GstElement

GstElement — Abstract base class for all pipeline elements

Functions

#define	GST_STATE()
#define	GST_STATE_GET_NEXT()
#define	GST_STATE_NEXT()
#define	GST_STATE_PENDING()
#define	GST_STATE_RETURN()
#define	GST_STATE_TARGET()
#define	GST_STATE_TRANSITION()
#define	GST_STATE_TRANSITION_CURRENT()
#define	GST_STATE_TRANSITION_NEXT()
#define	GST_STATE_GET_LOCK()
#define	GST_STATE_GET_COND()
#define	GST_ELEMENT_NAME()
#define	GST_ELEMENT_PARENT()
#define	GST_ELEMENT_BUS()
#define	GST_ELEMENT_CLOCK()
#define	GST_ELEMENT_PADS()
#define	GST_ELEMENT_START_TIME()
#define	GST_ELEMENT_ERROR()
#define	GST_ELEMENT_WARNING()
#define	GST_ELEMENT_INFO()
#define	GST_ELEMENT_IS_LOCKED_STATE()
void	gst_element_class_add_pad_template ()
GstPadTemplate *	gst_element_class_get_pad_template ()
GList *	gst_element_class_get_pad_template_list ()
void	gst_element_class_set_metadata ()
void	gst_element_class_set_static_metadata ()
void	gst_element_class_add_metadata ()
void	gst_element_class_add_static_metadata ()
gboolean	gst_element_add_pad ()
void	gst_element_create_all_pads ()
GstPad *	gst_element_get_compatible_pad ()
GstPadTemplate *	gst_element_get_compatible_pad_template ()
GstPad *	gst_element_get_request_pad ()
GstPad *	gst_element_get_static_pad ()
GstPad *	gst_element_request_pad ()
void	gst_element_no_more_pads ()
void	gst_element_release_request_pad ()
gboolean	gst_element_remove_pad ()
GstIterator *	gst_element_iterate_pads ()
GstIterator *	gst_element_iterate_sink_pads ()
GstIterator *	gst_element_iterate_src_pads ()
gboolean	gst_element_link ()
void	gst_element_unlink ()
gboolean	gst_element_link_many ()
void	gst_element_unlink_many ()

gboolean	gst_element_link_pads ()
gboolean	gst_element_link_pads_full ()
void	gst_element_unlink_pads ()
gboolean	gst_element_link_pads_filtered ()
gboolean	gst_element_link_filtered ()
const gchar *	gst_element_class_get_metadata ()
void	gst_element_set_base_time ()
GstClockTime	gst_element_get_base_time ()
void	gst_element_set_start_time ()
GstClockTime	gst_element_get_start_time ()
void	gst_element_set_bus ()
GstBus *	gst_element_get_bus ()
void	gst_element_set_context ()
GstElementFactory *	gst_element_get_factory ()
#define	gst_element_set_name()
#define	gst_element_get_name()
#define	gst_element_set_parent()
#define	gst_element_get_parent()
gboolean	gst_element_set_clock ()
GstClock *	gst_element_get_clock ()
GstClock *	gst_element_provide_clock ()
GstStateChangeReturn	gst_element_set_state ()
GstStateChangeReturn	gst_element_get_state ()
gboolean	gst_element_set_locked_state ()
gboolean	gst_element_is_locked_state ()
void	gst_element_abort_state ()
GstStateChangeReturn	gst_element_continue_state ()
void	gst_element_lost_state ()
const gchar *	gst_element_state_get_name ()
const gchar *	gst_element_state_change_return_get_name ()
gboolean	gst_element_sync_state_with_parent ()
GstStateChangeReturn	gst_element_change_state ()
void	gst_element_message_full ()
gboolean	gst_element_post_message ()
gboolean	gst_element_query ()
gboolean	gst_element_query_convert ()
gboolean	gst_element_query_position ()
gboolean	gst_element_query_duration ()
gboolean	gst_element_send_event ()
gboolean	gst_element_seek_simple ()
gboolean	gst_element_seek ()

Signals

void	no-more-pads	Run Last
void	pad-added	Run Last
void	pad-removed	Run Last

Types and Values

struct	GstElement
struct	GstElementClass
enum	GstElementFlags
enum	GstState
enum	GstStateChange
enum	GstStateChangeReturn
#define	GST_ELEMENT_METADATA_AUTHOR
#define	GST_ELEMENT_METADATA_DESCRIPTION
#define	GST_ELEMENT_METADATA_DOC_URI
#define	GST_ELEMENT_METADATA_ICON_NAME
#define	GST_ELEMENT_METADATA_KLASS
#define	GST_ELEMENT_METADATA_LONGNAME

Object Hierarchy

```
    GObject
    ╰── GInitiallyUnowned
        ╰── GstObject
        ╰── GstElement
            ╰── GstBin
```

Known Derived Interfaces

GstElement is required by GstTagSetter.

Includes

```
#include <gst/gst.h>
```

Description

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 GstElement subclasses.

The name of a GstElement 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.

Elements can have pads (of the type GstPad). These pads link to pads on other elements. GstBuffer flow between these linked pads. A GstElement has a GList of GstPad 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()`.

An existing pad of an element can be retrieved by name with `gst_element_get_static_pad()`. A new dynamic pad can be created using `gst_element_request_pad()` with a GstPadTemplate. 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 GstCaps. 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 GstState). You can get and set the state of an element with `gst_element_get_state()` and `gst_element_set_state()`. Setting a state triggers a GstStateChange. To get a string representation of a GstState, use `gst_element_state_get_name()`.

You can get and set a GstClock on an element using `gst_element_get_clock()` and `gst_element_set_clock()`. Some elements can provide a clock for the pipeline if the GST_ELEMENT_FLAG_PROVIDE_CLOCK flag is set. 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 the `GST_ELEMENT_FLAG_REQUIRE_CLOCK()` flag is set, a clock should be set on the element with `gst_element_set_clock()`.

Note that clock selection and distribution is normally handled by the toplevel GstPipeline so the clock functions are only to be used in very specific situations.

Functions

GST_STATE()

```
#define GST_STATE(elem)                  (GST_ELEMENT_CAST(elem)->current_state)
```

This macro returns the current GstState of the element.

Parameters

elem	a GstElement to return state for.

GST_STATE_GET_NEXT()

```
#define GST_STATE_GET_NEXT(cur,pending)         ((GstState)((cur) + __GST_SIGN ((gint)(pending) - (gint)(cur)))
```

Given a current state `cur` and a target state `pending` , calculate the next (intermediate) GstState.

Parameters

cur	A starting GstState
pending	A target GstState

GST_STATE_NEXT()

```
#define GST_STATE_NEXT(elem)            (GST_ELEMENT_CAST(elem)->next_state)
```

This macro returns the next GstState of the element.

Parameters

elem	a GstElement to return the next state for.

GST_STATE_PENDING()

```
#define GST_STATE_PENDING(elem)         (GST_ELEMENT_CAST(elem)->pending_state)
```

This macro returns the currently pending GstState of the element.

Parameters

elem	a GstElement to return the pending state for.

GST_STATE_RETURN()

```
#define GST_STATE_RETURN(elem)          (GST_ELEMENT_CAST(elem)->last_return)
```

This macro returns the last GstStateChangeReturn value.

Parameters

elem	a GstElement to return the last state result for.

GST_STATE_TARGET()

```
#define GST_STATE_TARGET(elem)          (GST_ELEMENT_CAST(elem)->target_state)
```

This macro returns the target GstState of the element.

Parameters

elem	a GstElement to return the target state for.

GST_STATE_TRANSITION()

```
#define GST_STATE_TRANSITION(cur,next)          ((GstStateChange)(((cur)<<3)|(next)))
```

Given a current state `cur` and a next state `next` , calculate the associated GstStateChange transition.

Parameters

cur	A current state
next	A next state

GST_STATE_TRANSITION_CURRENT()

```
#define GST_STATE_TRANSITION_CURRENT(trans)     ((GstState)((trans)>>3))
```

Given a state transition `trans` , extract the current GstState.

Parameters

trans	A GstStateChange

GST_STATE_TRANSITION_NEXT()

```
#define GST_STATE_TRANSITION_NEXT(trans)        ((GstState)((trans)&0x7))
```

Given a state transition *trans* , extract the next GstState.

Parameters

trans	A GstStateChange

GST_STATE_GET_LOCK()

```
#define GST_STATE_GET_LOCK(elem)               (&(GST_ELEMENT_CAST(elem)->state_lock))
```

Get a reference to the state lock of *elem* . This lock is used by the core. It is taken while getting or setting the state, during state changes, and while finalizing.

Parameters

elem	a GstElement

GST_STATE_GET_COND()

```
#define GST_STATE_GET_COND(elem)               (&GST_ELEMENT_CAST(elem)->state_cond)
```

Get the conditional used to signal the completion of a state change.

Parameters

elem	a GstElement

GST_ELEMENT_NAME()

```
#define GST_ELEMENT_NAME(elem)                  (GST_OBJECT_NAME(elem))
```

Gets the name of this element. Use only in core as this is not ABI-compatible. Others use `gst_element_get_name()`

Parameters

elem	A GstElement to query

GST_ELEMENT_PARENT()

```
#define GST_ELEMENT_PARENT(elem)                (GST_ELEMENT_CAST(GST_OBJECT_PARENT(elem)))
```

Get the parent object of this element.

Parameters

elem	A GstElement to query

GST_ELEMENT_BUS()

```
#define GST_ELEMENT_BUS(elem)                   (GST_ELEMENT_CAST(elem)->bus)
```

Get the message bus of this element.

Parameters

elem	A GstElement to query

GST_ELEMENT_CLOCK()

```
#define GST_ELEMENT_CLOCK(elem)                 (GST_ELEMENT_CAST(elem)->clock)
```

Get the clock of this element

Parameters

elem	A GstElement to query

GST_ELEMENT_PADS()

```
#define GST_ELEMENT_PADS(elem)                  (GST_ELEMENT_CAST(elem)->pads)
```

Get the pads of this elements.

Parameters

elem	A GstElement to query

GST_ELEMENT_START_TIME()

```
#define GST_ELEMENT_START_TIME(elem)            (GST_ELEMENT_CAST(elem)->start_time)
```

This macro returns the start_time of the *`elem`*. The start_time is the running_time of the pipeline when the element went to PAUSED.

Parameters

elem	a GstElement to return the start time for.

GST_ELEMENT_ERROR()

```
#define             GST_ELEMENT_ERROR(el, domain, code, text, debug)
```

Utility function that elements can use in case they encountered a fatal data processing error. The pipeline will post an error message and the application will be requested to stop further media processing.

Parameters

el	the element that generates the error
domain	like CORE, LIBRARY, RESOURCE or STREAM (see gstreamer-GstGError)
code	error code defined for that domain (see gstreamer-GstGError)
text	the message to display (format string and args enclosed in parentheses)
debug	debugging information for the message (format string and args enclosed in parentheses)

GST_ELEMENT_WARNING()

```
#define             GST_ELEMENT_WARNING(el, domain, code, text, debug)
```

Utility function that elements can use in case they encountered a non-fatal data processing problem. The pipeline will post a warning message and the application will be informed.

Parameters

el	the element that generates the warning
domain	like CORE, LIBRARY, RESOURCE or STREAM (see gstreamer-GstGError)
code	error code defined for that domain (see gstreamer-GstGError)
text	the message to display (format string and args enclosed in

	parentheses)
debug	debugging information for the message (format string and args enclosed in parentheses)

GST_ELEMENT_INFO()

```
#define             GST_ELEMENT_INFO(el, domain, code, text, debug)
```

Utility function that elements can use in case they want to inform the application of something noteworthy that is not an error. The pipeline will post a info message and the application will be informed.

Parameters

el	the element that generates the information
domain	like CORE, LIBRARY, RESOURCE or STREAM (see gstreamer-GstGError)
code	error code defined for that domain (see gstreamer-GstGError)
text	the message to display (format string and args enclosed in parentheses)
debug	debugging information for the message (format string and args enclosed in parentheses)

GST_ELEMENT_IS_LOCKED_STATE()

```
#define GST_ELEMENT_IS_LOCKED_STATE(elem)        (GST_OBJECT_FLAG_IS_SET(elem,GST_ELEMENT_FLAG_LOCKED_STATE))
```

Check if the element is in the locked state and therefore will ignore state changes from its parent object.

Parameters

elem	A GstElement to query

gst_element_class_add_pad_template ()

```
void
gst_element_class_add_pad_template (GstElementClass *klass,
                                    GstPadTemplate *templ);
```

Adds a padtemplate to an element class. This is mainly used in the _class_init functions of classes. If a pad template with the same name as an already existing one is added the old one is replaced by the new one.

Parameters

klass	the GstElementClass to add the pad template to.	
templ	a GstPadTemplate to add to the element class.	[transfer full]

gst_element_class_get_pad_template ()

```
GstPadTemplate *
gst_element_class_get_pad_template (GstElementClass *element_class,
                                    const gchar *name);
```

Retrieves a padtemplate from `element_class` with the given name.

If you use this function in the GInstanceInitFunc of an object class that has subclasses, make sure to pass the g_class parameter of the GInstanceInitFunc here.

Parameters

element_class	a GstElementClass to get the pad template of.

name	the name of the GstPadTemplate to get.

Returns

the GstPadTemplate with the given name, or `NULL` if none was found. No unreferencing is necessary.

[transfer none][nullable]

gst_element_class_get_pad_template_list ()

```
GList *
gst_element_class_get_pad_template_list
                                (GstElementClass *element_class);
```

Retrieves a list of the pad templates associated with *`element_class`* . The list must not be modified by the calling code.

If you use this function in the GInstanceInitFunc of an object class that has subclasses, make sure to pass the g_class parameter of the GInstanceInitFunc here.

Parameters

element_class	a GstElementClass to get pad templates of.

Returns

the GList of pad templates.

[transfer none][element-type Gst.PadTemplate]

gst_element_class_set_metadata ()

```
void
gst_element_class_set_metadata (GstElementClass *klass,
                                const gchar *longname,
                                const gchar *classification,
                                const gchar *description,
                                const gchar *author);
```

Sets the detailed information for a GstElementClass.

This function is for use in _class_init functions only.

Parameters

klass	class to set metadata for
longname	The long English name of the element. E.g. "File Sink"
classification	String describing the type of element, as an unordered list separated with slashes ('/'). See draft-klass.txt of the design docs for more details and common types. E.g: "Sink/File"
description	Sentence describing the purpose of the element. E.g: "Write stream to a file"
author	Name and contact details of the author(s). Use \n to separate multiple author metadata. E.g: "Joe Bloggs <joe.blogs at foo.com>"

gst_element_class_set_static_metadata ()

```
void
gst_element_class_set_static_metadata (GstElementClass *klass,
                                       const gchar *longname,
                                       const gchar *classification,
                                       const gchar *description,
                                       const gchar *author);
```

Sets the detailed information for a GstElementClass.

This function is for use in _class_init functions only.

Same as `gst_element_class_set_metadata()`, but *`longname`* , *`classification`* , *`description`* , and *`author`* must be static strings or inlined strings, as they will not be copied. (GStreamer plugins will be made resident once loaded, so this function can be used even from dynamically loaded plugins.)

Parameters

klass	class to set metadata for
longname	The long English name of the element. E.g. "File Sink"
classification	String describing the type of element, as an unordered list separated with slashes ('/'). See draft-klass.txt of the design docs for more details and common types. E.g: "Sink/File"
description	Sentence describing the purpose of the element. E.g: "Write stream to a file"
author	Name and contact details of the author(s). Use \n to separate multiple author metadata. E.g: "Joe Bloggs <joe.blogs at foo.com>"

gst_element_class_add_metadata ()

```
void
gst_element_class_add_metadata (GstElementClass *klass,
                                const gchar *key,
                                const gchar *value);
```

Set *`key`* with *`value`* as metadata in *`klass`* .

Parameters

klass	class to set metadata for
key	the key to set
value	the value to set

gst_element_class_add_static_metadata ()

```
void
gst_element_class_add_static_metadata (GstElementClass *klass,
                                       const gchar *key,
                                       const gchar *value);
```

Set *`key`* with *`value`* as metadata in *`klass`* .

Same as `gst_element_class_add_metadata()`, but *`value`* must be a static string or an inlined string, as it will not be copied. (GStreamer plugins will be made resident once loaded, so this function can be used even from dynamically loaded plugins.)

Parameters

klass	class to set metadata for
key	the key to set
value	the value to set

gst_element_add_pad ()

```
gboolean
gst_element_add_pad (GstElement *element,
                     GstPad *pad);
```

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 “pad-added” signal on the element.

Parameters

element	a GstElement to add the pad to.	
pad	the GstPad to add to the element.	[transfer full]

Returns

`TRUE` 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_create_all_pads ()

```
void
gst_element_create_all_pads (GstElement *element);
```

Creates a pad for each pad template that is always available. This function is only useful during object initialization of subclasses of GstElement.

Parameters

element	a GstElement to create pads for.	[transfer none]

gst_element_get_compatible_pad ()

```
GstPad *
gst_element_get_compatible_pad (GstElement *element,
                                GstPad *pad,
                                GstCaps *caps);
```

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.

This function will first attempt to find a compatible unlinked ALWAYS pad, and if none can be found, it will request a compatible REQUEST pad by looking at the templates of *`element`* .

Parameters

element	a GstElement in which the pad should be found.	[transfer none]
pad	the GstPad to find a compatible one for.	[transfer none]
caps	the GstCaps to use as a filter.	[allow-none]

Returns

the GstPad to which a link can be made, or `NULL` if one cannot be found. `gst_object_unref()` after usage.

[transfer full][nullable]

gst_element_get_compatible_pad_template ()

```
GstPadTemplate *
gst_element_get_compatible_pad_template
                               (GstElement *element,
                                GstPadTemplate *compattempl);
```

Retrieves a pad template from *`element`* that is compatible with *`compattempl`* . Pads from compatible templates can be linked together.

Parameters

element	a GstElement to get a compatible pad template for.	[transfer none]
compattempl	the GstPadTemplate to find a compatible template for.	[transfer none]

Returns

a compatible GstPadTemplate, or NULL if none was found. No unreferencing is necessary.

[transfer none][nullable]

gst_element_get_request_pad ()

```
GstPad *
gst_element_get_request_pad (GstElement *element,
                             const gchar *name);
```

Retrieves a pad from the element by name (e.g. "src_%d"). This version only retrieves request pads. The pad should be released with `gst_element_release_request_pad()`.

This method is slower than manually getting the pad template and calling `gst_element_request_pad()` if the pads should have a specific name (e.g. *name* is "src_1" instead of "src_u").

Parameters

element	a GstElement to find a request pad of.
name	the name of the request GstPad to retrieve.

Returns

requested GstPad if found, otherwise NULL. Release after usage.

[transfer full][nullable]

gst_element_get_static_pad ()

```
GstPad *
gst_element_get_static_pad (GstElement *element,
                            const gchar *name);
```

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

Parameters

element	a GstElement to find a static pad of.
name	the name of the static GstPad to retrieve.

Returns

the requested GstPad if found, otherwise NULL. unref after usage.

MT safe.

[transfer full][nullable]

gst_element_request_pad ()

```
GstPad *
gst_element_request_pad (GstElement *element,
                         GstPadTemplate *templ,
                         const gchar *name,
                         const GstCaps *caps);
```

Retrieves a request pad from the element according to the provided template. Pad templates can be looked up using `gst_element_factory_get_static_pad_templates()`.

The pad should be released with `gst_element_release_request_pad()`.

Parameters

element	a GstElement to find a request pad of.	
templ	a GstPadTemplate of which we want a pad of.	
name	the name of the request GstPad to retrieve. Can be `NULL`.	[transfer none][allow-none]
caps	the caps of the pad we want to request. Can be `NULL`.	[transfer none][allow-none]

Returns

requested GstPad if found, otherwise `NULL`. Release after usage.

[transfer full][nullable]

gst_element_no_more_pads ()

```
void
gst_element_no_more_pads (GstElement *element);
```

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 “no-more-pads” signal.

MT safe.

Parameters

element	a GstElement

gst_element_release_request_pad ()

```
void
gst_element_release_request_pad (GstElement *element,
                                 GstPad *pad);
```

Makes the element free the previously requested pad as obtained with `gst_element_request_pad()`.

This does not unref the pad. If the pad was created by using `gst_element_request_pad()`, `gst_element_release_request_pad()` needs to be followed by `gst_object_unref()` to free the *pad* .

MT safe.

Parameters

element	a GstElement to release the request pad of.
pad	the GstPad to release.

gst_element_remove_pad ()

```
gboolean
gst_element_remove_pad (GstElement *element,
                        GstPad *pad);
```

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_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 “pad-removed” signal on the element.

Parameters

element	a GstElement to remove pad from.	
pad	the GstPad to remove from the element.	[transfer full]

Returns

TRUE if the pad could be removed. Can return FALSE if the pad does not belong to the provided element.

MT safe.

gst_element_iterate_pads ()

```
GstIterator *
gst_element_iterate_pads (GstElement *element);
```

Retrieves an iterator of *element* 's pads. The iterator should be freed after usage. Also more specialized iterators exists such as `gst_element_iterate_src_pads()` or `gst_element_iterate_sink_pads()`.

The order of pads returned by the iterator will be the order in which the pads were added to the element.

Parameters

element	a GstElement to iterate pads of.

Returns

the GstIterator of GstPad.

MT safe.

[transfer full]

gst_element_iterate_sink_pads ()

```
GstIterator *
gst_element_iterate_sink_pads (GstElement *element);
```

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

The order of pads returned by the iterator will be the order in which the pads were added to the element.

Parameters

element	a GstElement.

Returns

the GstIterator of GstPad.

MT safe.

[transfer full]

gst_element_iterate_src_pads ()

```
GstIterator *
gst_element_iterate_src_pads (GstElement *element);
```

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

The order of pads returned by the iterator will be the order in which the pads were added to the element.

Parameters

element	a GstElement.

Returns

the GstIterator of GstPad.

MT safe.

[transfer full]

gst_element_link ()

```
gboolean
gst_element_link (GstElement *src,
                  GstElement *dest);
```

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 manually 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.

Parameters

src	a GstElement containing the source pad.	[transfer none]
dest	the GstElement containing the destination pad.	[transfer none]

Returns

`TRUE` if the elements could be linked, `FALSE` otherwise.

gst_element_unlink ()

```
void
gst_element_unlink (GstElement *src,
                    GstElement *dest);
```

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 requestpad, which has to be released using `gst_element_release_request_pad()`.

Parameters

src	the source GstElement to unlink.	[transfer none]
dest	the sink GstElement to unlink.	[transfer none]

gst_element_link_many ()

```
gboolean
gst_element_link_many (GstElement *element_1,
                       GstElement *element_2,
                       ...);
```

Chain together a series of elements. Uses `gst_element_link()`. Make sure you have added your elements to a bin or pipeline with `gst_bin_add()` before trying to link them.

Parameters

element_1	the first GstElement in the link chain.	[transfer none]
element_2	the second GstElement in the link chain.	[transfer none]

...	the NULL-terminated list of elements to link in order.	

Returns

TRUE on success, FALSE otherwise.

gst_element_unlink_many ()

```
void
gst_element_unlink_many (GstElement *element_1,
                         GstElement *element_2,
                         ...);
```

Unlinks a series of elements. Uses `gst_element_unlink()`.

Parameters

element_1	the first GstElement in the link chain.	[transfer none]
element_2	the second GstElement in the link chain.	[transfer none]
...	the NULL-terminated list of elements to unlink in order.	

gst_element_link_pads ()

```
gboolean
gst_element_link_pads (GstElement *src,
                       const gchar *srcpadname,
                       GstElement *dest,
                       const gchar *destpadname);
```

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.

Parameters

src	a GstElement containing the source pad.	
srcpadname	the name of the GstPad in source element or NULL for any pad.	[allow-none]
dest	the GstElement containing the destination pad.	[transfer none]
destpadname	the name of the GstPad in destination element, or NULL for any pad.	[allow-none]

Returns

TRUE if the pads could be linked, FALSE otherwise.

gst_element_link_pads_full ()

```
gboolean
gst_element_link_pads_full (GstElement *src,
                            const gchar *srcpadname,
                            GstElement *dest,
                            const gchar *destpadname,
                            GstPadLinkCheck flags);
```

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.

Calling `gst_element_link_pads_full()` with *flags* == `GST_PAD_LINK_CHECK_DEFAULT` is the same as calling `gst_element_link_pads()` and the recommended way of linking pads with safety checks applied.

This is a convenience function for `gst_pad_link_full()`.

Parameters

src	a GstElement containing the source pad.	
srcpadname	the name of the GstPad in source element or `NULL` for any pad.	[allow-none]
dest	the GstElement containing the destination pad.	[transfer none]
destpadname	the name of the GstPad in destination element, or `NULL` for any pad.	[allow-none]
flags	the GstPadLinkCheck to be performed when linking pads.	

Returns

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

gst_element_unlink_pads ()

```
void
gst_element_unlink_pads (GstElement *src,
                         const gchar *srcpadname,
                         GstElement *dest,
                         const gchar *destpadname);
```

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

This is a convenience function for `gst_pad_unlink()`.

Parameters

src	a (transfer none): GstElement containing the source pad.	
srcpadname	the name of the GstPad in source element.	
dest	a GstElement containing the destination pad.	[transfer none]
destpadname	the name of the GstPad in destination element.	

gst_element_link_pads_filtered ()

```
gboolean
gst_element_link_pads_filtered (GstElement *src,
                                const gchar *srcpadname,
                                GstElement *dest,
                                const gchar *destpadname,
                                GstCaps *filter);
```

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 `NULL`, makes sure that the caps of the link is a subset of *caps* .

Parameters

src	a GstElement containing the source pad.	
srcpadname	the name of the GstPad in source element or `NULL` for any pad.	[allow-none]
dest	the GstElement containing the destination pad.	[transfer none]
destpadname	the name of the GstPad in destination element or `NULL` for any pad.	[allow-none]
filter	the GstCaps to filter the link, or `NULL` for no filter.	[transfer none][allow-none]

Returns

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

gst_element_link_filtered ()

```
gboolean
gst_element_link_filtered (GstElement *src,
                           GstElement *dest,
                           GstCaps *filter);
```

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.

Parameters

src	a GstElement containing the source pad.	
dest	the GstElement containing the destination pad.	[transfer none]
filter	the GstCaps to filter the link, or `NULL` for no filter.	[transfer none][allow-none]

Returns

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

gst_element_class_get_metadata ()

```
const gchar *
gst_element_class_get_metadata (GstElementClass *klass,
                                const gchar *key);
```

Get metadata with *key* in *klass* .

Parameters

klass	class to get metadata for
key	the key to get

Returns

the metadata for *key* .

gst_element_set_base_time ()

```
void
gst_element_set_base_time (GstElement *element,
                           GstClockTime time);
```

Set the base time of an element. See `gst_element_get_base_time()`.

MT safe.

Parameters

element	a GstElement.
time	the base time to set.

gst_element_get_base_time ()

```
GstClockTime
gst_element_get_base_time (GstElement *element);
```

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 running time of the element.

Parameters

element	a GstElement.

Returns

the base time of the element.

MT safe.

gst_element_set_start_time ()

```
void
gst_element_set_start_time (GstElement *element,
                            GstClockTime time);
```

Set the start time of an element. The start time of the element is the running time of the element when it last went to the PAUSED state. In READY or after a flushing seek, it is set to 0.

Toplevel elements like GstPipeline will manage the start_time and base_time on its children. Setting the start_time to GST_CLOCK_TIME_NONE on such a toplevel element will disable the distribution of the base_time to the children and can be useful if the application manages the base_time itself, for example if you want to synchronize capture from multiple pipelines, and you can also ensure that the pipelines have the same clock.

MT safe.

Parameters

element	a GstElement.
time	the base time to set.

gst_element_get_start_time ()

```
GstClockTime
gst_element_get_start_time (GstElement *element);
```

Returns the start time of the element. The start time is the running time of the clock when this element was last put to PAUSED.

Usually the start_time is managed by a toplevel element such as GstPipeline.

MT safe.

Parameters

element	a GstElement.

Returns

the start time of the element.

gst_element_set_bus ()

```
void
gst_element_set_bus (GstElement *element,
                     GstBus *bus);
```

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

MT safe.

Parameters

element	a GstElement to set the bus of.	
bus	the GstBus to set.	[transfer none]

gst_element_get_bus ()

```
GstBus *
gst_element_get_bus (GstElement *element);
```

Returns the bus of the element. Note that only a GstPipeline will provide a bus for the application.

Parameters

element	a GstElement to get the bus of.

Returns

the element's GstBus. unref after usage.

MT safe.

[transfer full]

gst_element_set_context ()

```
void
gst_element_set_context (GstElement *element,
                         GstContext *context);
```

Sets the context of the element. Increases the refcount of the context.

MT safe.

Parameters

element	a GstElement to set the context of.	
context	the GstContext to set.	[transfer none]

gst_element_get_factory ()

```
GstElementFactory *
gst_element_get_factory (GstElement *element);
```

Retrieves the factory that was used to create this element.

Parameters

element	a GstElement to request the element factory of.

Returns

the GstElementFactory used for creating this element. no refcounting is needed.

[transfer none]

gst_element_set_name()

```
#define             gst_element_set_name(elem,name) gst_object_set_name(GST_OBJECT_CAST(elem),name)
```

Sets the name of the element, getting rid of the old name if there was one.

Parameters

elem	a GstElement to set the name of.
name	the new name

gst_element_get_name()

```
#define             gst_element_get_name(elem)      gst_object_get_name(GST_OBJECT_CAST(elem))
```

Returns a copy of the name of `elem` . Caller should `g_free()` the return value after usage. For a nameless element, this returns `NULL`, which you can safely `g_free()` as well.

Parameters

elem	a GstElement to get the name of `elem` .

Returns

the name of `elem` . `g_free()` after usage. MT safe.

[transfer full][nullable]

gst_element_set_parent()

```
#define             gst_element_set_parent(elem,parent)     gst_object_set_parent(GST_OBJECT_CAST(elem),par
```

Sets the parent of an element.

Parameters

elem	a GstElement to set the parent of.
parent	the new parent GstObject of the element.

gst_element_get_parent()

```
#define             gst_element_get_parent(elem)    gst_object_get_parent(GST_OBJECT_CAST(elem))
```

Get the parent of an element.

Parameters

elem	a GstElement to get the parent of.

Returns

the parent of an element.

[transfer full]

gst_element_set_clock ()

```
gboolean
gst_element_set_clock (GstElement *element,
                       GstClock *clock);
```

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

Parameters

element	a GstElement to set the clock for.
clock	the GstClock to set for the element.

Returns

TRUE 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 ()

```
GstClock *
gst_element_get_clock (GstElement *element);
```

Gets the currently configured clock of the element. This is the clock as was last set with `gst_element_set_clock()`.

Parameters

element	a GstElement to get the clock of.

Returns

the GstClock of the element. unref after usage.

MT safe.

[transfer full]

gst_element_provide_clock ()

```
GstClock *
gst_element_provide_clock (GstElement *element);
```

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.

Parameters

element	a GstElement to query

Returns

the GstClock provided by the element or `NULL` if no clock could be provided. Unref after usage.

MT safe.

[transfer full][nullable]

gst_element_set_state ()

```
GstStateChangeReturn
gst_element_set_state (GstElement *element,
                       GstState state);
```

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 `GST_MESSAGE_ASYNC_DONE` or `GST_MESSAGE_STATE_CHANGED` on the bus.

State changes to `GST_STATE_READY` or `GST_STATE_NULL` never return GST_STATE_CHANGE_ASYNC.

Parameters

element	a GstElement to change state of.
state	the element's new GstState.

Returns

Result of the state change using GstStateChangeReturn.

MT safe.

gst_element_get_state ()

```
GstStateChangeReturn
gst_element_get_state (GstElement *element,
                       GstState *state,
                       GstState *pending,
                       GstClockTime timeout);
```

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 GST_STATE_PLAYING. 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.

Parameters

element	a GstElement to get the state of.	
state	a pointer to GstState to hold the state. Can be NULL.	[out][allow-none]
pending	a pointer to GstState to hold the pending state. Can be NULL.	[out][allow-none]
timeout	a GstClockTime to specify the timeout for an async state change or GST_CLOCK_TIME_NONE for infinite timeout.	

Returns

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 ()

```
gboolean
gst_element_set_locked_state (GstElement *element,
                              gboolean locked_state);
```

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

MT safe.

Parameters

element	a GstElement
locked_state	TRUE to lock the element's state

Returns

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 ()

```
gboolean
gst_element_is_locked_state (GstElement *element);
```

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.

Parameters

element	a GstElement.

Returns

TRUE, if the element's state is locked.

gst_element_abort_state ()

```
void
gst_element_abort_state (GstElement *element);
```

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.

Parameters

element	a GstElement to abort the state of.

gst_element_continue_state ()

```
GstStateChangeReturn
gst_element_continue_state (GstElement *element,
                            GstStateChangeReturn ret);
```

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.

Parameters

element	a GstElement to continue the state change of.
ret	The previous state return value

Returns

The result of the commit state change.

MT safe.

gst_element_lost_state ()

```
void
gst_element_lost_state (GstElement *element);
```

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. 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.

Parameters

element	a GstElement the state is lost of

gst_element_state_get_name ()

```
const gchar *
gst_element_state_get_name (GstState state);
```

Gets a string representing the given state.

Parameters

state	a GstState to get the name of.

Returns

a string with the name of the state.

[transfer none]

gst_element_state_change_return_get_name ()

```
const gchar *
gst_element_state_change_return_get_name
                                (GstStateChangeReturn state_ret);
```

Gets a string representing the given state change result.

Parameters

state_ret	a GstStateChangeReturn to get the name of.

Returns

a string with the name of the state result.

[transfer none]

gst_element_sync_state_with_parent ()

```
gboolean
gst_element_sync_state_with_parent (GstElement *element);
```

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.

Parameters

element	a GstElement.

Returns

`TRUE`, if the element's state could be synced to the parent's state.

MT safe.

gst_element_change_state ()

```
GstStateChangeReturn
gst_element_change_state (GstElement *element,
                          GstStateChange transition);
```

Perform *`transition`* on *`element`* .

This function must be called with STATE_LOCK held and is mainly used internally.

Parameters

element	a GstElement
transition	the requested transition

Returns

the GstStateChangeReturn of the state transition.

gst_element_message_full ()

```
void
gst_element_message_full (GstElement *element,
                          GstMessageType type,
                          GQuark domain,
                          gint code,
                          gchar *text,
                          gchar *debug,
                          const gchar *file,
                          const gchar *function,
                          gint line);
```

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.

Parameters

element	a GstElement to send message from	
type	the GstMessageType	
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 `NULL`.	[allow-none][transfer full]
debug	an allocated debug message to be used as a replacement for the default debugging information, or `NULL`.	[allow-none][transfer full]
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 ()

```
gboolean
gst_element_post_message (GstElement *element,
                          GstMessage *message);
```

Post a message on the element's GstBus. 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.

Parameters

element	a GstElement posting the message	
message	a GstMessage to post.	[transfer full]

Returns

`TRUE` if the message was successfully posted. The function returns `FALSE` if the element did not have a bus.

MT safe.

gst_element_query ()

```
gboolean
gst_element_query (GstElement *element,
                   GstQuery *query);
```

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.

Please note that some queries might need a running pipeline to work.

Parameters

element	a GstElement to perform the query on.	
query	the GstQuery.	[transfer none]

Returns

TRUE if the query could be performed.

MT safe.

gst_element_query_convert ()

```
gboolean
gst_element_query_convert (GstElement *element,
                           GstFormat src_format,
                           gint64 src_val,
                           GstFormat dest_format,
                           gint64 *dest_val);
```

Queries an element to convert *src_val* in *src_format* to *dest_format* .

Parameters

element	a GstElement to invoke the convert query on.	
src_format	a GstFormat to convert from.	[inout]
src_val	a value to convert.	
dest_format	the GstFormat to convert to.	
dest_val	a pointer to the result.	[out]

Returns

TRUE if the query could be performed.

gst_element_query_position ()

```
gboolean
gst_element_query_position (GstElement *element,
                            GstFormat format,
                            gint64 *cur);
```

Queries an element (usually top-level pipeline or playbin element) for the stream position in nanoseconds. This will be a value between 0 and the stream duration (if the stream duration is known). This query will usually only work once the pipeline is prerolled (i.e. reached PAUSED or PLAYING state). The application will receive an ASYNC_DONE message on the pipeline bus when that is the case.

If one repeatedly calls this function one can also create a query and reuse it in `gst_element_query()`.

Parameters

element	a GstElement to invoke the position query on.	
format	the GstFormat requested	
cur	a location in which to store the current position, or `NULL`.	[out][allow-none]

Returns

TRUE if the query could be performed.

gst_element_query_duration ()

```
gboolean
gst_element_query_duration (GstElement *element,
                            GstFormat format,
                            gint64 *duration);
```

Queries an element (usually top-level pipeline or playbin element) for the total stream duration in nanoseconds. This query will only work once the pipeline is prerolled (i.e. reached PAUSED or PLAYING state). The application will receive an ASYNC_DONE message on the pipeline bus when that is the case.

If the duration changes for some reason, you will get a DURATION_CHANGED message on the pipeline bus, in which case you should re-query the duration using this function.

Parameters

element	a GstElement to invoke the duration query on.	
format	the GstFormat requested	
duration	A location in which to store the total duration, or `NULL`.	[out][allow-none]

Returns

TRUE if the query could be performed.

gst_element_send_event ()

```
gboolean
gst_element_send_event (GstElement *element,
                        GstEvent *event);
```

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 downstream events or a random linked source pad for upstream events.

This function takes ownership of the provided event so you should `gst_event_ref()` it if you want to reuse the event after this call.

MT safe.

Parameters

element	a GstElement to send the event to.	
event	the GstEvent to send to the element.	[transfer full]

Returns

TRUE if the event was handled. Events that trigger a preroll (such as flushing seeks and steps) will emit `GST_MESSAGE_ASYNC_DONE`.

gst_element_seek_simple ()

```
gboolean
gst_element_seek_simple (GstElement *element,
                         GstFormat format,
                         GstSeekFlags seek_flags,
                         gint64 seek_pos);
```

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 TRUE on a seekable media type or FALSE 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 TRUE when it receives the event in the READY state.

Parameters

element	a GstElement to seek on

format	a GstFormat 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.

Returns

TRUE if the seek operation succeeded. Flushing seeks will trigger a preroll, which will emit GST_MESSAGE_ASYNC_DONE.

gst_element_seek ()

```
gboolean
gst_element_seek (GstElement *element,
                  gdouble rate,
                  GstFormat format,
                  GstSeekFlags flags,
                  GstSeekType start_type,
                  gint64 start,
                  GstSeekType stop_type,
                  gint64 stop);
```

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().

MT safe.

Parameters

element	a GstElement to send the event to.
rate	The new playback rate
format	The format of the seek values
flags	The optional seek flags.
start_type	The type and flags for the new start position
start	The value of the new start position
stop_type	The type and flags for the new stop position
stop	The value of the new stop position

Returns

TRUE if the event was handled. Flushing seeks will trigger a preroll, which will emit GST_MESSAGE_ASYNC_DONE.

Types and Values

struct GstElement

```
struct GstElement {
  GRecMutex             state_lock;

  /* element state */
  GCond                 state_cond;
  guint32               state_cookie;
  GstState              target_state;
  GstState              current_state;
  GstState              next_state;
  GstState              pending_state;
  GstStateChangeReturn  last_return;

  GstBus               *bus;

  /* allocated clock */
  GstClock             *clock;
  GstClockTimeDiff      base_time; /* NULL/READY: 0 - PAUSED: current time - PLAYING: difference to clock */
  GstClockTime          start_time;

  /* element pads, these lists can only be iterated while holding
   * the LOCK or checking the cookie after each LOCK. */
  guint16               numpads;
  GList                *pads;
  guint16               numsrcpads;
  GList                *srcpads;
  guint16               numsinkpads;
  GList                *sinkpads;
  guint32               pads_cookie;
};
```

GStreamer element abstract base class.

Members

GRecMutex *state_lock*;	Used to serialize execution of `gst_element_set_state()`	
GCond *state_cond*;	Used to signal completion of a state change	
guint32 *state_cookie*;	Used to detect concurrent execution of `gst_element_set_state()` and `gst_element_get_state()`	
GstState *target_state*;	the target state of an element as set by the application	
GstState *current_state*;	the current state of an element	
GstState *next_state*;	the next state of an element, can be GST_STATE_VOID_PENDING if the element is in the correct state.	
GstState *pending_state*;	the final state the element should go to, can be GST_STATE_VOID_PENDING if the element is in the correct state	
GstStateChangeReturn *last_return*;	the last return value of an element state change	
GstBus **bus*;	the bus of the element. This bus is provided to the element by the parent element or the application. A GstPipeline has a bus of its own.	
GstClock **clock*;	the clock of the element. This clock is usually provided to the element by the toplevel GstPipeline.	
GstClockTimeDiff *base_time*;	the time of the clock right before the element is set to PLAYING. Subtracting *base_time* from the current clock time in the PLAYING state will yield the running_time against the clock.	
GstClockTime *start_time*;	the running_time of the last PAUSED state	
guint16 *numpads*;	number of pads of the element, includes both source and sink pads.	
GList **pads*;	list of pads.	[element-type Gst.Pad]
guint16 *numsrcpads*;	number of source pads of the element.	
GList **srcpads*;	list of source pads.	[element-type Gst.Pad]
guint16 *numsinkpads*;	number of sink pads of the element.	
GList **sinkpads*;	list of sink pads.	[element-type Gst.Pad]
guint32 *pads_cookie*;	updated whenever the a pad is added or removed	

struct GstElementClass

```
struct GstElementClass {
  GstObjectClass         parent_class;

  /* the element metadata */
  gpointer               metadata;

  /* factory that the element was created from */
  GstElementFactory     *elementfactory;

  /* templates for our pads */
  GList                 *padtemplates;
  gint                   numpadtemplates;
  guint32                pad_templ_cookie;

  /* virtual methods for subclasses */

  /* request/release pads */
  GstPad*               (*request_new_pad)      (GstElement *element, GstPadTemplate *templ,
                                                 const gchar* name, const GstCaps *caps);
  void                  (*release_pad)          (GstElement *element, GstPad *pad);

  /* state changes */
  GstStateChangeReturn (*get_state)             (GstElement * element, GstState * state,
                                                 GstState * pending, GstClockTime timeout);
  GstStateChangeReturn (*set_state)             (GstElement *element, GstState state);
  GstStateChangeReturn (*change_state)          (GstElement *element, GstStateChange transition);
  void                 (*state_changed)         (GstElement *element, GstState oldstate,
                                                 GstState newstate, GstState pending);

  /* bus */
  void                  (*set_bus)              (GstElement * element, GstBus * bus);

  /* set/get clocks */
  GstClock*             (*provide_clock)        (GstElement *element);
  gboolean              (*set_clock)            (GstElement *element, GstClock *clock);

  /* query functions */
  gboolean              (*send_event)           (GstElement *element, GstEvent *event);

  gboolean              (*query)                (GstElement *element, GstQuery *query);

  gboolean              (*post_message)         (GstElement *element, GstMessage *message);

  void                  (*set_context)          (GstElement *element, GstContext *context);
};
```

GStreamer element class. Override the vmethods to implement the element functionality.

Members

GstObjectClass *parent_class*;	the parent class structure
gpointer *metadata*;	metadata for elements of this class
GstElementFactory **elementfactory*;	the GstElementFactory that creates these elements
GList **padtemplates*;	a GList of GstPadTemplate
gint *numpadtemplates*;	the number of padtemplates
guint32 *pad_templ_cookie*;	changed whenever the padtemplates change
request_new_pad ()	called when a new pad is requested
release_pad ()	called when a request pad is to be released
get_state ()	get the state of the element
set_state ()	set a new state on the element
change_state ()	called by *set_state* to perform an incremental state change
state_changed ()	called immediately after a new state was set.
set_bus ()	set a GstBus on the element
provide_clock ()	gets the GstClock provided by the element
set_clock ()	set the GstClock on the element
send_event ()	send a GstEvent to the element

query ()	perform a GstQuery on the element
post_message ()	called when a message is posted on the element. Chain up to the parent class' handler to have it posted on the bus.
set_context ()	set a GstContext on the element

enum GstElementFlags

The standard flags that an element may have.

Members

GST_ELEMENT_FLAG_LOCKED_STATE	ignore state changes from parent
GST_ELEMENT_FLAG_SINK	the element is a sink
GST_ELEMENT_FLAG_SOURCE	the element is a source.
GST_ELEMENT_FLAG_PROVIDE_CLOCK	the element can provide a clock
GST_ELEMENT_FLAG_REQUIRE_CLOCK	the element requires a clock
GST_ELEMENT_FLAG_INDEXABLE	the element can use an index
GST_ELEMENT_FLAG_LAST	offset to define more flags

enum GstState

The possible states an element can be in. States can be changed using `gst_element_set_state()` and checked using `gst_element_get_state()`.

Members

GST_STATE_VOID_PENDING	no pending state.
GST_STATE_NULL	the NULL state or initial state of an element.
GST_STATE_READY	the element is ready to go to PAUSED.
GST_STATE_PAUSED	the element is PAUSED, it is ready to accept and process data. Sink elements however only accept one buffer and then block.

GST_STATE_PLAYING	the element is PLAYING, the GstClock is running and the data is flowing.

enum GstStateChange

These are the different state changes an element goes through. `GST_STATE_NULL` ⇒ `GST_STATE_PLAYING` is called an upwards state change and `GST_STATE_PLAYING` ⇒ `GST_STATE_NULL` a downwards state change.

Members

GST_STATE_CHANGE_NULL_TO_READY	state change from NULL to READY. • The element must check if the resources it needs are available. Device sinks and -sources typically try to probe the device to constrain their caps. • The element opens the device (in case feature need to be probed).
GST_STATE_CHANGE_READY_TO_PAUSED	state change from READY to PAUSED. • The element pads are activated in order to receive data in PAUSED. Streaming threads are started. • Some elements might need to return `GST_STATE_CHANGE_ASYNC` and complete the state change when they have enough information. It is a requirement for sinks to return `GST_STATE_CHANGE_ASYNC` and complete the state change when they receive the first buffer or `GST_EVENT_EOS` (preroll). Sinks also block the dataflow when in PAUSED. • A pipeline resets the running_time to 0. • Live sources return `GST_STATE_CHANGE_NO_PREROLL` and don't generate data.
GST_STATE_CHANGE_PAUSED_TO_PLAYING	state change from PAUSED to PLAYING. • Most elements ignore this state change. • The pipeline selects a GstClock and distributes this to all the children before setting them to PLAYING. This means that it is only allowed to synchronize on the GstClock in the PLAYING state. • The pipeline uses the GstClock and the running_time to calculate the base_time. The base_time is distributed to all children when performing the state change.

	• Sink elements stop blocking on the preroll buffer or event and start rendering the data. • Sinks can post `GST_MESSAGE_EOS` in the PLAYING state. It is not allowed to post `GST_MESSAGE_EOS` when not in the PLAYING state. • While streaming in PAUSED or PLAYING elements can create and remove sometimes pads. • Live sources start generating data and return `GST_STATE_CHANGE_SUCCESS`.
GST_STATE_CHANGE_PLAYING_TO_PAUSED	state change from PLAYING to PAUSED. • Most elements ignore this state change. • The pipeline calculates the running_time based on the last selected GstClock and the base_time. It stores this information to continue playback when going back to the PLAYING state. • Sinks unblock any GstClock wait calls. • When a sink does not have a pending buffer to play, it returns `GST_STATE_CHANGE_ASYNC` from this state change and completes the state change when it receives a new buffer or an `GST_EVENT_EOS`. • Any queued `GST_MESSAGE_EOS` items are removed since they will be reposted when going back to the PLAYING state. The EOS messages are queued in GstBin containers. • Live sources stop generating data and return `GST_STATE_CHANGE_NO_PREROLL`.
GST_STATE_CHANGE_PAUSED_TO_READY	state change from PAUSED to READY. • Sinks unblock any waits in the preroll. • Elements unblock any waits on devices • Chain or get_range functions return `GST_FLOW_FLUSHING`. • The element pads are deactivated so that streaming becomes impossible and all streaming threads are stopped. • The sink forgets all negotiated formats • Elements remove all sometimes pads

GST_STATE_CHANGE_READY_TO_NULL	state change from READY to NULL. • Elements close devices • Elements reset any internal state.

enum GstStateChangeReturn

The possible return values from a state change function such as `gst_element_set_state()`. Only *`GST_STATE_CHANGE_FAILURE`* is a real failure.

Members

GST_STATE_CHANGE_FAILURE	the state change failed
GST_STATE_CHANGE_SUCCESS	the state change succeeded
GST_STATE_CHANGE_ASYNC	the state change will happen asynchronously
GST_STATE_CHANGE_NO_PREROLL	the state change succeeded but the element cannot produce data in `GST_STATE_PAUSED`. This typically happens with live sources.

GST_ELEMENT_METADATA_AUTHOR

```
#define GST_ELEMENT_METADATA_AUTHOR        "author"
```

Name and contact details of the author(s). Use \n to separate multiple author details. E.g: "Joe Bloggs <joe.blogs at foo.com>"

GST_ELEMENT_METADATA_DESCRIPTION

```
#define GST_ELEMENT_METADATA_DESCRIPTION   "description"
```

Sentence describing the purpose of the element. E.g: "Write stream to a file"

GST_ELEMENT_METADATA_DOC_URI

```
#define GST_ELEMENT_METADATA_DOC_URI       "doc-uri"
```

Set uri pointing to user documentation. Applications can use this to show help for e.g. effects to users.

GST_ELEMENT_METADATA_ICON_NAME

```
#define GST_ELEMENT_METADATA_ICON_NAME     "icon-name"
```

Elements that bridge to certain other products can include an icon of that used product. Application can show the icon in menus/selectors to help identifying specific elements.

GST_ELEMENT_METADATA_KLASS

```
#define GST_ELEMENT_METADATA_KLASS         "klass"
```

String describing the type of element, as an unordered list separated with slashes ('/'). See draft-klass.txt of the design docs for more details and common types. E.g: "Sink/File"

GST_ELEMENT_METADATA_LONGNAME

```
#define GST_ELEMENT_METADATA_LONGNAME      "long-name"
```

The long English name of the element. E.g. "File Sink"

Signal Details

The "no-more-pads" signal

```
void
user_function (GstElement *gstelement,
               gpointer    user_data)
```

This signals that the element will not generate more dynamic pads. Note that this signal will usually be emitted from the context of the streaming thread.

Parameters

gstelement	the object which received the signal
user_data	user data set when the signal handler was connected.

Flags: Run Last

The "pad-added" signal

```
void
user_function (GstElement *gstelement,
               GstPad     *new_pad,
               gpointer    user_data)
```

a new GstPad has been added to the element. Note that this signal will usually be emitted from the context of the streaming thread. Also keep in mind that if you add new elements to the pipeline in the signal handler you will need to set them to the desired target state with `gst_element_set_state()` or `gst_element_sync_state_with_parent()`.

Parameters

gstelement	the object which received the signal
new_pad	the pad that has been added
user_data	user data set when the signal handler was connected.

Flags: Run Last

The "pad-removed" signal

```
void
user_function (GstElement *gstelement,
               GstPad     *old_pad,
               gpointer    user_data)
```

a GstPad has been removed from the element

Parameters

gstelement	the object which received the signal
old_pad	the pad that has been removed
user_data	user data set when the signal handler was connected.

Flags: Run Last

See Also

GstElementFactory, GstPad

Generated by GTK-Doc V1.21