret	whether the pipeline will automatically flush its bus when going from READY to NULL state or not. MT safe.

Since 0.10.4

`gst-pipeline-set-delay` (*self* `<gst-pipeline>`) (*delay* `unsigned-long-long`) [Function]

`set-delay` [Method]

Set the expected delay needed for all elements to perform the PAUSED to PLAYING state change. *delay* will be added to the base time of the elements so that they wait an additional *delay* amount of time before starting to process buffers and cannot be `<gst-clock-time-none>`.

This option is used for tuning purposes and should normally not be used.

MT safe.

pipeline	a `<gst-pipeline>`
delay	the delay

Since 0.10.5

`gst-pipeline-get-delay` (*self* `<gst-pipeline>`) [Function]
⇒ (*ret* `unsigned-long-long`)

`get-delay` [Method]

Get the configured delay (see `gst-pipeline-set-delay`).

pipeline	a `<gst-pipeline>`
ret	The configured delay. MT safe.

Since 0.10.5

28 GstPluginFeature

Base class for contents of a GstPlugin

28.1 Overview

This is a base class for anything that can be added to a `<gst-plugin>`.

28.2 Usage

`<gst-plugin-feature>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-plugin-feature-type-name-filter` [Function]
(*self* `<gst-plugin-feature>`) (*data* `<gst-type-name-data*>`)
⇒ (*ret* `bool`)

`type-name-filter` [Method]

Compares type and name of plugin feature. Can be used with `gst-filter-run`.

feature	the `<gst-plugin-feature>`
data	the type and name to check against
ret	TRUE if equal.

`gst-plugin-feature-set-rank` (*self* `<gst-plugin-feature>`) [Function]
(*rank* `unsigned-int`)

`set-rank` [Method]

Specifies a rank for a plugin feature, so that autoplugging uses the most appropriate feature.

feature	feature to rank
rank	rank value - higher number means more priority rank

`gst-plugin-feature-set-name` (*self* `<gst-plugin-feature>`) [Function]
(*name* `mchars`)

`set-name` [Method]

Sets the name of a plugin feature. The name uniquely identifies a feature within all features of the same type. Renaming a plugin feature is not allowed. A copy is made of the name so you should free the supplied *name* after calling this function.

feature	a feature
name	the name to set

`gst-plugin-feature-get-rank` (*self* `<gst-plugin-feature>`) [Function]
⇒ (*ret* `unsigned-int`)

`get-rank` [Method]

Gets the rank of a plugin feature.

feature	a feature
ret	The rank of the feature

`gst-plugin-feature-get-name` (*self* `<gst-plugin-feature>`) [Function]
  ⇒ (*ret* `mchars`)

`get-name` [Method]

Gets the name of a plugin feature.

feature	a feature
ret	the name

`gst-plugin-feature-load` (*self* `<gst-plugin-feature>`) [Function]
  ⇒ (*ret* `<gst-plugin-feature>`)

`load` [Method]

Loads the plugin containing *feature* if it's not already loaded. *feature* is unaffected; use the return value instead.

Normally this function is used like this:

```
GstPluginFeature *loaded_feature;
loaded_feature = gst_plugin_feature_load (feature);
// presumably, we're no longer interested in the potentially-unloaded feature
gst_object_unref (feature);
feature = loaded_feature;
```

feature	the plugin feature to check
ret	A reference to the loaded feature, or NULL on error.

`gst-plugin-feature-check-version` (*self* `<gst-plugin-feature>`) [Function]
  (*min_major* `unsigned-int`) (*min_minor* `unsigned-int`)
  (*min_micro* `unsigned-int`) ⇒ (*ret* `bool`)

`check-version` [Method]

Checks whether the given plugin feature is at least the required version

feature	a feature
min-major	minimum required major version
min-minor	minimum required minor version
min-micro	minimum required micro version
ret	`#t` if the plugin feature has at least the required version, otherwise `#f`.

29 GstPlugin

Container for features loaded from a shared object module

29.1 Overview

GStreamer is extensible, so `<gst-element>` instances can be loaded at runtime. A plugin system can provide one or more of the basic *GStreamer*`<gst-plugin-feature>` subclasses.

A plugin should export a symbol `gst_plugin_desc` that is a struct of type `<gst-plugin-desc>`. the plugin loader will check the version of the core library the plugin was linked against and will create a new `<gst-plugin>`. It will then call the `<gst-plugin-init-func>` function that was provided in the `gst_plugin_desc`.

Once you have a handle to a `<gst-plugin>` (e.g. from the `<gst-registry-pool>`), you can add any object that subclasses `<gst-plugin-feature>`.

Use `gst-plugin-find-feature` and `gst-plugin-get-feature-list` to find features in a plugin.

Usually plugins are always automaticlly loaded so you don't need to call `gst-plugin-load` explicitly to bring it into memory. There are options to statically link plugins to an app or even use GStreamer without a plugin repository in which case `gst-plugin-load` can be needed to bring the plugin into memory.

29.2 Usage

`<gst-plugin>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-plugin-error-quark` ⇒ (*ret* `unsigned-int`) [Function]

Get the error quark.

ret	The error quark used in GError messages

`gst-plugin-get-name` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-name` [Method]

Get the short name of the plugin

plugin	plugin to get the name of
ret	the name of the plugin

`gst-plugin-get-description` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-description` [Method]

Get the long descriptive name of the plugin

plugin	plugin to get long name of
ret	the long name of the plugin

`gst-plugin-get-filename` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-filename` [Method]

get the filename of the plugin

plugin plugin to get the filename of

ret the filename of the plugin

`gst-plugin-get-license` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-license` [Method]

get the license of the plugin

plugin plugin to get the license of

ret the license of the plugin

`gst-plugin-get-package` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-package` [Method]

get the package the plugin belongs to.

plugin plugin to get the package of

ret the package of the plugin

`gst-plugin-get-origin` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-origin` [Method]

get the URL where the plugin comes from

plugin plugin to get the origin of

ret the origin of the plugin

`gst-plugin-get-source` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-source` [Method]

get the source module the plugin belongs to.

plugin plugin to get the source of

ret the source of the plugin

`gst-plugin-get-version` (*self* `<gst-plugin>`) ⇒ (*ret* `mchars`) [Function]
`get-version` [Method]

get the version of the plugin

plugin plugin to get the version of

ret the version of the plugin

`gst-plugin-get-module` (*self* `<gst-plugin>`) ⇒ (*ret* `<g-module*>`) [Function]
`get-module` [Method]

Gets the `<g-module>` of the plugin. If the plugin isn't loaded yet, NULL is returned.

plugin plugin to query

ret module belonging to the plugin or NULL if the plugin isn't loaded yet.

`gst-plugin-is-loaded` (*self* `<gst-plugin>`) ⇒ (*ret* `bool`) [Function]
`is-loaded` [Method]

queries if the plugin is loaded into memory

plugin plugin to query

ret TRUE is loaded, FALSE otherwise

gst-plugin-name-filter (*self* `<gst-plugin>`) (*name* `mchars`) ⇒ (*ret* `bool`) [Function]

name-filter [Method]

A standard filter that returns TRUE when the plugin is of the given name.

plugin	the plugin to check
name	the name of the plugin
ret	TRUE if the plugin is of the given name.

gst-plugin-load-file (*filename* `mchars`) ⇒ (*ret* `<gst-plugin>`) [Function]

Loads the given plugin and refs it. Caller needs to unref after use.

filename	the plugin filename to load
error	pointer to a NULL-valued GError
ret	a reference to the existing loaded GstPlugin, a reference to the newly-loaded GstPlugin, or NULL if an error occurred.

gst-plugin-load (*self* `<gst-plugin>`) ⇒ (*ret* `<gst-plugin>`) [Function]

load [Method]

Loads *plugin*. Note that the *return value* is the loaded plugin; *plugin* is untouched. The normal use pattern of this function goes like this:

```
GstPlugin *loaded_plugin;
loaded_plugin = gst_plugin_load (plugin);
// presumably, we're no longer interested in the potentially-unloaded plugin
gst_object_unref (plugin);
plugin = loaded_plugin;
```

plugin	plugin to load
ret	A reference to a loaded plugin, or NULL on error.

gst-plugin-load-by-name (*name* `mchars`) ⇒ (*ret* `<gst-plugin>`) [Function]

Load the named plugin. Refs the plugin.

name	name of plugin to load
ret	A reference to a loaded plugin, or NULL on error.

30 GstQuery

Dynamically register new query types. Provide functions to create queries, and to set and parse values in them.

30.1 Overview

GstQuery functions are used to register a new query types to the gstreamer core. Query types can be used to perform queries on pads and elements.

Queries can be created using the `gst-query-new-xxx` functions. Query values can be set using `gst-query-set-xxx`, and parsed using `gst-query-parse-xxx` helpers.

The following example shows how to query the duration of a pipeline:

```
GstQuery *query;
gboolean res;
query = gst_query_new_duration (GST_FORMAT_TIME);
res = gst_element_query (pipeline, query);
if (res) {
  gint64 duration;
  gst_query_parse_duration (query, NULL, &duration);
  g_print ("duration = %"GST_TIME_FORMAT, GST_TIME_ARGS (duration));
}
else {
  g_print ("duration query failed...");
}
gst_query_unref (query);
```

Last reviewed on 2006-02-14 (0.10.4)

30.2 Usage

`<gst-query>` [Class]

`gst-query-type-get-name` (*self* `<gst-query-type*>`) [Function]
⇒ (*ret* `mchars`)

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

query	the query type
ret	a reference to the static name of the query.

`gst-query-type-to-quark` (*self* `<gst-query-type*>`) [Function]
⇒ (*ret* `unsigned-int`)

Get the unique quark for the given query type.

query	the query type
ret	the quark associated with the query type

`gst-query-type-register` (*nick* `mchars`) (*description* `mchars`) [Function]
⇒ (*ret* `<gst-query-type>`)

Create a new GstQueryType based on the nick or return an already registered query with that nick

nick The nick of the new query

description
The description of the new query

ret A new GstQueryType or an already registered query with the same nick.

`gst-query-type-get-by-nick` (*nick* `mchars`) [Function]
⇒ (*ret* `<gst-query-type>`)

Get the query type registered with *nick*.

nick The nick of the query

ret The query registered with *nick* or `<gst-query-none>` if the query was not registered.

`gst-query-types-contains` (*self* `<gst-query-type*>`) [Function]
(*type* `<gst-query-type>`) ⇒ (*ret* `bool`)

See if the given `<gst-query-type>` is inside the *types* query types array.

types The query array to search

type the `<gst-query-type>` to find

ret TRUE if the type is found inside the array

`gst-query-type-iterate-definitions` ⇒ (*ret* `<gst-iterator*>`) [Function]

Get a `<gst-iterator>` of all the registered query types. The definitions iterated over are read only.

ret A `<gst-iterator>` of `<gst-query-type-definition>`.

`gst-query-new-application` (*type* `<gst-query-type>`) [Function]
(*structure* `<gst-structure>`) ⇒ (*ret* `<gst-query>`)

Constructs a new custom application query object. Use `gst-query-unref` when done with it.

type the query type

structure a structure for the query

ret a `<gst-query>`

`gst-query-get-structure` (*self* `<gst-query>`) [Function]
⇒ (*ret* `<gst-structure>`)

`get-structure` [Method]

Get the structure of a query.

query a `<gst-query>`

ret The `<gst-structure>` of the query. The structure is still owned by the query and will therefore be freed when the query is unreffed.

gst-query-new-convert (*src_format* `<gst-format>`) (*value* `int64`) (*dest_format* `<gst-format>`) ⇒ (*ret* `<gst-query>`) [Function]

Constructs a new convert query object. Use `gst-query-unref` when done with it. A convert query is used to ask for a conversion between one format and another.

src-format the source `<gst-format>` for the new query

value the value to convert

dest-format the target `<gst-format>`

ret A `<gst-query>`

gst-query-set-convert (*self* `<gst-query>`) (*src_format* `<gst-format>`) (*src_value* `int64`) (*dest_format* `<gst-format>`) (*dest_value* `int64`) [Function]

set-convert [Method]

Answer a convert query by setting the requested values.

query a `<gst-query>`

src-format the source `<gst-format>`

src-value the source value

dest-format the destination `<gst-format>`

dest-value the destination value

gst-query-parse-convert (*self* `<gst-query>`) (*src_format* `<gst-format*>`) (*dest_format* `<gst-format*>`) ⇒ (*src_value* `int64`) (*dest_value* `int64`) [Function]

parse-convert [Method]

Parse a convert query answer. Any of *src-format*, *src-value*, *dest-format*, and *dest-value* may be NULL, in which case that value is omitted.

query a `<gst-query>`

src-format the storage for the `<gst-format>` of the source value, or NULL

src-value the storage for the source value, or NULL

dest-format the storage for the `<gst-format>` of the destination value, or NULL

dest-value the storage for the destination value, or NULL

gst-query-new-position (*format* `<gst-format>`) ⇒ (*ret* `<gst-query>`) [Function]

Constructs a new query stream position query object. Use `gst-query-unref` when done with it. A position query is used to query the current position of playback in the streams, in some format.

format the default `<gst-format>` for the new query

ret A `<gst-query>`

`gst-query-set-position` (*self* `<gst-query>`) (*format* `<gst-format>`) (*cur* `int64`) [Function]
`set-position` [Method]

Answer a position query by setting the requested value in the given format.

query	a `<gst-query>` with query type GST_QUERY_POSITION
format	the requested `<gst-format>`
cur	the position to set

`gst-query-parse-position` (*self* `<gst-query>`) (*format* `<gst-format*>`) ⇒ (*cur* `int64`) [Function]
`parse-position` [Method]

Parse a position query, writing the format into *format*, and the position into *cur*, if the respective parameters are non-NULL.

query	a `<gst-query>`
format	the storage for the `<gst-format>` of the position values (may be NULL)
cur	the storage for the current position (may be NULL)

`gst-query-new-duration` (*format* `<gst-format>`) ⇒ (*ret* `<gst-query>`) [Function]

Constructs a new stream duration query object to query in the given format. Use `gst-query-unref` when done with it. A duration query will give the total length of the stream.

format	the `<gst-format>` for this duration query
ret	A `<gst-query>`

`gst-query-set-duration` (*self* `<gst-query>`) (*format* `<gst-format>`) (*duration* `int64`) [Function]
`set-duration` [Method]

Answer a duration query by setting the requested value in the given format.

query	a `<gst-query>`
format	the `<gst-format>` for the duration
duration	the duration of the stream

`gst-query-parse-duration` (*self* `<gst-query>`) (*format* `<gst-format*>`) ⇒ (*duration* `int64`) [Function]
`parse-duration` [Method]

Parse a duration query answer. Write the format of the duration into *format*, and the value into *duration*, if the respective variables are non-NULL.

query	a `<gst-query>`
format	the storage for the `<gst-format>` of the duration value, or NULL.
duration	the storage for the total duration, or NULL.

gst-query-new-seeking (*format* `<gst-format>`) [Function]
⇒ (*ret* `<gst-query>`)

Constructs a new query object for querying seeking properties of the stream.

format the default `<gst-format>` for the new query

ret A `<gst-query>`

gst-query-set-seeking (*self* `<gst-query>`) (*format* `<gst-format>`) [Function]
(*seekable* `bool`) (*segment_start* `int64`) (*segment_end* `int64`)

set-seeking [Method]

Set the seeking query result fields in *query*.

query a `<gst-query>`

format the format to set for the *segment-start* and *segment-end* values

seekable the seekable flag to set

segment-start
the segment_start to set

segment-end
the segment_end to set

gst-query-parse-seeking (*self* `<gst-query>`) [Function]
(*format* `<gst-format*>`) ⇒ (*seekable* `bool`) (*segment_start* `int64`)
(*segment_end* `int64`)

parse-seeking [Method]

Parse a seeking query, writing the format into *format*, and other results into the passed parameters, if the respective parameters are non-NULL

query a GST_QUERY_SEEKING type query `<gst-query>`

format the format to set for the *segment-start* and *segment-end* values

seekable the seekable flag to set

segment-start
the segment_start to set

segment-end
the segment_end to set

gst-query-new-formats ⇒ (*ret* `<gst-query>`) [Function]

Constructs a new query object for querying formats of the stream.

ret A `<gst-query>`

Since 0.10.4

gst-query-set-formatsv (*self* `<gst-query>`) (*n_formats* `int`) [Function]
(*formats* `<gst-format*>`)

set-formatsv [Method]

Set the formats query result fields in *query*. The number of formats passed in the *formats* array must be equal to *n-formats*.

query a `<gst-query>`

n-formats the number of formats to set.

formats An array containing *n-formatsgst-format* values.

Since 0.10.4

`gst-query-parse-formats-length` (*self* `<gst-query>`) [Function]
⇒ (*n_formats* `unsigned-int`)
`parse-formats-length` [Method]
Parse the number of formats in the formats *query*.

query a `<gst-query>`

n-formats the number of formats in this query.

Since 0.10.4

`gst-query-parse-formats-nth` (*self* `<gst-query>`) [Function]
(*nth* `unsigned-int`) (*format* `<gst-format*>`)
`parse-formats-nth` [Method]
Parse the format query and retrieve the *nth* format from it into *format*. If the list contains less elements than *nth*, *format* will be set to GST_FORMAT_UNDEFINED.

query a `<gst-query>`

nth the nth format to retrieve.

format a pointer to store the nth format

Since 0.10.4

`gst-query-new-segment` (*format* `<gst-format>`) [Function]
⇒ (*ret* `<gst-query>`)
Constructs a new segment query object. Use `gst-query-unref` when done with it. A segment query is used to discover information about the currently configured segment for playback.

format the `<gst-format>` for the new query

ret a `<gst-query>`

`gst-query-set-segment` (*self* `<gst-query>`) (*rate* `double`) [Function]
(*format* `<gst-format>`) (*start_value* `int64`) (*stop_value* `int64`)
`set-segment` [Method]
Answer a segment query by setting the requested values. The normal playback segment of a pipeline is 0 to duration at the default rate of 1.0. If a seek was performed on the pipeline to play a different segment, this query will return the range specified in the last seek.

start-value and *stop-value* will respectively contain the configured playback range start and stop values expressed in *format*. The values are always between 0 and the duration of the media and *start-value* <= *stop-value*. *rate* will contain the playback rate. For negative rates, playback will actually happen from *stop-value* to *start-value*.

query	a `<gst-query>`
rate	the rate of the segment
format	the `<gst-format>` of the segment values (*start-value* and *stop-value*)
start-value	the start value
stop-value	the stop value

`gst-query-parse-segment` (*self* `<gst-query>`) (*format* `<gst-format*>`) ⇒ (*rate* `double`) (*start_value* `int64`) (*stop_value* `int64`) [Function]

`parse-segment` [Method]

Parse a segment query answer. Any of *rate*, *format*, *start-value*, and *stop-value* may be NULL, which will cause this value to be omitted.

See `gst-query-set-segment` for an explanation of the function arguments.

query	a `<gst-query>`
rate	the storage for the rate of the segment, or NULL
format	the storage for the `<gst-format>` of the values, or NULL
start-value	the storage for the start value, or NULL
stop-value	the storage for the stop value, or NULL

31 GstRegistry

Abstract base class for management of objects

31.1 Overview

One registry holds the metadata of a set of plugins. All registries build the `<gst-registry-pool>`.

Design:

The `<gst-registry>` object is a list of plugins and some functions for dealing with them. `<gst-plugins>` are matched 1-1 with a file on disk, and may or may not be loaded at a given time. There may be multiple `<gst-registry>` objects, but the "default registry" is the only object that has any meaning to the core.

The registry.xml file is actually a cache of plugin information. This is unlike versions prior to 0.10, where the registry file was the primary source of plugin information, and was created by the gst-register command.

The primary source, at all times, of plugin information is each plugin file itself. Thus, if an application wants information about a particular plugin, or wants to search for a feature that satisfies given criteria, the primary means of doing so is to load every plugin and look at the resulting information that is gathered in the default registry. Clearly, this is a time consuming process, so we cache information in the registry.xml file.

On startup, plugins are searched for in the plugin search path. This path can be set directly using the '`GST_PLUGIN_PATH`' environment variable. The registry file is loaded from ~/.gstreamer-$GST_MAJORMINOR/registry-$ARCH.xml or the file listed in the '`GST_REGISTRY`' env var. The only reason to change the registry location is for testing.

For each plugin that is found in the plugin search path, there could be 3 possibilities for cached information:

- the cache may not contain information about a given file.
- the cache may have stale information.
- the cache may have current information.

In the first two cases, the plugin is loaded and the cache updated. In addition to these cases, the cache may have entries for plugins that are not relevant to the current process. These are marked as not available to the current process. If the cache is updated for whatever reason, it is marked dirty.

A dirty cache is written out at the end of initialization. Each entry is checked to make sure the information is minimally valid. If not, the entry is simply dropped.

Implementation notes:

The "cache" and "default registry" are different concepts and can represent different sets of plugins. For various reasons, at init time, the cache is stored in the default registry, and plugins not relevant to the current process are marked with the '`GST_PLUGIN_FLAG_CACHED`' bit. These plugins are removed at the end of intitialization.

31.2 Usage

`<gst-registry>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`plugin-added` (*arg0* `<gpointer>`) [Signal on `<gst-registry>`]

Signals that a plugin has been added to the registry (possibly replacing a previously-added one by the same name)

`feature-added` (*arg0* `<gpointer>`) [Signal on `<gst-registry>`]

Signals that a feature has been added to the registry (possibly replacing a previously-added one by the same name)

`gst-registry-get-default` ⇒ (*ret* `<gst-registry>`) [Function]

Retrieves the default registry. The caller does not own a reference on the registry, as it is alive as long as GStreamer is initialized.

ret The default `<gst-registry>`.

`gst-registry-get-feature-list` (*self* `<gst-registry>`) (*type* `<gtype>`) ⇒ (*ret* `glist-of`) [Function]

`get-feature-list` [Method]

Retrieves a `<g-list>` of `<gst-plugin-feature>` of *type*.

registry a `<gst-registry>`

type a `<g-type>`.

ret a `<g-list>` of `<gst-plugin-feature>` of *type*. gst_plugin_feature_list_free after usage. MT safe.

`gst-registry-get-path-list` (*self* `<gst-registry>`) ⇒ (*ret* `glist-of`) [Function]

`get-path-list` [Method]

Get the list of paths for the given registry.

registry the registry to get the pathlist of

ret A Glist of paths as strings. g_list_free after use. MT safe.

`gst-registry-get-plugin-list` (*self* `<gst-registry>`) ⇒ (*ret* `glist-of`) [Function]

`get-plugin-list` [Method]

Get a copy of all plugins registered in the given registry. The refcount of each element in the list in incremented.

registry the registry to search

ret a `<g-list>` of `<gst-plugin>`. gst_plugin_list_free after use. MT safe.

`gst-registry-add-plugin` (*self* `<gst-registry>`) [Function]
(*plugin* `<gst-plugin>`) ⇒ (*ret* `bool`)

`add-plugin` [Method]

Add the plugin to the registry. The plugin-added signal will be emitted. This function will sink *plugin*.

registry	the registry to add the plugin to
plugin	the plugin to add
ret	TRUE on success. MT safe.

`gst-registry-remove-plugin` (*self* `<gst-registry>`) [Function]
(*plugin* `<gst-plugin>`)

`remove-plugin` [Method]

Remove the plugin from the registry.

MT safe.

registry	the registry to remove the plugin from
plugin	the plugin to remove

`gst-registry-plugin-filter` (*self* `<gst-registry>`) [Function]
(*filter* `<gst-plugin-filter>`) (*first* `bool`) (*user_data* `<gpointer>`)
⇒ (*ret* `glist-of`)

`plugin-filter` [Method]

Runs a filter against all plugins in the registry and returns a `<g-list>` with the results. If the first flag is set, only the first match is returned (as a list with a single object). Every plugin is reffed; use `gst-plugin-list-free` after use, which will unref again.

registry	registry to query
filter	the filter to use
first	only return first match
user-data	user data passed to the filter function
ret	a `<g-list>` of `<gst-plugin>`. Use `gst-plugin-list-free` after usage. MT safe.

`gst-registry-feature-filter` (*self* `<gst-registry>`) [Function]
(*filter* `<gst-plugin-feature-filter>`) (*first* `bool`)
(*user_data* `<gpointer>`) ⇒ (*ret* `glist-of`)

`feature-filter` [Method]

Runs a filter against all features of the plugins in the registry and returns a GList with the results. If the first flag is set, only the first match is returned (as a list with a single object).

registry	registry to query
filter	the filter to use
first	only return first match

user-data user data passed to the filter function

ret a GList of plugin features, gst_plugin_feature_list_free after use. MT safe.

`gst-registry-find-plugin` (*self* `<gst-registry>`) (*name* `mchars`) ⇒ (*ret* `<gst-plugin>`) [Function]

`find-plugin` [Method]

Find the plugin with the given name in the registry. The plugin will be reffed; caller is responsible for unreffing.

registry the registry to search

name the plugin name to find

ret The plugin with the given name or NULL if the plugin was not found. `gst-object-unref` after usage. MT safe.

`gst-registry-find-feature` (*self* `<gst-registry>`) (*name* `mchars`) (*type* `<gtype>`) ⇒ (*ret* `<gst-plugin-feature>`) [Function]

`find-feature` [Method]

Find the pluginfeature with the given name and type in the registry.

registry the registry to search

name the pluginfeature name to find

type the pluginfeature type to find

ret The pluginfeature with the given name and type or NULL if the plugin was not found. `gst-object-unref` after usage. MT safe.

`gst-registry-lookup-feature` (*self* `<gst-registry>`) (*name* `mchars`) ⇒ (*ret* `<gst-plugin-feature>`) [Function]

`lookup-feature` [Method]

Find a `<gst-plugin-feature>` with *name* in *registry*.

registry a `<gst-registry>`

name a `<gst-plugin-feature>` name

ret a `<gst-plugin-feature>` with its refcount incremented, use `gst-object-unref` after usage. MT safe.

`gst-registry-scan-path` (*self* `<gst-registry>`) (*path* `mchars`) ⇒ (*ret* `bool`) [Function]

`scan-path` [Method]

Add the given path to the registry. The syntax of the path is specific to the registry. If the path has already been added, do nothing.

registry the registry to add the path to

path the path to add to the registry

ret ‘`#t`’ if registry changed

`gst-registry-xml-read-cache` (*self* `<gst-registry>`) (*location* `mchars`) ⇒ (*ret* `bool`) [Function]

`xml-read-cache` [Method]

Read the contents of the XML cache file at *location* into *registry*.

registry	a `<gst-registry>`
location	a filename
ret	'`#t`' on success.

`gst-registry-xml-write-cache` (*self* `<gst-registry>`) (*location* `mchars`) ⇒ (*ret* `bool`) [Function]

`xml-write-cache` [Method]

Write *registry* in an XML format at the location given by *location*. Directories are automatically created.

registry	a `<gst-registry>`
location	a filename
ret	TRUE on success.

`gst-registry-lookup` (*self* `<gst-registry>`) (*filename* `mchars`) ⇒ (*ret* `<gst-plugin>`) [Function]

`lookup` [Method]

Look up a plugin in the given registry with the given filename. If found, plugin is reffed.

registry	the registry to look up in
filename	the name of the file to look up
ret	the `<gst-plugin>` if found, or NULL if not. `gst-object-unref` after usage.

`gst-registry-remove-feature` (*self* `<gst-registry>`) (*feature* `<gst-plugin-feature>`) [Function]

`remove-feature` [Method]

Remove the feature from the registry.

MT safe.

registry	the registry to remove the feature from
feature	the feature to remove

`gst-registry-add-feature` (*self* `<gst-registry>`) (*feature* `<gst-plugin-feature>`) ⇒ (*ret* `bool`) [Function]

`add-feature` [Method]

Add the feature to the registry. The feature-added signal will be emitted. This function sinks *feature*.

registry	the registry to add the plugin to
feature	the feature to add
ret	TRUE on success. MT safe.

32 GstSegment

Structure describing the configured region of interest in a media file.

32.1 Overview

This helper structure holds the relevant values for tracking the region of interest in a media file, called a segment.

The structure can be used for two purposes:

- performing seeks (handling seek events)
- tracking playback regions (handling newsegment events)

The segment is usually configured by the application with a seek event which is propagated upstream and eventually handled by an element that performs the seek.

The configured segment is then propagated back downstream with a newsegment event. This information is then used to clip media to the segment boundaries.

A segment structure is initialized with `gst-segment-init`, which takes a `<gst-format>` that will be used as the format of the segment values. The segment will be configured with a start value of 0 and a stop/duration of -1, which is undefined. The default rate and applied_rate is 1.0.

If the segment is used for managing seeks, the segment duration should be set with `gst-segment-set-duration`. The public duration field contains the duration of the segment. When using the segment for seeking, the start and time members should normally be left to their default 0 value. The stop position is left to -1 unless explicitly configured to a different value after a seek event.

The current position in the segment should be set with the `gst-segment-set-last-stop`. The public last_stop field contains the last set stop position in the segment.

For elements that perform seeks, the current segment should be updated with the `gst-segment-set-seek` and the values from the seek event. This method will update all the segment fields. The last_stop field will contain the new playback position. If the cur_type was different from GST_SEEK_TYPE_NONE, playback continues from the last_stop position, possibly with updated flags or rate.

For elements that want to use `<gst-segment>` to track the playback region, use `gst-segment-set-newsegment` to update the segment fields with the information from the newsegment event. The `gst-segment-clip` method can be used to check and clip the media data to the segment boundaries.

For elements that want to synchronize to the pipeline clock, `gst-segment-to-running-time` can be used to convert a timestamp to a value that can be used to synchronize to the clock. This function takes into account all accumulated segments as well as any rate or applied_rate conversions.

For elements that need to perform operations on media data in stream_time, `gst-segment-to-stream-time` can be used to convert a timestamp and the segment info to stream time (which is always between 0 and the duration of the stream).

Last reviewed on 2007-05-17 (0.10.13)

32.2 Usage

`gst-segment-clip` (*self* `<gst-segment*>`) (*format* `<gst-format>`) (*start* `int64`) (*stop* `int64`) ⇒ (*ret* `bool`) (*clip_start* `int64`) (*clip_stop* `int64`) [Function]

Clip the given *start* and *stop* values to the segment boundaries given in *segment*. *start* and *stop* are compared and clipped to *segment* start and stop values.

If the function returns FALSE, *start* and *stop* are known to fall outside of *segment* and *clip-start* and *clip-stop* are not updated.

When the function returns TRUE, *clip-start* and *clip-stop* will be updated. If *clip-start* or *clip-stop* are different from *start* or *stop* respectively, the region fell partially in the segment.

Note that when *stop* is -1, *clip-stop* will be set to the end of the segment. Depending on the use case, this may or may not be what you want.

segment	a `<gst-segment>` structure.
format	the format of the segment.
start	the start position in the segment
stop	the stop position in the segment
clip-start	the clipped start position in the segment
clip-stop	the clipped stop position in the segment
ret	TRUE if the given *start* and *stop* times fall partially or completely in *segment*, FALSE if the values are completely outside of the segment.

`gst-segment-init` (*self* `<gst-segment*>`) (*format* `<gst-format>`) [Function]

The start/last_stop positions are set to 0 and the stop/duration fields are set to -1 (unknown). The default rate of 1.0 and no flags are set.

Initialize *segment* to its default values.

segment	a `<gst-segment>` structure.
format	the format of the segment.

`gst-segment-new` ⇒ (*ret* `<gst-segment*>`) [Function]

Allocate a new `<gst-segment>` structure and initialize it using `gst-segment-init`.

ret	a new `<gst-segment>`, free with `gst-segment-free`.

`gst-segment-set-duration` (*self* `<gst-segment*>`) (*format* `<gst-format>`) (*duration* `int64`) [Function]

Set the duration of the segment to *duration*. This function is mainly used by elements that perform seeking and know the total duration of the segment.

This field should be set to allow seeking requests relative to the duration.

segment	a `<gst-segment>` structure.
format	the format of the segment.
duration	the duration of the segment info or -1 if unknown.

gst-segment-set-last-stop (*self* `<gst-segment*>`) (*format* `<gst-format>`) (*position* `int64`) [Function]

Set the last observed stop position in the segment to *position*.

This field should be set to allow seeking requests relative to the current playing position.

segment	a `<gst-segment>` structure.
format	the format of the segment.
position	the position

gst-segment-set-newsegment (*self* `<gst-segment*>`) (*update* `bool`) (*rate* `double`) (*format* `<gst-format>`) (*start* `int64`) (*stop* `int64`) (*time* `int64`) [Function]

Update the segment structure with the field values of a new segment event and with a default applied_rate of 1.0.

segment	a `<gst-segment>` structure.
update	flag indicating a new segment is started or updated
rate	the rate of the segment.
format	the format of the segment.
start	the new start value
stop	the new stop value
time	the new stream time

Since 0.10.6

gst-segment-set-newsegment-full (*self* `<gst-segment*>`) (*update* `bool`) (*rate* `double`) (*applied_rate* `double`) (*format* `<gst-format>`) (*start* `int64`) (*stop* `int64`) (*time* `int64`) [Function]

Update the segment structure with the field values of a new segment event.

segment	a `<gst-segment>` structure.
update	flag indicating a new segment is started or updated
rate	the rate of the segment.
applied-rate	the applied rate of the segment.
format	the format of the segment.
start	the new start value
stop	the new stop value
time	the new stream time

`gst-segment-set-seek` (*self* `<gst-segment*>`) (*rate* `double`) (*format* `<gst-format>`) (*flags* `<gst-seek-flags>`) (*cur_type* `<gst-seek-type>`) (*cur* `int64`) (*stop_type* `<gst-seek-type>`) (*stop* `int64`) ⇒ (*update* `bool`) [Function]

Update the segment structure with the field values of a seek event (see `gst-event-new-seek`).

After calling this method, the segment field last_stop and time will contain the requested new position in the segment. The new requested position in the segment depends on *rate* and *start-type* and *stop-type*.

For positive *rate*, the new position in the segment is the new *segment* start field when it was updated with a *start-type* different from `<gst-seek-type-none>`. If no update was performed on *segment* start position (`<gst-seek-type-none>`), *start* is ignored and *segment* last_stop is unmodified.

For negative *rate*, the new position in the segment is the new *segment* stop field when it was updated with a *stop-type* different from `<gst-seek-type-none>`. If no stop was previously configured in the segment, the duration of the segment will be used to update the stop position. If no update was performed on *segment* stop position (`<gst-seek-type-none>`), *stop* is ignored and *segment* last_stop is unmodified.

The applied rate of the segment will be set to 1.0 by default. If the caller can apply a rate change, it should update *segment* rate and applied_rate after calling this function.

update will be set to TRUE if a seek should be performed to the segment last_stop field. This field can be FALSE if, for example, only the *rate* has been changed but not the playback position.

segment	a `<gst-segment>` structure.
rate	the rate of the segment.
format	the format of the segment.
flags	the seek flags for the segment
start-type	the seek method
start	the seek start value
stop-type	the seek method
stop	the seek stop value
update	boolean holding whether last_stop was updated.

`gst-segment-to-running-time` (*self* `<gst-segment*>`) (*format* `<gst-format>`) (*position* `int64`) ⇒ (*ret* `int64`) [Function]

Translate *position* to the total running time using the currently configured and previously accumulated segments. Position is a value between *segment* start and stop time.

This function is typically used by elements that need to synchronize to the global clock in a pipeline. The runnning time is a constantly increasing value starting from 0. When `gst-segment-init` is called, this value will reset to 0.

This function returns -1 if the position is outside of *segment* start and stop.

segment	a `<gst-segment>` structure.
format	the format of the segment.
position	the position in the segment
ret	the position as the total running time or -1 when an invalid position was given.

`gst-segment-to-stream-time` (*self* `<gst-segment*>`) (*format* `<gst-format>`) (*position* `int64`) ⇒ (*ret* `int64`) [Function]

Translate *position* to stream time using the currently configured segment. The *position* value must be between *segment* start and stop value.

This function is typically used by elements that need to operate on the stream time of the buffers it receives, such as effect plugins. In those use cases, *position* is typically the buffer timestamp or clock time that one wants to convert to the stream time. The stream time is always between 0 and the total duration of the media stream.

segment	a `<gst-segment>` structure.
format	the format of the segment.
position	the position in the segment
ret	the position in stream_time or -1 when an invalid position was given.

33 GstStructure

Generic structure containing fields of names and values

33.1 Overview

A `<gst-structure>` is a collection of key/value pairs. The keys are expressed as GQuarks and the values can be of any GType.

In addition to the key/value pairs, a `<gst-structure>` also has a name. The name starts with a letter and can be folled by letters, numbers and any of `"/-_.:"`.

`<gst-structure>` is used by various GStreamer subsystems to store information in a flexible and extensible way. A `<gst-structure>` does not have a refcount because it usually is part of a higher level object such as `<gst-caps>`. It provides a means to enforce mutability using the refcount of the parent with the `gst-structure-set-parent-refcount` method.

A `<gst-structure>` can be created with `gst-structure-empty-new` or `gst-structure-new`, which both take a name and an optional set of key/value pairs along with the types of the values.

Field values can be changed with `gst-structure-set-value` or `gst-structure-set`.

Field values can be retrieved with `gst-structure-get-value` or the more convenient gst_structure_get_*() functions.

Fields can be removed with `gst-structure-remove-field` or `gst-structure-remove-fields`.

Last reviewed on 2007-10-16 (0.10.15)

33.2 Usage

`<gst-structure>` [Class]

`gst-structure-empty-new` (*name* `mchars`) ⇒ (*ret* `<gst-structure>`) [Function]

Creates a new, empty `<gst-structure>` with the given *name*.

See `gst-structure-set-name` for constraints on the *name* parameter.

name	name of new structure
ret	a new, empty `<gst-structure>`

`gst-structure-id-empty-new` (*quark* `unsigned-int`) ⇒ (*ret* `<gst-structure>`) [Function]

Creates a new, empty `<gst-structure>` with the given name as a GQuark.

quark	name of new structure
ret	a new, empty `<gst-structure>`

`gst-structure-get-name` (*self* `<gst-structure>`) ⇒ (*ret* `mchars`) [Function]

Get the name of *structure* as a string.

structure	a `<gst-structure>`
ret	the name of the structure.

`gst-structure-has-name` (*self* `<gst-structure>`) (*name* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks if the structure has the given name

structure	a `<gst-structure>`
name	structure name to check for
ret	TRUE if *name* matches the name of the structure.

`gst-structure-set-name` (*self* `<gst-structure>`) (*name* `mchars`) [Function]

Sets the name of the structure to the given *name*. The string provided is copied before being used. It must not be empty, start with a letter and can be followed by letters, numbers and any of "/-_.:".

structure	a `<gst-structure>`
name	the new name of the structure

`gst-structure-get-name-id` (*self* `<gst-structure>`) ⇒ (*ret* `unsigned-int`) [Function]

Get the name of *structure* as a GQuark.

structure	a `<gst-structure>`
ret	the quark representing the name of the structure.

`gst-structure-id-get-value` (*self* `<gst-structure>`) (*field* `unsigned-int`) ⇒ (*ret* `<gvalue>`) [Function]

Get the value of the field with GQuark *field*.

structure	a `<gst-structure>`
field	the `<g-quark>` of the field to get
ret	the `<gvalue>` corresponding to the field with the given name identifier.

`gst-structure-id-set-value` (*self* `<gst-structure>`) (*field* `unsigned-int`) (*value* `<gvalue>`) [Function]

Sets the field with the given GQuark *field* to *value*. If the field does not exist, it is created. If the field exists, the previous value is replaced and freed.

structure	a `<gst-structure>`
field	a `<g-quark>` representing a field
value	the new value of the field

`gst-structure-get-value` (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `<gvalue>`) [Function]

Get the value of the field with name *fieldname*.

structure	a `<gst-structure>`
fieldname	the name of the field to get
ret	the `<gvalue>` corresponding to the field with the given name.

`gst-structure-set-value` (*self* `<gst-structure>`) (*fieldname* `mchars`) (*value* `<gvalue>`) [Function]

Sets the field with the given name *field* to *value*. If the field does not exist, it is created. If the field exists, the previous value is replaced and freed.

structure a `<gst-structure>`

fieldname the name of the field to set

value the new value of the field

`gst-structure-remove-field` (*self* `<gst-structure>`) (*fieldname* `mchars`) [Function]

Removes the field with the given name. If the field with the given name does not exist, the structure is unchanged.

structure a `<gst-structure>`

fieldname the name of the field to remove

`gst-structure-remove-all-fields` (*self* `<gst-structure>`) [Function]

Removes all fields in a GstStructure.

structure a `<gst-structure>`

`gst-structure-get-field-type` (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `<gtype>`) [Function]

Finds the field with the given name, and returns the type of the value it contains. If the field is not found, G_TYPE_INVALID is returned.

structure a `<gst-structure>`

fieldname the name of the field

ret the `<gvalue>` of the field

`gst-structure-foreach` (*self* `<gst-structure>`) (*proc* `scm`) [Function]

Calls the provided function once for each field in the `<gst-structure>`. The function must not modify the fields. Also see `gst-structure-map-in-place`.

structure a `<gst-structure>`

func a function to call for each field

user-data private data

ret TRUE if the supplied function returns TRUE For each of the fields, FALSE otherwise.

`gst-structure-n-fields` (*self* `<gst-structure>`) ⇒ (*ret* `int`) [Function]

Get the number of fields in the structure.

structure a `<gst-structure>`

ret the number of fields in the structure

gst-structure-has-field (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) [Function]

Check if *structure* contains a field named *fieldname*.

structure	a `<gst-structure>`
fieldname	the name of a field
ret	TRUE if the structure contains a field with the given name

gst-structure-has-field-typed (*self* `<gst-structure>`) (*fieldname* `mchars`) (*type* `<gtype>`) ⇒ (*ret* `bool`) [Function]

Check if *structure* contains a field named *fieldname* and with GType *type*.

structure	a `<gst-structure>`
fieldname	the name of a field
type	the type of a value
ret	TRUE if the structure contains a field with the given name and type

gst-structure-get-boolean (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) (*value* `bool`) [Function]

Sets the boolean pointed to by *value* corresponding to the value of the given field. Caller is responsible for making sure the field exists and has the correct type.

structure	a `<gst-structure>`
fieldname	the name of a field
value	a pointer to a `<gboolean>` to set
ret	TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain a boolean, this function returns FALSE.

gst-structure-get-int (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) (*value* `int`) [Function]

Sets the int pointed to by *value* corresponding to the value of the given field. Caller is responsible for making sure the field exists and has the correct type.

Returns: '`#t`' if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain an int, this function

structure	a `<gst-structure>`
fieldname	the name of a field
value	a pointer to an int to set
ret	'`#f`'.

gst-structure-get-fourcc (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) (*value* `unsigned-int32`) [Function]

Sets the `<gst-fourcc>` pointed to by *value* corresponding to the value of the given field. Caller is responsible for making sure the field exists and has the correct type.

Returns: TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain a fourcc, this function *structure* a `<gst-structure>`

fieldname the name of a field

value a pointer to a `<gst-fourcc>` to set

ret FALSE.

`gst-structure-get-double` (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) (*value* `double`) [Function]

Sets the double pointed to by *value* corresponding to the value of the given field. Caller is responsible for making sure the field exists and has the correct type.

structure a `<gst-structure>`

fieldname the name of a field

value a pointer to a `<gst-fourcc>` to set

ret TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain a double, this function returns FALSE.

`gst-structure-get-string` (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `mchars`) [Function]

Finds the field corresponding to *fieldname*, and returns the string contained in the field's value. Caller is responsible for making sure the field exists and has the correct type.

The string should not be modified, and remains valid until the next call to a gst_structure_*() function with the given structure.

structure a `<gst-structure>`

fieldname the name of a field

ret a pointer to the string or NULL when the field did not exist or did not contain a string.

`gst-structure-get-date` (*self* `<gst-structure>`) (*fieldname* `mchars`) (*value* `<g-date**>`) ⇒ (*ret* `bool`) [Function]

Sets the date pointed to by *value* corresponding to the date of the given field. Caller is responsible for making sure the field exists and has the correct type.

Returns: TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain a data, this function

structure a `<gst-structure>`

fieldname the name of a field

value a pointer to a `<g-date>` to set

ret FALSE.

`gst-structure-get-clock-time` (*self* `<gst-structure>`) (*fieldname* `mchars`) (*value* `<gst-clock-time*>`) ⇒ (*ret* `bool`) [Function]

Sets the clock time pointed to by *value* corresponding to the clock time of the given field. Caller is responsible for making sure the field exists and has the correct type.

structure a `<gst-structure>`

fieldname the name of a field

value a pointer to a `<gst-clock-time>` to set

ret TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain a `<gst-clock-time>`, this function returns FALSE.

`gst-structure-get-enum` (*self* `<gst-structure>`) (*fieldname* `mchars`) (*enumtype* `<gtype>`) ⇒ (*ret* `bool`) (*value* `int`) [Function]

Sets the int pointed to by *value* corresponding to the value of the given field. Caller is responsible for making sure the field exists, has the correct type and that the enumtype is correct.

structure a `<gst-structure>`

fieldname the name of a field

enumtype the enum type of a field

value a pointer to an int to set

ret TRUE if the value could be set correctly. If there was no field with *fieldname* or the existing field did not contain an enum of the given type, this function returns FALSE.

`gst-structure-get-fraction` (*self* `<gst-structure>`) (*fieldname* `mchars`) ⇒ (*ret* `bool`) (*value_numerator* `int`) (*value_denominator* `int`) [Function]

Sets the integers pointed to by *value-numerator* and *value-denominator* corresponding to the value of the given field. Caller is responsible for making sure the field exists and has the correct type.

structure a `<gst-structure>`

fieldname the name of a field

value-numerator
a pointer to an int to set

value-denominator
a pointer to an int to set

ret TRUE if the values could be set correctly. If there was no field with *fieldname* or the existing field did not contain a GstFraction, this function returns FALSE.

`gst-structure-map-in-place` (*self* `<gst-structure>`) (*func* `<gst-structure-map-func>`) (*user_data* `<gpointer>`) ⇒ (*ret* `bool`) [Function]

Calls the provided function once for each field in the `<gst-structure>`. In contrast to `gst-structure-foreach`, the function may modify but not delete the fields. The structure must be mutable.

structure a `<gst-structure>`

func	a function to call for each field
user-data	private data
ret	TRUE if the supplied function returns TRUE For each of the fields, FALSE otherwise.

`gst-structure-nth-field-name` (*self* `<gst-structure>`) (*index* `unsigned-int`) ⇒ (*ret* `mchars`) [Function]

Get the name of the given field number, counting from 0 onwards.

structure	a `<gst-structure>`
index	the index to get the name of
ret	the name of the given field number

`gst-structure-set-parent-refcount` (*self* `<gst-structure>`) ⇒ (*refcount* `int`) [Function]

Sets the parent_refcount field of `<gst-structure>`. This field is used to determine whether a structure is mutable or not. This function should only be called by code implementing parent objects of `<gst-structure>`, as described in the MT Refcounting section of the design documents.

structure	a `<gst-structure>`
refcount	a pointer to the parent's refcount

`gst-structure-to-string` (*self* `<gst-structure>`) ⇒ (*ret* `mchars`) [Function]

Converts *structure* to a human-readable string representation.

structure	a `<gst-structure>`
ret	a pointer to string allocated by `g-malloc`. `g-free` after usage.

`gst-structure-from-string` (*string* `mchars`) ⇒ (*ret* `<gst-structure>`) [Function]

Creates a `<gst-structure>` from a string representation. If end is not NULL, a pointer to the place inside the given string where parsing ended will be returned.

string	a string representation of a `<gst-structure>`.
end	pointer to store the end of the string in.
ret	a new `<gst-structure>` or NULL when the string could not be parsed. Free after usage.

`gst-structure-fixate-field-boolean` (*self* `<gst-structure>`) (*field_name* `mchars`) (*target* `bool`) ⇒ (*ret* `bool`) [Function]

Fixates a `<gst-structure>` by changing the given *field-name* field to the given *target* boolean if that field is not fixed yet.

structure	a `<gst-structure>`
field-name	a field in *structure*
target	the target value of the fixation
ret	TRUE if the structure could be fixated

34 GstSystemClock

Default clock that uses the current system time

34.1 Overview

The GStreamer core provides a GstSystemClock based on the system time. Asynchronous callbacks are scheduled from an internal thread.

Clock implementors are encouraged to subclass this systemclock as it implements the async notification.

Subclasses can however override all of the important methods for sync and async notifications to implement their own callback methods or blocking wait operations.

Last reviewed on 2006-03-08 (0.10.4)

34.2 Usage

`<gst-system-clock>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-system-clock-obtain` ⇒ (*ret* `<gst-clock>`) [Function]

Get a handle to the default system clock. The refcount of the clock will be increased so you need to unref the clock after usage.

ret the default clock. MT safe.

35 GstTagList

List of tags and values used to describe media metadata

35.1 Overview

List of tags and values used to describe media metadata.

Last reviewed on 2005-11-23 (0.9.5)

35.2 Usage

`gst-tag-register` (*name* `mchars`) (*flag* `<gst-tag-flag>`) (*type* `<gtype>`) (*nick* `mchars`) (*blurb* `mchars`) (*func* `<gst-tag-merge-func>`) [Function]

Registers a new tag type for the use with GStreamer's type system. If a type with that name is already registered, that one is used. The old registration may have used a different type however. So don't rely on your supplied values.

Important: if you do not supply a merge function the implication will be that there can only be one single value for this tag in a tag list and any additional values will silenty be discarded when being added (unless `<gst-tag-merge-replace>`, `<gst-tag-merge-replace-all>`, or `<gst-tag-merge-prepend>` is used as merge mode, in which case the new value will replace the old one in the list).

The merge function will be called from `gst-tag-list-copy-value` when it is required that one or more values for a tag be condensed into one single value. This may happen from `gst-tag-list-get-string`, `gst-tag-list-get-int`, `gst-tag-list-get-double` etc. What will happen exactly in that case depends on how the tag was registered and if a merge function was supplied and if so which one.

Two default merge functions are provided: `gst-tag-merge-use-first` and `gst-tag-merge-strings-with-commas`.

name	the name or identifier string
flag	a flag describing the type of tag info
type	the type this data is in
nick	human-readable name
blurb	a human-readable description about this tag
func	function for merging multiple values of this tag, or NULL

`gst-tag-merge-use-first` (*dest* `<gvalue>`) (*src* `<gvalue>`) [Function]

This is a convenience function for the func argument of `gst-tag-register`. It creates a copy of the first value from the list.

dest	uninitialized GValue to store result in
src	GValue to copy from

`gst-tag-merge-strings-with-comma` (*dest* `<gvalue>`) (*src* `<gvalue>`) [Function]

This is a convenience function for the func argument of `gst-tag-register`. It concatenates all given strings using a comma. The tag must be registered as a G_TYPE_STRING or this function will fail.

dest	uninitialized GValue to store result in
src	GValue to copy from

`gst-tag-exists` (*tag* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks if the given type is already registered.

tag	name of the tag
ret	TRUE if the type is already registered

`gst-tag-get-nick` (*tag* `mchars`) ⇒ (*ret* `mchars`) [Function]

Returns the human-readable name of this tag, You must not change or free this string.

tag	the tag
ret	the human-readable name of this tag

`gst-tag-get-description` (*tag* `mchars`) ⇒ (*ret* `mchars`) [Function]

Returns the human-readable description of this tag, You must not change or free this string.

tag	the tag
ret	the human-readable description of this tag

`gst-tag-get-flag` (*tag* `mchars`) ⇒ (*ret* `<gst-tag-flag>`) [Function]

Gets the flag of *tag*.

tag	the tag
ret	the flag of this tag.

`gst-tag-is-fixed` (*tag* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks if the given tag is fixed. A fixed tag can only contain one value. Unfixed tags can contain lists of values.

tag	tag to check
ret	TRUE, if the given tag is fixed.

`gst-tag-list-new` ⇒ (*ret* `<gst-tag-list*>`) [Function]

Creates a new empty GstTagList.

ret	An empty tag list

`gst-is-tag-list` (*p* `<gconstpointer>`) ⇒ (*ret* `bool`) [Function]

Checks if the given pointer is a taglist.

p	Object that might be a taglist
ret	TRUE, if the given pointer is a taglist

gst-tag-list-insert (*self* `<gst-tag-list*>`) (*from* `<gst-tag-list*>`) (*mode* `<gst-tag-merge-mode>`) [Function]

Inserts the tags of the second list into the first list using the given mode.

into	list to merge into
from	list to merge from
mode	the mode to use

gst-tag-list-merge (*self* `<gst-tag-list*>`) (*list2* `<gst-tag-list*>`) (*mode* `<gst-tag-merge-mode>`) ⇒ (*ret* `<gst-tag-list*>`) [Function]

Merges the two given lists into a new list. If one of the lists is NULL, a copy of the other is returned. If both lists are NULL, NULL is returned.

list1	first list to merge
list2	second list to merge
mode	the mode to use
ret	the new list

gst-tag-list-get-tag-size (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `unsigned-int`) [Function]

Checks how many value are stored in this tag list for the given tag.

list	a taglist
tag	the tag to query
ret	The number of tags stored

gst-tag-list-remove-tag (*self* `<gst-tag-list*>`) (*tag* `mchars`) [Function]

Removes the given tag from the taglist.

list	list to remove tag from
tag	tag to remove

gst-tag-list-foreach (*self* `<gst-tag-list*>`) (*func* `<gst-tag-foreach-func>`) (*user_data* `<gpointer>`) [Function]

Calls the given function for each tag inside the tag list. Note that if there is no tag, the function won't be called at all.

list	list to iterate over
func	function to be called for each tag
user-data	user specified data

gst-tag-list-get-value-index (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `<gvalue>`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>`

tag tag to read out

index number of entry to read out

ret The GValue for the specified entry or NULL if the tag wasn't available or the tag doesn't have as many entries

`gst-tag-list-copy-value` (*dest* `<gvalue>`) (*list* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag. You must `g-value-unset` the value after use.

dest uninitialized `<gvalue>` to copy into

list list to get the tag from

tag tag to read out

ret TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-char` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*value* `mchars`) ⇒ (*ret* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list a `<gst-tag-list>` to get the tag from

tag tag to read out

value location for the result

ret TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-char-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) (*value* `mchars`) ⇒ (*ret* `bool`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list a `<gst-tag-list>` to get the tag from

tag tag to read out

index number of entry to read out

value location for the result

ret TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uchar` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*value* `<guchar*>`) ⇒ (*ret* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list a `<gst-tag-list>` to get the tag from

tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uchar-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) (*value* `<guchar*>`) ⇒ (*ret* `bool`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-boolean` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-boolean-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `bool`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-int` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `int`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from

tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-int-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `int`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uint` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `unsigned-int`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uint-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `unsigned-int`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-long` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `long`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from

tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-long-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `long`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-ulong` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `unsigned-long`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-ulong-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `unsigned-long`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-int64` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `int64`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from

tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-int64-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `int64`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uint64` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `unsigned-int64`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-uint64-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `unsigned-int64`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-float` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `float`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from

tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-float-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `float`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-double` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `double`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-double-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `double`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-string` (*self* `<gst-tag-list*>`) (*tag* `mchars`) ⇒ (*ret* `bool`) (*value* `mchars`) [Function]

Copies the contents for the given tag into the value, possibly merging multiple values into one if multiple values are associated with the tag.

Use gst_tag_list_get_string_index (list, tag, 0, value) if you want to retrieve the first string associated with this tag unmodified.

The resulting string in *value* should be freed by the caller using g_free when no longer needed

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-string-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) ⇒ (*ret* `bool`) (*value* `mchars`) [Function]

Gets the value that is at the given index for the given tag in the given list.

The resulting string in *value* should be freed by the caller using g_free when no longer needed

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-pointer` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*value* `<gpointer*>`) ⇒ (*ret* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

`gst-tag-list-get-pointer-index` (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) (*value* `<gpointer*>`) ⇒ (*ret* `bool`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list.

gst-tag-list-get-date (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*value* `<g-date**>`) ⇒ (*ret* `bool`) [Function]

Copies the contents for the given tag into the value, merging multiple values into one if multiple values are associated with the tag.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list or if it was `#f`.

gst-tag-list-get-date-index (*self* `<gst-tag-list*>`) (*tag* `mchars`) (*index* `unsigned-int`) (*value* `<g-date**>`) ⇒ (*ret* `bool`) [Function]

Gets the value that is at the given index for the given tag in the given list.

list	a `<gst-tag-list>` to get the tag from
tag	tag to read out
index	number of entry to read out
value	location for the result
ret	TRUE, if a value was copied, FALSE if the tag didn't exist in the given list or if it was `#f`.

36 GstTagSetter

Element interface that allows setting and retrieval of media metadata

36.1 Overview

36.2

Element interface that allows setting of media metadata.

Elements that support changing a stream's metadata will implement this interface. Examples of such elements are 'vorbisenc', 'theoraenc' and 'id3v2mux'.

If you just want to retrieve metadata in your application then all you need to do is watch for tag messages on your pipeline's bus. This interface is only for setting metadata, not for extracting it. To set tags from the application, find tagsetter elements and set tags using e.g. `gst-tag-setter-merge-tags` or `gst-tag-setter-add-tags`. The application should do that before the element goes to '`GST_STATE_PAUSED`'.

Elements implementing the `<gst-tag-setter>` interface often have to merge any tags received from upstream and the tags set by the application via the interface. This can be done like this:

```
GstTagMergeMode merge_mode;
const GstTagList *application_tags;
const GstTagList *event_tags;
GstTagSetter *tagsetter;
GstTagList *result;

tagsetter = GST_TAG_SETTER (element);

merge_mode = gst_tag_setter_get_tag_merge_mode (tagsetter);
tagsetter_tags = gst_tag_setter_get_tag_list (tagsetter);
event_tags = (const GstTagList *) element->event_tags;

GST_LOG_OBJECT (tagsetter, "merging tags, merge mode = %d", merge_mode);
GST_LOG_OBJECT (tagsetter, "event tags: %" GST_PTR_FORMAT, event_tags);
GST_LOG_OBJECT (tagsetter, "set   tags: %" GST_PTR_FORMAT, application_tags);

result = gst_tag_list_merge (application_tags, event_tags, merge_mode);

GST_LOG_OBJECT (tagsetter, "final tags: %" GST_PTR_FORMAT, result);
```

Last reviewed on 2006-05-18 (0.10.6)

36.3 Usage

`gst-tag-setter-merge-tags` (*self* `<gst-tag-setter*>`) (*list* `<gst-tag-list*>`) (*mode* `<gst-tag-merge-mode>`) [Function]

Merges the given list into the setter's list using the given mode.

setter	a `<gst-tag-setter>`
list	a tag list to merge from
mode	the mode to merge with

`gst-tag-setter-get-tag-list` (*self* `<gst-tag-setter*>`) [Function]
⇒ (*ret* `<gst-tag-list*>`)

Returns the current list of tags the setter uses. The list should not be modified or freed.

setter	a `<gst-tag-setter>`
ret	a current snapshot of the taglist used in the setter or NULL if none is used.

`gst-tag-setter-set-tag-merge-mode` (*self* `<gst-tag-setter*>`) [Function]
(*mode* `<gst-tag-merge-mode>`)

Sets the given merge mode that is used for adding tags from events to tags specified by this interface. The default is `<gst-tag-merge-keep>`, which keeps the tags set with this interface and discards tags from events.

setter	a `<gst-tag-setter>`
mode	The mode with which tags are added

`gst-tag-setter-get-tag-merge-mode` (*self* `<gst-tag-setter*>`) [Function]
⇒ (*ret* `<gst-tag-merge-mode>`)

Queries the mode by which tags inside the setter are overwritten by tags from events

setter	a `<gst-tag-setter>`
ret	the merge mode used inside the element.

37 GstTask

Abstraction of GStreamer streaming threads.

37.1 Overview

`<gst-task>` is used by `<gst-element>` and `<gst-pad>` to provide the data passing threads in a `<gst-pipeline>`.

A `<gst-pad>` will typically start a `<gst-task>` to push or pull data to/from the peer pads. Most source elements start a `<gst-task>` to push data. In some cases a demuxer element can start a `<gst-task>` to pull data from a peer element. This is typically done when the demuxer can perform random access on the upstream peer element for improved performance.

Although convenience functions exist on `<gst-pad>` to start/pause/stop tasks, it might sometimes be needed to create a `<gst-task>` manually if it is not related to a `<gst-pad>`.

Before the `<gst-task>` can be run, it needs a `<g-static-rec-mutex>` that can be set with `gst-task-set-lock`.

The task can be started, paused and stopped with `gst-task-start`, `gst-task-pause` and `gst-task-stop` respectively.

A `<gst-task>` will repeadedly call the `<gst-task-function>` with the user data that was provided when creating the task with `gst-task-create`. Before calling the function it will acquire the provided lock.

Stopping a task with `gst-task-stop` will not immediatly make sure the task is not running anymore. Use `gst-task-join` to make sure the task is completely stopped and the thread is stopped.

After creating a `<gst-task>`, use `gst-object-unref` to free its resources. This can only be done it the task is not running anymore.

Last reviewed on 2006-02-13 (0.10.4)

37.2 Usage

`<gst-task>` [Class]

> This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-task-cleanup-all` [Function]

> Wait for all tasks to be stopped. This is mainly used internally to ensure proper cleanup of internal datastructures in testsuites.
>
> MT safe.

`gst-task-create` (*func* `<gst-task-function>`) (*data* `<gpointer>`) [Function]
⇒ (*ret* `<gst-task>`)

> Create a new Task that will repeadedly call the provided *func* with *data* as a parameter. Typically the task will run in a new thread.
>
> The function cannot be changed after the task has been created. You must create a new GstTask to change the function.

func	The `<gst-task-function>` to use
data	User data to pass to *func*
ret	A new `<gst-task>`. MT safe.

`gst-task-get-state` (*self* `<gst-task>`) ⇒ (*ret* `<gst-task-state>`) [Function]
`get-state` [Method]

Get the current state of the task.

task	The `<gst-task>` to query
ret	The `<gst-task-state>` of the task MT safe.

`gst-task-join` (*self* `<gst-task>`) ⇒ (*ret* `bool`) [Function]
`join` [Method]

Joins *task*. After this call, it is safe to unref the task and clean up the lock set with `gst-task-set-lock`.

The task will automatically be stopped with this call.

This function cannot be called from within a task function as this would cause a deadlock. The function will detect this and print a g_warning.

task	The `<gst-task>` to join
ret	TRUE if the task could be joined. MT safe.

`gst-task-pause` (*self* `<gst-task>`) ⇒ (*ret* `bool`) [Function]
`pause` [Method]

Pauses *task*. This method can also be called on a task in the stopped state, in which case a thread will be started and will remain in the paused state. This function does not wait for the task to complete the paused state.

task	The `<gst-task>` to pause
ret	TRUE if the task could be paused. MT safe.

`gst-task-set-lock` (*self* `<gst-task>`) [Function]
(*mutex* `<g-static-rec-mutex*>`)
`set-lock` [Method]

Set the mutex used by the task. The mutex will be acquired before calling the `<gst-task-function>`.

This function has to be called before calling `gst-task-pause` or `gst-task-start`.

MT safe.

task	The `<gst-task>` to use
mutex	The GMutex to use

`gst-task-start` (*self* `<gst-task>`) ⇒ (*ret* `bool`) [Function]
`start` [Method]

Starts *task*. The *task* must have a lock associated with it using `gst-task-set-lock` or thsi function will return FALSE.

task	The `<gst-task>` to start
ret	TRUE if the task could be started. MT safe.

`gst-task-stop` (*self* `<gst-task>`) ⇒ (*ret* `bool`) [Function]
`stop` [Method]

Stops *task*. This method merely schedules the task to stop and will not wait for the task to have completely stopped. Use `gst-task-join` to stop and wait for completion.

task	The `<gst-task>` to stop
ret	TRUE if the task could be stopped. MT safe.

38 GstTrace

Tracing functionality

38.1 Overview

Traces allows to track object allocation. They provide a instance counter per `<g-type>`. The counter is incremented for each object allocated and decremented it when it's freed.

```
// trace un-freed object instances
gst_alloc_trace_set_flags_all (GST_ALLOC_TRACE_LIVE);
if (!gst_alloc_trace_available ()) {
  g_warning ("Trace not available (recompile with trace enabled).");
}
gst_alloc_trace_print_live ();
// do something here
gst_alloc_trace_print_live ();
```

Last reviewed on 2005-11-21 (0.9.5)

38.2 Usage

gst-trace-new (*filename* `mchars`) (*size* `int`) ⇒ (*ret* `<gst-trace*>`) [Function]

Create a ringbuffer of *size* in the file with *filename* to store trace results in.

filename	a filename
size	the max size of the file
ret	a new `<gst-trace>`.

gst-trace-destroy (*self* `<gst-trace*>`) [Function]

Flush an close the previously allocated *trace*.

trace	the `<gst-trace>` to destroy

gst-trace-flush (*self* `<gst-trace*>`) [Function]

Flush any pending trace entries in *trace* to the trace file. *trace* can be NULL in which case the default `<gst-trace>` will be flushed.

trace	the `<gst-trace>` to flush.

gst-trace-text-flush (*self* `<gst-trace*>`) [Function]

Flush any pending trace entries in *trace* to the trace file, formatted as a text line with timestamp and sequence numbers. *trace* can be NULL in which case the default `<gst-trace>` will be flushed.

trace	the `<gst-trace>` to flush.

gst-trace-set-default (*self* `<gst-trace*>`) [Function]

Set the default `<gst-trace>` to *trace*.

trace	the `<gst-trace>` to set as the default.

`gst-trace-read-tsc` ⇒ (*dst* `int64`) [Function]

Read a platform independent timer value that can be used in benchmarks.

dst	pointer to hold the result.

`gst-alloc-trace-available` ⇒ (*ret* `bool`) [Function]

Check if alloc tracing was compiled into the core

ret	TRUE if the core was compiled with alloc tracing enabled.

`gst-alloc-trace-list` ⇒ (*ret* `glist-of`) [Function]

Get a list of all registered alloc trace objects.

ret	a GList of GstAllocTrace objects.

`gst-alloc-trace-live-all` ⇒ (*ret* `int`) [Function]

Get the total number of live registered alloc trace objects.

ret	the total number of live registered alloc trace objects.

`gst-alloc-trace-print-all` [Function]

Print the status of all registered alloc trace objects.

`gst-alloc-trace-set-flags-all` (*flags* `<gst-alloc-trace-flags>`) [Function]

Enable the specified options on all registered alloc trace objects.

flags	the options to enable

`gst-alloc-trace-get` (*name* `mchars`) ⇒ (*ret* `<gst-alloc-trace*>`) [Function]

Get the named alloc trace object.

name	the name of the alloc trace object
ret	a GstAllocTrace with the given name or NULL when no alloc tracer was registered with that name.

`gst-alloc-trace-print` (*self* `<gst-alloc-trace*>`) [Function]

Print the status of the given GstAllocTrace.

trace	the GstAllocTrace to print

`gst-alloc-trace-print-live` [Function]

Print the status of all registered alloc trace objects, ignoring those without live objects.

`gst-alloc-trace-set-flags` (*self* `<gst-alloc-trace*>`) [Function]
    (*flags* `<gst-alloc-trace-flags>`)

Enable the given features on the given GstAllocTrace object.

trace	the GstAllocTrace
flags	flags to set

39 GstTypeFindFactory

Information about registered typefind functions

39.1 Overview

These functions allow querying informations about registered typefind functions. How to create and register these functions is described in the section "Writing typefind functions".

```
typedef struct {
  guint8 *data;
  guint size;
  guint probability;
  GstCaps *data;
} MyTypeFind;
static void
my_peek (gpointer data, gint64 offset, guint size)
{
  MyTypeFind *find = (MyTypeFind *) data;
  if (offset >= 0 && offset + size <= find->size) {
    return find->data + offset;
  }
  return NULL;
}
static void
my_suggest (gpointer data, guint probability, GstCaps *caps)
{
  MyTypeFind *find = (MyTypeFind *) data;
  if (probability > find->probability) {
    find->probability = probability;
    gst_caps_replace (&find->caps, caps);
  }
}
static GstCaps *
find_type (guint8 *data, guint size)
{
  GList *walk, *type_list;
  MyTypeFind find = {data, size, 0, NULL};
  GstTypeFind gst_find = {my_peek, my_suggest, &find, };
  walk = type_list = gst_type_find_factory_get_list ();
  while (walk) {
    GstTypeFindFactory *factory = GST_TYPE_FIND_FACTORY (walk->data);
    walk = g_list_next (walk)
    gst_type_find_factory_call_function (factory, &gst_find);
  }
  g_list_free (type_list);
  return find.caps;
```

```
    };
```

The above example shows how to write a very simple typefinder that identifies the given data. You can get quite a bit more complicated than that though.

Last reviewed on 2005-11-09 (0.9.4)

39.2 Usage

`<gst-type-find-factory>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`gst-type-find-factory-get-list` ⇒ (*ret* `glist-of`) [Function]

Gets the list of all registered typefind factories. You must free the list using gst_plugin_feature_list_free.

ret	the list of all registered `<gst-type-find-factory>`.

`gst-type-find-factory-get-caps` (*self* `<gst-type-find-factory>`) ⇒ (*ret* `<gst-caps>`) [Function]

`get-caps` [Method]

Gets the `<gst-caps>` associated with a typefind factory.

factory	A `<gst-type-find-factory>`
ret	The `<gst-caps>` associated with this factory

40 GstTypeFind

Stream type detection

40.1 Overview

The following functions allow you to detect the media type of an unknown stream.

Last reviewed on 2005-11-09 (0.9.4)

40.2 Usage

`gst-type-find-peek` (*self* `<gst-type-find*>`) (*offset* `int64`) (*size* `unsigned-int`) ⇒ (*ret* `<guint8*>`) [Function]

Returns the *size* bytes of the stream to identify beginning at offset. If offset is a positive number, the offset is relative to the beginning of the stream, if offset is a negative number the offset is relative to the end of the stream. The returned memory is valid until the typefinding function returns and must not be freed.

find	The `<gst-type-find>` object the function was called with
offset	The offset
size	The number of bytes to return
ret	the requested data, or NULL if that data is not available.

`gst-type-find-suggest` (*self* `<gst-type-find*>`) (*probability* `unsigned-int`) (*caps* `<gst-caps>`) [Function]

If a `<gst-type-find-function>` calls this function it suggests the caps with the given probability. A `<gst-type-find-function>` may supply different suggestions in one call. It is up to the caller of the `<gst-type-find-function>` to interpret these values.

find	The `<gst-type-find>` object the function was called with
probability	The probability in percent that the suggestion is right
caps	The fixed `<gst-caps>` to suggest

`gst-type-find-get-length` (*self* `<gst-type-find*>`) ⇒ (*ret* `unsigned-int64`) [Function]

Get the length of the data stream.

find	The `<gst-type-find>` the function was called with
ret	The length of the data stream, or 0 if it is not available.

`gst-type-find-register` (*plugin* `<gst-plugin>`) (*name* `mchars`) (*rank* `unsigned-int`) (*func* `<gst-type-find-function>`) (*possible_caps* `<gst-caps>`) (*data* `<gpointer>`) (*data_notify* `<g-destroy-notify>`) ⇒ (*ret* `bool`) (*extensions* `mchars`) [Function]

Registers a new typefind function to be used for typefinding. After registering this function will be available for typefinding. This function is typically called during an element's plugin initialization.

plugin	A `<gst-plugin>`.
name	The name for registering
rank	The rank (or importance) of this typefind function
func	The `<gst-type-find-function>` to use
extensions	Optional extensions that could belong to this type
possible-caps	Optionally the caps that could be returned when typefinding succeeds
data	Optional user data. This user data must be available until the plugin is unloaded.
data-notify	a `<g-destroy-notify>` that will be called on *data* when the plugin is unloaded.
ret	TRUE on success, FALSE otherwise

41 GstUriHandler

Interface to ease URI handling in plugins.

41.1 Overview

The URIHandler is an interface that is implemented by Source and Sink `<gst-element>` to simplify then handling of URI.

An application can use the following functions to quickly get an element that handles the given URI for reading or writing (`gst-element-make-from-uri`).

Source and Sink plugins should implement this interface when possible.

Last reviewed on 2005-11-09 (0.9.4)

41.2 Usage

`gst-uri-protocol-is-valid` (*protocol* `mchars`) ⇒ (*ret* `bool`) [Function]

Tests if the given string is a valid protocol identifier. Protocols must consist of alphanumeric characters and not start with a number.

protocol	A string
ret	TRUE if the string is a valid protocol identifier, FALSE otherwise.

`gst-uri-is-valid` (*uri* `mchars`) ⇒ (*ret* `bool`) [Function]

Tests if the given string is a valid URI identifier. URIs start with a valid protocol followed by "://" and maybe a string identifying the location.

uri	A URI string
ret	TRUE if the string is a valid URI

`gst-uri-has-protocol` (*uri* `mchars`) (*protocol* `mchars`) ⇒ (*ret* `bool`) [Function]

Checks if the protocol of a given valid URI matches *protocol*.

uri	an URI string
protocol	a protocol string (e.g. "http")
ret	'`#t`' if the protocol matches.

Since 0.10.4

`gst-uri-get-protocol` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Extracts the protocol out of a given valid URI. The returned string must be freed using `g-free`.

uri	A URI string
ret	The protocol for this URI.

`gst-uri-get-location` (*uri* `mchars`) ⇒ (*ret* `mchars`) [Function]

Extracts the location out of a given valid URI, ie. the protocol and "://" are stripped from the URI, which means that the location returned includes the hostname if one is specified. The returned string must be freed using `g-free`.

uri A URI string

ret The location for this URI. Returns NULL if the URI isn't valid. If the URI does not contain a location, an empty string is returned.

`gst-uri-construct` (*protocol* `mchars`) (*location* `mchars`) ⇒ (*ret* `mchars`) [Function]

Constructs a URI for a given valid protocol and location.

protocol Protocol for URI

location Location for URI

ret a new string for this URI. Returns NULL if the given URI protocol is not valid, or the given location is NULL.

`gst-element-make-from-uri` (*type* `<gst-uri-type>`) (*uri* `mchars`) (*elementname* `mchars`) ⇒ (*ret* `<gst-element>`) [Function]

Creates an element for handling the given URI.

type Whether to create a source or a sink

uri URI to create an element for

elementname
Name of created element, can be NULL.

ret a new element or NULL if none could be created

`gst-uri-handler-get-uri-type` (*self* `<gst-uri-handler*>`) ⇒ (*ret* `unsigned-int`) [Function]

Gets the type of the given URI handler

Returns: the `<gst-uri-type>` of the URI handler.

handler A `<gst-uri-handler>`.

ret `<gst-uri-unknown>` if the *handler* isn't implemented correctly.

`gst-uri-handler-get-protocols` (*self* `<gst-uri-handler*>`) ⇒ (*ret* `<gchar**>`) [Function]

Gets the list of protocols supported by *handler*. This list may not be modified.

Returns: the supported protocols.

handler A `<gst-uri-handler>`.

ret NULL if the *handler* isn't implemented properly, or the *handler* doesn't support any protocols.

`gst-uri-handler-get-uri` (*self* `<gst-uri-handler*>`) ⇒ (*ret* `mchars`) [Function]

Gets the currently handled URI.

Returns: the URI currently handled by the *handler*.

handler A `<gst-uri-handler>`

ret NULL if there are no URI currently handled. The returned string must not be modified or freed.

`gst-uri-handler-set-uri` (*self* `<gst-uri-handler*>`) (*uri* `mchars`) [Function]
⇒ (*ret* `bool`)

Tries to set the URI of the given handler.

handler	A `<gst-uri-handler>`
uri	URI to set
ret	TRUE if the URI was set successfully, else FALSE.

`gst-uri-handler-new-uri` (*self* `<gst-uri-handler*>`) (*uri* `mchars`) [Function]

Emits the new-uri signal for a given handler, when that handler has a new URI. This function should only be called by URI handlers themselves.

handler	A `<gst-uri-handler>`
uri	new URI or NULL if it was unset

42 GstUtils

Various utility functions

42.1 Overview

When defining own plugins, use the GST_BOILERPLATE ease gobject creation.

42.2 Usage

`gst-atomic-int-set` (*value* `int`) ⇒ (*atomic_int* `int`) [Function]

Unconditionally sets the atomic integer to *value*.

atomic-int	pointer to an atomic integer
value	value to set

`gst-flow-get-name` (*ret* `<gst-flow-return>`) ⇒ (*ret* `mchars`) [Function]

Gets a string representing the given flow return.

ret	a `<gst-flow-return>` to get the name of.
ret	a static string with the name of the flow return.

`gst-flow-to-quark` (*ret* `<gst-flow-return>`) ⇒ (*ret* `unsigned-int`) [Function]

Get the unique quark for the given GstFlowReturn.

ret	a `<gst-flow-return>` to get the quark of.
ret	the quark associated with the flow return or 0 if an invalid return was specified.

`gst-print-element-args` (*buf* `<g-string*>`) (*indent* `int`) (*element* `<gst-element>`) [Function]

Print the element argument in a human readable format in the given GString.

buf	the buffer to print the args in
indent	initial indentation
element	the element to print the args of

`gst-print-pad-caps` (*buf* `<g-string*>`) (*indent* `int`) (*pad* `<gst-pad>`) [Function]

Write the pad capabilities in a human readable format into the given GString.

buf	the buffer to print the caps in
indent	initial indentation
pad	the pad to print the caps from

43 GstValue

GValue and GParamSpec implementations specific to GStreamer

43.1 Overview

GValue and GParamSpec implementations specific to GStreamer.

Note that operations on the same GstValue (or GValue) from multiple threads may lead to undefined behaviour.

Last reviewed on 2006-03-07 (0.10.4)

43.2 Usage

44 GstVersion

GStreamer version macros.

44.1 Overview

Use the GST_VERSION_* macros e.g. when defining own plugins. The GStreamer runtime checks if these plugin and core version match and refuses to use a plugin compiled against a different version of GStreamer. You can also use the macros to keep the GStreamer version information in your application.

Use the `gst-version` function if you want to know which version of GStreamer you are currently linked against.

The version macros get defined by including "gst/gst.h".

44.2 Usage

45 Gst

Media library supporting arbitrary formats and filter graphs.

45.1 Overview

GStreamer is a framework for constructing graphs of various filters (termed elements here) that will handle streaming media. Any discreet (packetizable) media type is supported, with provisions for automatically determining source type. Formatting/framing information is provided with a powerful negotiation framework. Plugins are heavily used to provide for all elements, allowing one to construct plugins outside of the GST library, even released binary-only if license require (please don't).

GStreamer borrows heavily from both the OGI media pipeline and Microsoft's DirectShow, hopefully taking the best of both and leaving the cruft behind. Its interface is slowly getting stable.

The *GStreamer* library should be initialized with `gst-init` before it can be used. You should pass pointers to the main argc and argv variables so that GStreamer can process its own command line options, as shown in the following example.

```
int
main (int argc, char *argv[])
{
  // initialize the GStreamer library
  gst_init (&argc, &argv);
  ...
}
```

It's allowed to pass two NULL pointers to `gst-init` in case you don't want to pass the command line args to GStreamer.

You can also use GOption to initialize your own parameters as shown in the next code fragment:

```
static gboolean stats = FALSE;
...
int
main (int argc, char *argv[])
{
 GOptionEntry options[] = {
  {"tags", 't', 0, G_OPTION_ARG_NONE, &tags,
      N_("Output tags (also known as metadata)"), NULL},
  {NULL}
 };
 // must initialise the threading system before using any other GLib funtion
 if (!g_thread_supported ())
   g_thread_init (NULL);
 ctx = g_option_context_new ("[ADDITIONAL ARGUMENTS]");
 g_option_context_add_main_entries (ctx, options, GETTEXT_PACKAGE);
```

```
  g_option_context_add_group (ctx, gst_init_get_option_group ());
  if (!g_option_context_parse (ctx, &argc, &argv, &err)) {
    g_print ("Error initializing: %s\n", GST_STR_NULL (err->message));
    exit (1);
  }
  g_option_context_free (ctx);
...
}
```

Use `gst-version` to query the library version at runtime or use the GST_VERSION_* macros to find the version at compile time. Optionally `gst-version-string` returns a printable string.

The `gst-deinit` call is used to clean up all internal resources used by *GStreamer*. It is mostly used in unit tests to check for leaks.

Last reviewed on 2006-08-11 (0.10.10)

45.2 Usage

`gst-init` (*argv* `<char***>`) ⇒ (*argc* `int`) [Function]

Initializes the GStreamer library, setting up internal path lists, registering built-in elements, and loading standard plugins.

This function should be called before calling any other GLib functions. If this is not an option, your program must initialise the GLib thread system using `g-thread-init` before any other GLib functions are called.

This function will terminate your program if it was unable to initialize GStreamer for some reason. If you want your program to fall back, use `gst-init-check` instead.

WARNING: This function does not work in the same way as corresponding functions in other glib-style libraries, such as `gtk-init`. In particular, unknown command line options cause this function to abort program execution.

argc	pointer to application's argc
argv	pointer to application's argv

`gst-init-check` (*argv* `<char***>`) ⇒ (*ret* `bool`) (*argc* `int`) [Function]

Initializes the GStreamer library, setting up internal path lists, registering built-in elements, and loading standard plugins.

This function will return '`#f`' if GStreamer could not be initialized for some reason. If you want your program to fail fatally, use `gst-init` instead.

This function should be called before calling any other GLib functions. If this is not an option, your program must initialise the GLib thread system using `g-thread-init` before any other GLib functions are called.

argc	pointer to application's argc
argv	pointer to application's argv
err	pointer to a `<g-error>` to which a message will be posted on error
ret	'`#t`' if GStreamer could be initialized.

`gst-init-get-option-group` ⇒ (*ret* `<g-option-group*>`) [Function]

Returns a `<g-option-group>` with GStreamer's argument specifications. The group is set up to use standard GOption callbacks, so when using this group in combination with GOption parsing methods, all argument parsing and initialization is automated.

This function is useful if you want to integrate GStreamer with other libraries that use GOption (see `g-option-context-add-group`).

If you use this function, you should make sure you initialise the GLib threading system as one of the very first things in your program (see the example at the beginning of this section).

ret	a pointer to GStreamer's option group.

`gst-deinit` [Function]

Clean up any resources created by GStreamer in `gst-init`.

It is normally not needed to call this function in a normal application as the resources will automatically be freed when the program terminates. This function is therefore mostly used by testsuites and other memory profiling tools.

After this call GStreamer (including this method) should not be used anymore.

`gst-version` ⇒ (*major* `unsigned-int`) (*minor* `unsigned-int`) (*micro* `unsigned-int`) (*nano* `unsigned-int`) [Function]

Gets the version number of the GStreamer library.

major	pointer to a guint to store the major version number
minor	pointer to a guint to store the minor version number
micro	pointer to a guint to store the micro version number
nano	pointer to a guint to store the nano version number

`gst-version-string` ⇒ (*ret* `mchars`) [Function]

This function returns a string that is useful for describing this version of GStreamer to the outside world: user agent strings, logging, ...

ret	a newly allocated string describing this version of GStreamer.

`gst-segtrap-is-enabled` ⇒ (*ret* `bool`) [Function]

Some functions in the GStreamer core might install a custom SIGSEGV handler to better catch and report errors to the application. Currently this feature is enabled by default when loading plugins.

Applications might want to disable this behaviour with the `gst-segtrap-set-enabled` function. This is typically done if the application wants to install its own handler without GStreamer interfering.

ret	'`#t`' if GStreamer is allowed to install a custom SIGSEGV handler.

Since 0.10.10

`gst-segtrap-set-enabled` (*enabled* `bool`) [Function]

Applications might want to disable/enable the SIGSEGV handling of the GStreamer core. See `gst-segtrap-is-enabled` for more information.

enabled whether a custom SIGSEGV handler should be installed.

Since 0.10.10

`gst-registry-fork-is-enabled` ⇒ (*ret* `bool`) [Function]

By default GStreamer will perform a `fork` when scanning and rebuilding the registry file.

Applications might want to disable this behaviour with the `gst-registry-fork-set-enabled` function.

ret '`#t`' if GStreamer will use `fork` when rebuilding the registry. On platforms without `fork`, this function will always return '`#f`'.

Since 0.10.10

`gst-registry-fork-set-enabled` (*enabled* `bool`) [Function]

Applications might want to disable/enable the usage of `fork` when rebuilding the registry. See `gst-registry-fork-is-enabled` for more information.

On platforms without `fork`, this function will have no effect on the return value of `gst-registry-fork-is-enabled`.

enabled whether rebuilding the registry may fork

Since 0.10.10

46 GstXML

XML save/restore operations of pipelines

46.1 Overview

GStreamer pipelines can be saved to xml files using `gst-xml-write-file`. They can be loaded back using `gst-xml-parse-doc` / `gst-xml-parse-file` / `gst-xml-parse-memory`. Additionally one can load saved pipelines into the gst-editor to inspect the graph.

`<gst-element>` implementations need to override `gst-object-save-thyself` and `gst-object-restore-thyself`.

46.2 Usage

`<gst-xml>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`object-loaded` (*arg0* `<gst-object>`) (*arg1* `<gpointer>`) [Signal on `<gst-xml>`]

Signals that a new object has been deserialized.

`gst-xml-write` (*element* `<gst-element>`) ⇒ (*ret* `<xml-doc-ptr>`) [Function]

Converts the given element into an XML presentation.

element	The element to write out
ret	a pointer to an XML document

`gst-xml-write-file` (*element* `<gst-element>`) (*out* `<file*>`) ⇒ (*ret* `int`) [Function]

Converts the given element into XML and writes the formatted XML to an open file.

element	The element to write out
out	an open file, like stdout
ret	number of bytes written on success, -1 otherwise.

`gst-xml-new` ⇒ (*ret* `<gst-xml>`) [Function]

Create a new GstXML parser object.

ret	a pointer to a new GstXML object.

`gst-xml-parse-doc` (*self* `<gst-xml>`) (*doc* `<xml-doc-ptr>`) (*root* `<guchar*>`) ⇒ (*ret* `bool`) [Function]

`parse-doc` [Method]

Fills the GstXML object with the elements from the xmlDocPtr.

xml	a pointer to a GstXML object
doc	a pointer to an xml document to parse
root	The name of the root object to build
ret	TRUE on success, FALSE otherwise

`gst-xml-parse-file` (*self* `<gst-xml>`) (*fname* `<guchar*>`) (*root* `<guchar*>`) ⇒ (*ret* `bool`) [Function]

`parse-file` [Method]

Fills the GstXML object with the corresponding elements from the XML file fname. Optionally it will only build the element from the element node root (if it is not NULL). This feature is useful if you only want to build a specific element from an XML file but not the pipeline it is embedded in.

Pass `"-"` as fname to read from stdin. You can also pass a URI of any format that libxml supports, including http.

xml	a pointer to a GstXML object
fname	The filename with the xml description
root	The name of the root object to build
ret	TRUE on success, FALSE otherwise

`gst-xml-parse-memory` (*self* `<gst-xml>`) (*buffer* `<guchar*>`) (*size* `unsigned-int`) (*root* `mchars`) ⇒ (*ret* `bool`) [Function]

`parse-memory` [Method]

Fills the GstXML object with the corresponding elements from an in memory XML buffer.

xml	a pointer to a GstXML object
buffer	a pointer to the in memory XML buffer
size	the size of the buffer
root	the name of the root objects to build
ret	TRUE on success

`gst-xml-get-element` (*self* `<gst-xml>`) (*name* `<guchar*>`) ⇒ (*ret* `<gst-element>`) [Function]

`get-element` [Method]

This function is used to get a pointer to the GstElement corresponding to name in the pipeline description. You would use this if you have to do anything to the element after loading.

xml	The GstXML to get the element from
name	The name of element to retrieve
ret	a pointer to a new GstElement, caller owns returned reference.

`gst-xml-get-topelements` (*self* `<gst-xml>`) ⇒ (*ret* `glist-of`) [Function]

`get-topelements` [Method]

Retrieve a list of toplevel elements.

xml	The GstXML to get the elements from
ret	a GList of top-level elements. The caller does not own a copy of the list and must not free or modify the list. The caller also does not own a reference to any of the elements in the list and should obtain its own reference using `gst-object-ref` if necessary.

gst-xml-make-element (*cur* `<xml-node-ptr>`) (*parent* `<gst-object>`) ⇒ (*ret* `<gst-element>`) [Function]

Load the element from the XML description

cur	the xml node
parent	the parent of this object when it's loaded
ret	the new element

Concept Index

(Index is nonexistent)

Function Index