

GstCaps

GstCaps — Structure describing sets of media formats

Functions

```
#define GST_CAPS_REFCOUNT()
#define GST_CAPS_REFCOUNT_VALUE()
#define GST_CAPS_IS_SIMPLE()
#define GST_STATIC_CAPS()
#define GST_CAPS_FLAGS()
#define GST_CAPS_FLAG_IS_SET()
#define GST_CAPS_FLAG_SET()
#define GST_CAPS_FLAG_UNSET()
gboolean (*GstCapsForeachFunc) ()
gboolean (*GstCapsMapFunc) ()
gboolean (*GstCapsFilterMapFunc) ()
GstCaps * gst_caps_new_empty ()
GstCaps * gst_caps_new_empty_simple ()
GstCaps * gst_caps_new_any ()
GstCaps * gst_caps_new_simple ()
GstCaps * gst_caps_new_full ()
GstCaps * gst_caps_new_full_valist ()
#define gst_caps_is_writable()
GstCaps * gst_caps_copy ()
GstCaps * gst_caps_copy_nth ()
GstCaps * gst_static_caps_get ()
void gst_static_caps_cleanup ()
void gst_caps_append ()
GstCaps * gst_caps_merge ()
void gst_caps_append_structure ()
void gst_caps_append_structure_full ()
void gst_caps_remove_structure ()
GstStructure * gst_caps_steal_structure ()
GstCaps * gst_caps_merge_structure ()
GstCaps * gst_caps_merge_structure_full ()
guint gst_caps_get_size ()
GstStructure * gst_caps_get_structure ()
GstCapsFeatures * gst_caps_get_features ()
void gst_caps_set_features ()
void gst_caps_set_value ()
void gst_caps_set_simple ()
void gst_caps_set_simple_valist ()
gboolean gst_caps_foreach ()
gboolean gst_caps_map_in_place ()
void gst_caps_filter_and_map_in_place ()
gboolean gst_caps_is_any ()
```

```

gboolean gst_caps_is_empty ()
gboolean gst_caps_is_fixed ()
gboolean gst_caps_is_equal ()
gboolean gst_caps_is_equal_fixed ()
gboolean gst_caps_is_strictly_equal ()
gboolean gst_caps_is_always_compatible ()
gboolean gst_caps_is_subset ()
gboolean gst_caps_is_subset_structure ()
gboolean gst_caps_is_subset_structure_full ()
gboolean gst_caps_can_intersect ()
GstCaps * gst_caps_intersect ()
GstCaps * gst_caps_intersect_full ()
GstCaps * gst_caps_normalize ()
GstCaps * gst_caps_simplify ()
gboolean gst_caps_replace ()
gboolean gst_caps_take ()
gchar * gst_caps_to_string ()
GstCaps * gst_caps_from_string ()
GstCaps * gst_caps_subtract ()
#define gst_caps_make_writable()
GstCaps * gst_caps_truncate ()
GstCaps * gst_caps_fixate ()
GstCaps * gst_caps_ref ()
void gst_caps_unref ()

```

Types and Values

```

struct GstCaps
struct GstStaticCaps
enum GstCapsIntersectMode
enum GstCapsFlags
#define GST_CAPS_ANY
#define GST_CAPS_NONE
#define GST_STATIC_CAPS_ANY
#define GST_STATIC_CAPS_NONE

```

Object Hierarchy

```

GBoxed
└─ GstCaps

```

Includes

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

Description

Caps (capabilities) are lightweight refcounted objects describing media types. They are composed of an array of [GstStructure](#).

Caps are exposed on [GstPadTemplate](#) to describe all possible types a given pad can handle. They are also stored in the [GstRegistry](#) along with a description of the [GstElement](#).

Caps are exposed on the element pads using the [gst_pad_query_caps\(\)](#) pad function. This function describes the possible types that the pad can handle or produce at runtime.

A [GstCaps](#) can be constructed with the following code fragment:

```
1  GstCaps *caps = gst_caps_new_simple ("video/x-raw",
2    "format", G_TYPE_STRING, "I420",
3    "framerate", GST_TYPE_FRACTION, 25, 1,
4    "pixel-aspect-ratio", GST_TYPE_FRACTION, 1, 1,
5    "width", G_TYPE_INT, 320,
6    "height", G_TYPE_INT, 240,
7    NULL);
```

A [GstCaps](#) is fixed when it has no properties with ranges or lists. Use [gst_caps_is_fixed\(\)](#) to test for fixed caps. Fixed caps can be used in a caps event to notify downstream elements of the current media type.

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

Be aware that the current [GstCaps](#) / [GstStructure](#) serialization into string has limited support for nested [GstCaps](#) / [GstStructure](#) fields. It can only support one level of nesting. Using more levels will lead to unexpected behavior when using serialization features, such as [gst_caps_to_string\(\)](#) or [gst_value_serialize\(\)](#) and their counterparts.

Functions

GST_CAPS_REFCOUNT()

```
#define GST_CAPS_REFCOUNT(caps)          GST_MINI_OBJECT_REFCOUNT(caps)
```

Get access to the reference count field of the caps

Parameters

caps a [GstCaps](#)

GST_CAPS_REFCOUNT_VALUE()

```
#define GST_CAPS_REFCOUNT_VALUE(caps)    GST_MINI_OBJECT_REFCOUNT_VALUE(caps)
```

Get the reference count value of the caps.

Parameters

caps a [GstCaps](#)

GST_CAPS_IS_SIMPLE()

```
#define GST_CAPS_IS_SIMPLE(caps) (gst_caps_get_size(caps) == 1)
```

Convenience macro that checks if the number of structures in the given caps is exactly one.

Parameters

caps the [GstCaps](#) instance to check

GST_STATIC_CAPS()

```
#define                    GST_STATIC_CAPS(string)
```

Creates a new [GstCaps](#) static caps from an input string. This can be used in pad templates.

Parameters

string the string describing the caps

GST_CAPS_FLAGS()

```
#define GST_CAPS_FLAGS(caps)                    GST_MINI_OBJECT_FLAGS(caps)
```

A flags word containing [GstCapsFlags](#) flags set on this caps.

Parameters

caps a [GstCaps](#).

GST_CAPS_FLAG_IS_SET()

```
#define GST_CAPS_FLAG_IS_SET(caps,flag)                    GST_MINI_OBJECT_FLAG_IS_SET (caps, flag)
```

Gives the status of a specific flag on a caps.

Parameters

caps a [GstCaps](#).

flag the [GstCapsFlags](#) to check.

GST_CAPS_FLAG_SET()

```
#define GST_CAPS_FLAG_SET(caps,flag)                    GST_MINI_OBJECT_FLAG_SET (caps, flag)
```

Sets a caps flag on a caps.

Parameters

caps a [GstCaps](#).

flag the [GstCapsFlags](#) to set.

GST_CAPS_FLAG_UNSET()

```
#define GST_CAPS_FLAG_UNSET(caps,flag)          GST_MINI_OBJECT_FLAG_UNSET (caps, flag)
```

Clears a caps flag.

Parameters

caps a [GstCaps](#).
flag the [GstCapsFlags](#) to clear.

GstCapsForeachFunc ()

```
gboolean  
(*GstCapsForeachFunc) (GstCapsFeatures *features,  
                      GstStructure *structure,  
                      gpointer user_data);
```

A function that will be called in [gst_caps_foreach\(\)](#). The function may not modify `features` or `structure` .

Parameters

features the [GstCapsFeatures](#)
structure the [GstStructure](#)
user_data user data

Returns

[TRUE](#) if the foreach operation should continue, [FALSE](#) if the foreach operation should stop with [FALSE](#).

Since: [1.6](#)

GstCapsMapFunc ()

```
gboolean  
(*GstCapsMapFunc) (GstCapsFeatures *features,  
                    GstStructure *structure,  
                    gpointer user_data);
```

A function that will be called in [gst_caps_map_in_place\(\)](#). The function may modify `features` and `structure` .

Parameters

features the [GstCapsFeatures](#)
structure the [GstStructure](#)
user_data user data

Returns

[TRUE](#) if the map operation should continue, [FALSE](#) if the map operation should stop with [FALSE](#).

GstCapsFilterMapFunc ()

```
gboolean  
(*GstCapsFilterMapFunc) (GstCapsFeatures *features,  
                          GstStructure *structure,  
                          gpointer user_data);
```

A function that will be called in `gst_caps_filter_and_map_in_place()`. The function may modify `features` and `structure`, and both will be removed from the caps if `FALSE` is returned.

Parameters

<code>features</code>	the GstCapsFeatures
<code>structure</code>	the GstStructure
<code>user_data</code>	user data

Returns

`TRUE` if the features and structure should be preserved, `FALSE` if it should be removed.

gst_caps_new_empty ()

```
GstCaps *  
gst_caps_new_empty (void);
```

Creates a new [GstCaps](#) that is empty. That is, the returned [GstCaps](#) contains no media formats. The [GstCaps](#) is guaranteed to be writable. Caller is responsible for unrefing the returned caps.

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_new_empty_simple ()

```
GstCaps *  
gst_caps_new_empty_simple (const char *media_type);
```

Creates a new [GstCaps](#) that contains one [GstStructure](#) with name `media_type`. Caller is responsible for unrefing the returned caps.

Parameters

<code>media_type</code>	the media type of the structure
-------------------------	---------------------------------

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_new_any ()

```
GstCaps *  
gst_caps_new_any (void);
```

Creates a new [GstCaps](#) that indicates that it is compatible with any media format.

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_new_simple ()

```
GstCaps *  
gst_caps_new_simple (const char *media_type,  
                    const char *fieldname,  
                    ...);
```

Creates a new [GstCaps](#) that contains one [GstStructure](#). The structure is defined by the arguments, which have the same format as [gst_structure_new\(\)](#). Caller is responsible for unrefing the returned caps.

Parameters

media_type	the media type of the structure
fieldname	first field to set
...	additional arguments

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_new_full ()

```
GstCaps *  
gst_caps_new_full (GstStructure *struct1,  
                  ...);
```

Creates a new [GstCaps](#) and adds all the structures listed as arguments. The list must be [NULL](#)-terminated. The structures are not copied; the returned [GstCaps](#) owns the structures.

Parameters

struct1	the first structure to add
...	additional structures to add

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_new_full_valist ()

```
GstCaps *
gst_caps_new_full_valist (GstStructure *structure,
                          va_list var_args);
```

Creates a new [GstCaps](#) and adds all the structures listed as arguments. The list must be [NULL](#)-terminated. The structures are not copied; the returned [GstCaps](#) owns the structures.

Parameters

structure the first structure to add
var_args additional structures to add

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_is_writable()

```
#define            gst_caps_is_writable(caps)    gst_mini_object_is_writable (GST_MINI_OBJECT_CAST (caps))
```

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

Parameters

caps a [GstCaps](#)

gst_caps_copy ()

```
GstCaps *
gst_caps_copy (const GstCaps *caps);
```

Creates a new [GstCaps](#) as a copy of the old `caps` . The new caps will have a refcount of 1, owned by the caller. The structures are copied as well.

Note that this function is the semantic equivalent of a [gst_caps_ref\(\)](#) followed by a [gst_caps_make_writable\(\)](#). If you only want to hold on to a reference to the data, you should use [gst_caps_ref\(\)](#).

When you are finished with the caps, call [gst_caps_unref\(\)](#) on it.

Parameters

caps a [GstCaps](#).

Returns

the new [GstCaps](#)

gst_caps_copy_nth ()

```
GstCaps *  
gst_caps_copy_nth (const GstCaps *caps,  
                  guint nth);
```

Creates a new [GstCaps](#) and appends a copy of the nth structure contained in caps .

Parameters

caps the [GstCaps](#) to copy
nth the nth structure to copy

Returns

the new [GstCaps](#).

[transfer full]

gst_static_caps_get ()

```
GstCaps *  
gst_static_caps_get (GstStaticCaps *static_caps);
```

Converts a [GstStaticCaps](#) to a [GstCaps](#).

Parameters

static_caps the [GstStaticCaps](#) to convert

Returns

a pointer to the [GstCaps](#). 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.

[transfer full]

gst_static_caps_cleanup ()

```
void  
gst_static_caps_cleanup (GstStaticCaps *static_caps);
```

Clean up the cached caps contained in static_caps .

Parameters

static_caps the [GstStaticCaps](#) to clean

gst_caps_append ()

```
void  
gst_caps_append (GstCaps *caps1,
```

```
GstCaps *caps2);
```

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.

Parameters

<code>caps1</code>	the GstCaps that will be appended to	
<code>caps2</code>	the GstCaps to append.	[transfer full]

gst_caps_merge ()

```
GstCaps *  
gst_caps_merge (GstCaps *caps1,  
                GstCaps *caps2);
```

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 a writable copy of `caps1` , and then `caps2` is freed. If either caps is ANY, the resulting caps will be ANY.

Parameters

<code>caps1</code>	the GstCaps that will take the new entries.	[transfer full]
<code>caps2</code>	the GstCaps to merge in.	[transfer full]

Returns

the merged caps.

[\[transfer full\]](#)

gst_caps_append_structure ()

```
void  
gst_caps_append_structure (GstCaps *caps,  
                          GstStructure *structure);
```

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

Parameters

<code>caps</code>	the GstCaps that will be appended to	
<code>structure</code>	the GstStructure to append.	[transfer.full]

gst_caps_append_structure_full ()

```
void  
gst_caps_append_structure_full (GstCaps *caps,  
                               GstStructure *structure,  
                               GstCapsFeatures *features);
```

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

Parameters

caps	the GstCaps that will be appended to	
structure	the GstStructure to append.	[transfer full]
features	the GstCapsFeatures to append.	[transfer full][allow-none]

Since: [1.2](#)

gst_caps_remove_structure ()

```
void
gst_caps_remove_structure (GstCaps *caps,
                           guint idx);
```

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

Parameters

caps	the GstCaps to remove from
idx	Index of the structure to remove

gst_caps_steal_structure ()

```
GstStructure *
gst_caps_steal_structure (GstCaps *caps,
                          guint index);
```

Retrieves the structure with the given index from the list of structures contained in caps . The caller becomes the owner of the returned structure.

Parameters

caps	the GstCaps to retrieve from
index	Index of the structure to retrieve

Returns

a pointer to the [GstStructure](#) corresponding to index .

[\[transfer full\]](#)

gst_caps_merge_structure ()

```
GstCaps *
gst_caps_merge_structure (GstCaps *caps,
                          GstStructure *structure);
```

Appends structure to caps if its not already expressed by caps .

Parameters

caps	the GstCaps to merge into.	[transfer.full]
structure	the GstStructure to merge.	[transfer.full]

Returns

the merged caps.

[transfer.full]

gst_caps_merge_structure_full ()

```
GstCaps *  
gst_caps_merge_structure_full (GstCaps *caps,  
                              GstStructure *structure,  
                              GstCapsFeatures *features);
```

Appends structure with features to caps if its not already expressed by caps .

Parameters

caps	the GstCaps to merge into.	[transfer.full]
structure	the GstStructure to merge.	[transfer.full]
features	the GstCapsFeatures to merge.	[transfer.full][allow-none]

Returns

the merged caps.

[transfer.full]

Since: 1.2

gst_caps_get_size ()

```
guint  
gst_caps_get_size (const GstCaps *caps);
```

Gets the number of structures contained in caps .

Parameters

caps a [GstCaps](#)

Returns

the number of structures that caps contains

gst_caps_get_structure ()

```
GstStructure *
gst_caps_get_structure (const GstCaps *caps,
                       guint index);
```

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 GstCaps should not be modified. However, if you know the caps are writable, either because you have just copied them or made them writable with `gst_caps_make_writable()`, you may modify the structure returned in the usual way, e.g. with functions like `gst_structure_set()`.

You do not need to free or unref the structure returned, it belongs to the GstCaps.

Parameters

caps	a GstCaps
index	the index of the structure

Returns

a pointer to the GstStructure corresponding to index .

[transfer:none]

gst_caps_get_features ()

```
GstCapsFeatures *
gst_caps_get_features (const GstCaps *caps,
                      guint index);
```

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

WARNING: This function takes a const GstCaps *, but returns a non-const GstCapsFeatures *. This is for programming convenience -- the caller should be aware that structures inside a constant GstCaps should not be modified. However, if you know the caps are writable, either because you have just copied them or made them writable with `gst_caps_make_writable()`, you may modify the features returned in the usual way, e.g. with functions like `gst_caps_features_add()`.

You do not need to free or unref the structure returned, it belongs to the GstCaps.

Parameters

caps	a GstCaps
index	the index of the structure

Returns

a pointer to the GstCapsFeatures corresponding to index .

[transfer:none]

Since: 1.2

gst_caps_set_features ()

```
void
gst_caps_set_features (GstCaps *caps,
                      guint index,
                      GstCapsFeatures *features);
```

Sets the [GstCapsFeatures](#) features for the structure at `index` .

Parameters

<code>caps</code>	a GstCaps	
<code>index</code>	the index of the structure	
<code>features</code>	the GstCapsFeatures to set.	[allow-none][transfer:full]

Since: [1.2](#)

gst_caps_set_value ()

```
void
gst_caps_set_value (GstCaps *caps,
                   const char *field,
                   const GValue *value);
```

Sets the given `field` on all structures of `caps` to the given `value` . This is a convenience function for calling [gst_structure_set_value\(\)](#) on all structures of `caps` .

Parameters

<code>caps</code>	a writable caps
<code>field</code>	name of the field to set
<code>value</code>	value to set the field to

gst_caps_set_simple ()

```
void
gst_caps_set_simple (GstCaps *caps,
                    const char *field,
                    ...);
```

Sets fields in a [GstCaps](#). The arguments must be passed in the same manner as [gst_structure_set\(\)](#), and be `NULL`-terminated.

Parameters

<code>caps</code>	the GstCaps to set
<code>field</code>	first field to set
<code>...</code>	additional parameters

gst_caps_set_simple_valist ()

```
void
gst_caps_set_simple_valist (GstCaps *caps,
                           const char *field,
                           va_list varargs);
```

Sets fields in a [GstCaps](#). The arguments must be passed in the same manner as [gst_structure_set\(\)](#), and be `NULL`-terminated.

Parameters

caps	the GstCaps to set
field	first field to set
varargs	additional parameters

gst_caps_foreach ()

```
gboolean
gst_caps_foreach (const GstCaps *caps,
                 GstCapsForeachFunc func,
                 gpointer user_data);
```

Calls the provided function once for each structure and caps feature in the [GstCaps](#). The function must not modify the fields. Also see [gst_caps_map_in_place\(\)](#) and [gst_caps_filter_and_map_in_place\(\)](#).

Parameters

caps	a GstCaps	
func	a function to call for each field.	[scope call]
user_data	private data.	[closure]

Returns

`TRUE` if the supplied function returns `TRUE` for each call, `FALSE` otherwise.

Since: 1.6

gst_caps_map_in_place ()

```
gboolean
gst_caps_map_in_place (GstCaps *caps,
                     GstCapsMapFunc func,
                     gpointer user_data);
```

Calls the provided function once for each structure and caps feature in the [GstCaps](#). In contrast to [gst_caps_foreach\(\)](#), the function may modify but not delete the structures and features. The caps must be mutable.

Parameters

caps	a GstCaps	
func	a function to call for each field.	[scope call]
user_data	private data.	[closure]

Returns

`TRUE` if the supplied function returns `TRUE` for each call, `FALSE` otherwise.

Since: 1.6

gst_caps_filter_and_map_in_place ()

```
void
gst_caps_filter_and_map_in_place (GstCaps *caps,
                                 GstCapsFilterMapFunc func,
                                 gpointer user_data);
```

Calls the provided function once for each structure and caps feature in the `GstCaps`. In contrast to `gst_caps_foreach()`, the function may modify the structure and features. In contrast to `gst_caps_filter_and_map_in_place()`, the structure and features are removed from the caps if `FALSE` is returned from the function. The caps must be mutable.

Parameters

caps	a <code>GstCaps</code>	
func	a function to call for each field.	<code>[scope call]</code>
user_data	private data.	<code>[closure]</code>

Since: 1.6

gst_caps_is_any ()

```
gboolean
gst_caps_is_any (const GstCaps *caps);
```

Determines if caps represents any media format.

Parameters

caps the `GstCaps` to test

Returns

`TRUE` if caps represents any format.

gst_caps_is_empty ()

```
gboolean
gst_caps_is_empty (const GstCaps *caps);
```

Determines if caps represents no media formats.

Parameters

caps the `GstCaps` to test

Returns

`TRUE` if caps represents no formats.

gst_caps_is_fixed ()

```
gboolean  
gst_caps_is_fixed (const GstCaps *caps);
```

Fixed `GstCaps` 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`.

Parameters

caps the `GstCaps` to test

Returns

`TRUE` if caps is fixed

gst_caps_is_equal ()

```
gboolean  
gst_caps_is_equal (const GstCaps *caps1,  
                  const GstCaps *caps2);
```

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

Parameters

caps1 a `GstCaps`
caps2 another `GstCaps`

Returns

`TRUE` if both caps are equal.

gst_caps_is_equal_fixed ()

```
gboolean  
gst_caps_is_equal_fixed (const GstCaps *caps1,  
                        const GstCaps *caps2);
```

Tests if two `GstCaps` are equal. This function only works on fixed `GstCaps`.

Parameters

caps1 the `GstCaps` to test
caps2 the `GstCaps` to test

Returns

`TRUE` if the arguments represent the same format

gst_caps_is_strictly_equal ()

```
gboolean
gst_caps_is_strictly_equal (const GstCaps *caps1,
                           const GstCaps *caps2);
```

Checks if the given caps are exactly the same set of caps.

Parameters

caps1 a [GstCaps](#)
caps2 another [GstCaps](#)

Returns

[TRUE](#) if both caps are strictly equal.

gst_caps_is_always_compatible ()

```
gboolean
gst_caps_is_always_compatible (const GstCaps *caps1,
                               const GstCaps *caps2);
```

A given [GstCaps](#) 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 .

Parameters

caps1 the [GstCaps](#) to test
caps2 the [GstCaps](#) to test

Returns

[TRUE](#) if caps1 is a subset of caps2 .

gst_caps_is_subset ()

```
gboolean
gst_caps_is_subset (const GstCaps *subset,
                   const GstCaps *superset);
```

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

Parameters

subset a [GstCaps](#)
superset a potentially greater [GstCaps](#)

Returns

[TRUE](#) if subset is a subset of superset

gst_caps_is_subset_structure ()

```
gboolean
gst_caps_is_subset_structure (const GstCaps *caps,
                             const GstStructure *structure);
```

Checks if structure is a subset of caps . See [gst_caps_is_subset\(\)](#) for more information.

Parameters

caps a [GstCaps](#)
structure a potential [GstStructure](#) subset of caps

Returns

[TRUE](#) if structure is a subset of caps

gst_caps_is_subset_structure_full ()

```
gboolean
gst_caps_is_subset_structure_full (const GstCaps *caps,
                                   const GstStructure *structure,
                                   const GstCapsFeatures *features);
```

Checks if structure is a subset of caps . See [gst_caps_is_subset\(\)](#) for more information.

Parameters

caps a [GstCaps](#)
structure a potential [GstStructure](#) subset of caps
features a [GstCapsFeatures](#) for structure . [allow-none]

Returns

[TRUE](#) if structure is a subset of caps

Since: 1.2

gst_caps_can_intersect ()

```
gboolean
gst_caps_can_intersect (const GstCaps *caps1,
                        const GstCaps *caps2);
```

Tries intersecting caps1 and caps2 and reports whether the result would not be empty

Parameters

caps1 a [GstCaps](#) to intersect
caps2 a [GstCaps](#) to intersect

Returns

TRUE if intersection would be not empty

gst_caps_intersect ()

```
GstCaps *  
gst_caps_intersect (GstCaps *caps1,  
                   GstCaps *caps2);
```

Creates a new [GstCaps](#) that contains all the formats that are common to both `caps1` and `caps2` . Defaults to [GST_CAPS_INTERSECT_ZIG_ZAG](#) mode.

Parameters

`caps1` a [GstCaps](#) to intersect
`caps2` a [GstCaps](#) to intersect

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_intersect_full ()

```
GstCaps *  
gst_caps_intersect_full (GstCaps *caps1,  
                        GstCaps *caps2,  
                        GstCapsIntersectMode mode);
```

Creates a new [GstCaps](#) that contains all the formats that are common to both `caps1` and `caps2` , the order is defined by the [GstCapsIntersectMode](#) used.

Parameters

`caps1` a [GstCaps](#) to intersect
`caps2` a [GstCaps](#) to intersect
`mode` The intersection algorithm/mode to use

Returns

the new [GstCaps](#).

[transfer full]

gst_caps_normalize ()

```
GstCaps *  
gst_caps_normalize (GstCaps *caps);
```

Returns a [GstCaps](#) that represents the same set of formats as `caps` , but contains no lists. Each list is expanded into separate [GstStructures](#) .

This function takes ownership of `caps` and will call `gst_caps_make_writable()` on it so you must not use `caps` afterwards unless you keep an additional reference to it with `gst_caps_ref()`.

Parameters

`caps` a `GstCaps` to normalize. [transfer.full]

Returns

the normalized `GstCaps`.

[transfer.full]

gst_caps_simplify ()

```
GstCaps *  
gst_caps_simplify (GstCaps *caps);
```

Converts the given `caps` 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.

This function takes ownership of `caps` and will call `gst_caps_make_writable()` on it if necessary, so you must not use `caps` afterwards unless you keep an additional reference to it with `gst_caps_ref()`.

This method does not preserve the original order of `caps` .

Parameters

`caps` a `GstCaps` to simplify. [transfer.full]

Returns

The simplified `caps`.

[transfer.full]

gst_caps_replace ()

```
gboolean  
gst_caps_replace (GstCaps **old_caps,  
                 GstCaps *new_caps);
```

Modifies a pointer to a `GstCaps` to point to a different `GstCaps`. The modification is done atomically (so this is useful for ensuring thread safety in some cases), and the reference counts are updated appropriately (the old `caps` is unrefed, the new is refed).

Either `new_caps` or the `GstCaps` pointed to by `old_caps` may be `NULL`.

Parameters

<code>old_caps</code>	pointer to a pointer to a <code>GstCaps</code> to be replaced.	[inout][transfer.full][nullable]
<code>new_caps</code>	pointer to a <code>GstCaps</code> that will replace the caps pointed to by <code>old_caps</code> .	[transfer.none][allow-none]

Returns

`TRUE` if `new_caps` was different from `old_caps`

gst_caps_take ()

```
gboolean
gst_caps_take (GstCaps **old_caps,
               GstCaps *new_caps);
```

Modifies a pointer to a [GstCaps](#) to point to a different [GstCaps](#). This function is similar to [gst_caps_replace\(\)](#) except that it takes ownership of `new_caps` .

Parameters

<code>old_caps</code>	pointer to a pointer to a GstCaps to be replaced.	[inout] [transfer full]
<code>new_caps</code>	pointer to a GstCaps that will replace the caps pointed to by <code>old_caps</code> .	[transfer full] [allow-none]

Returns

[TRUE](#) if `new_caps` was different from `old_caps`

gst_caps_to_string ()

```
gchar *
gst_caps_to_string (const GstCaps *caps);
```

Converts `caps` to a string representation. This string representation can be converted back to a [GstCaps](#) by [gst_caps_from_string\(\)](#).

For debugging purposes its easier to do something like this:

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

This prints the caps in human readable form.

The current implementation of serialization will lead to unexpected results when there are nested [GstCaps](#) / [GstStructure](#) deeper than one level.

Parameters

`caps` a [GstCaps](#)

Returns

a newly allocated string representing `caps` .

[\[transfer full\]](#)

gst_caps_from_string ()

```
GstCaps *
gst_caps_from_string (const gchar *string);
```

Converts `caps` from a string representation.

The current implementation of serialization will lead to unexpected results when there are nested [GstCaps](#) / [GstStructure](#) deeper than one level.

Parameters

string a string to convert to [GstCaps](#)

Returns

a newly allocated [GstCaps](#).

[transfer:full]

gst_caps_subtract ()

```
GstCaps *  
gst_caps_subtract (GstCaps *minuend,  
                  GstCaps *subtrahend);
```

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.

Parameters

minuend [GstCaps](#) to subtract from

subtrahend [GstCaps](#) to subtract

Returns

the resulting caps.

[transfer:full]

gst_caps_make_writable()

```
#define            gst_caps_make_writable(caps)    GST_CAPS_CAST (gst_mini_object_make_writable (GST_MINI_OBJECT_CAST (caps)))
```

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\(\)](#).

Parameters

caps a [GstCaps](#).

[transfer:full]

Returns

a writable caps which may or may not be the same as caps .

[transfer full]

gst_caps_truncate ()

```
GstCaps *  
gst_caps_truncate (GstCaps *caps);
```

Discard all but the first structure from caps . Useful when fixating.

This function takes ownership of caps and will call `gst_caps_make_writable()` on it if necessary, so you must not use caps afterwards unless you keep an additional reference to it with `gst_caps_ref()`.

Parameters

caps the `GstCaps` to truncate. [transfer full]

Returns

truncated caps.

[transfer full]

gst_caps_fixate ()

```
GstCaps *  
gst_caps_fixate (GstCaps *caps);
```

Modifies the given caps into a representation with only fixed values. First the caps will be truncated and then the first structure will be fixated with `gst_structure_fixate()`.

This function takes ownership of caps and will call `gst_caps_make_writable()` on it so you must not use caps afterwards unless you keep an additional reference to it with `gst_caps_ref()`.

Parameters

caps a `GstCaps` to fixate. [transfer full]

Returns

the fixated caps.

[transfer full]

gst_caps_ref ()

```
GstCaps *  
gst_caps_ref (GstCaps *caps);
```

Add a reference to a `GstCaps` object.

From this point on, until the caller calls `gst_caps_unref()` or `gst_caps_make_writable()`, it is guaranteed that the caps object will not change. This means its structures won't change, etc. To use a `GstCaps` object, you must always have a refcount on it -- either the one made implicitly by e.g. `gst_caps_new_simple()`, or via taking one explicitly with this function.

Parameters

`caps` the [GstCaps](#) to reference

Returns

the same [GstCaps](#) object.

gst_caps_unref ()

```
void
gst_caps_unref (GstCaps *caps);
```

Unref a [GstCaps](#) and free all its structures and the structures' values when the refcount reaches 0.

Parameters

`caps` a [GstCaps](#).

Types and Values

struct GstCaps

```
struct GstCaps {
    GstMiniObject mini_object;
};
```

Object describing media types.

Members

[GstMiniObject](#) `mini_object`; the parent type

struct GstStaticCaps

```
struct GstStaticCaps {
    GstCaps *caps;
    const char *string;
};
```

Datastructure to initialize [GstCaps](#) from a string description usually used in conjunction with [GST_STATIC_CAPS\(\)](#) and [gst_static_caps_get\(\)](#) to instantiate a [GstCaps](#).

Members

[GstCaps](#) `*caps`; the cached [GstCaps](#)
`const char *string`; a string describing a caps

enum GstCapsIntersectMode

Modes of caps intersection

GST_CAPS_INTERSECT_ZIG_ZAG tries to preserve overall order of both caps by iterating on the caps' structures as the following matrix shows:

```

1  caps1
2
3  | 1  2  4  7
4  caps2 | 3  5  8 10
5  | 6  9 11 12

```

Used when there is no explicit precedence of one caps over the other. e.g. tee's sink pad getcaps function, it will probe its src pad peers' for their caps and intersect them with this mode.

`GST_CAPS_INTERSECT_FIRST` is useful when an element wants to preserve another element's caps priority order when intersecting with its own caps. Example: If caps1 is [A, B, C] and caps2 is [E, B, D, A], the result would be [A, B], maintaining the first caps priority on the intersection.

Members

GST_CAPS_INTERSECT_ZIG_ZAG	Zig-zags over both caps.
GST_CAPS_INTERSECT_FIRST	Keeps the first caps order.

enum GstCapsFlags

Extra flags for a caps.

Members

GST_CAPS_FLAG_ANY	Caps has no specific content, but can contain anything.
-------------------	---

GST CAPS ANY

```
#define GST_CAPS_ANY _gst_caps_any
```

Means that the element/pad can output 'anything'. Useful for elements that output unknown media, such as `filesrc`. This macro returns a singleton and should not be unrefed.

GST CAPS NONE

```
#define GST_CAPS_NONE          _gst_caps_none
```

The opposite of `GST_CAPS_ANY`: it means that the pad/element outputs an undefined media type that can not be detected. This macro returns a singleton and should not be unreffed.

GST_STATIC_CAPS_ANY

```
#define GST_STATIC CAPS ANY          GST_STATIC CAPS("ANY")
```

Creates a new `GstCaps` static caps that matches anything. This can be used in pad templates.

GST_STATIC_CAPS_NONE

```
#define GST_STATIC_CAPS_NONE      GST_STATIC_CAPS("NONE")
```

Creates a new [GstCaps](#) static caps that matches nothing. This can be used in pad templates.

See Also

[GstStructure](#), [GstMiniObject](#)

Generated by GTK-Doc V1.24