

OptiX Utility Library

3.0.0

Generated by Doxygen 1.7.6.1

Wed Nov 21 2012 12:59:03

Contents

1	Module Documentation	1
1.1	rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU.	1
1.1.1	Detailed Description	1
1.1.2	Typedef Documentation	2
1.1.3	Enumeration Type Documentation	2
1.1.4	Function Documentation	4
1.2	OptiXpp: C++ wrapper for the OptiX C API.	9
1.2.1	Detailed Description	9
1.2.2	Typedef Documentation	20
1.2.3	Function Documentation	22
2	Class Documentation	61
2.1	optix::AccelerationObj Class Reference	61
2.1.1	Detailed Description	61
2.1.2	Member Function Documentation	62
2.1.3	Friends And Related Function Documentation	63
2.2	optix::APIObj Class Reference	64
2.2.1	Detailed Description	65
2.2.2	Constructor & Destructor Documentation	65
2.2.3	Member Function Documentation	65
2.3	optix::BufferObj Class Reference	66
2.3.1	Detailed Description	68
2.3.2	Member Function Documentation	68
2.3.3	Friends And Related Function Documentation	71
2.4	optix::ContextObj Class Reference	71
2.4.1	Detailed Description	74
2.4.2	Member Function Documentation	74
2.4.3	Friends And Related Function Documentation	84
2.5	optix::DestroyableObj Class Reference	84
2.5.1	Detailed Description	85
2.5.2	Constructor & Destructor Documentation	86
2.5.3	Member Function Documentation	86

2.6	optix::Exception Class Reference	86
2.6.1	Detailed Description	87
2.6.2	Constructor & Destructor Documentation	87
2.6.3	Member Function Documentation	87
2.7	optix::GeometryGroupObj Class Reference	88
2.7.1	Detailed Description	89
2.7.2	Member Function Documentation	89
2.7.3	Friends And Related Function Documentation	90
2.8	optix::GeometryInstanceObj Class Reference	90
2.8.1	Detailed Description	91
2.8.2	Member Function Documentation	92
2.8.3	Friends And Related Function Documentation	94
2.9	optix::GeometryObj Class Reference	94
2.9.1	Detailed Description	95
2.9.2	Member Function Documentation	95
2.9.3	Friends And Related Function Documentation	98
2.10	optix::GroupObj Class Reference	98
2.10.1	Detailed Description	99
2.10.2	Member Function Documentation	99
2.10.3	Friends And Related Function Documentation	100
2.11	optix::Handle< T > Class Template Reference	100
2.11.1	Detailed Description	101
2.11.2	Constructor & Destructor Documentation	101
2.11.3	Member Function Documentation	102
2.12	optix::MaterialObj Class Reference	104
2.12.1	Detailed Description	105
2.12.2	Member Function Documentation	105
2.12.3	Friends And Related Function Documentation	107
2.13	optix::ProgramObj Class Reference	107
2.13.1	Detailed Description	108
2.13.2	Member Function Documentation	108
2.13.3	Friends And Related Function Documentation	110
2.14	RTUtraversalresult Struct Reference	110
2.14.1	Detailed Description	110

2.14.2	Member Data Documentation	110
2.15	optix::ScopedObj Class Reference	111
2.15.1	Detailed Description	111
2.15.2	Constructor & Destructor Documentation	111
2.15.3	Member Function Documentation	112
2.16	optix::SelectorObj Class Reference	112
2.16.1	Detailed Description	113
2.16.2	Member Function Documentation	113
2.16.3	Friends And Related Function Documentation	115
2.17	optix::TextureSamplerObj Class Reference	115
2.17.1	Detailed Description	117
2.17.2	Member Function Documentation	117
2.17.3	Friends And Related Function Documentation	120
2.18	optix::TransformObj Class Reference	120
2.18.1	Detailed Description	121
2.18.2	Member Function Documentation	121
2.18.3	Friends And Related Function Documentation	122
2.19	optix::VariableObj Class Reference	122
2.19.1	Detailed Description	125
2.19.2	Member Function Documentation	125
2.19.3	Friends And Related Function Documentation	131
3	File Documentation	131
3.1	optixpp_namespace.h File Reference	131
3.1.1	Detailed Description	133
3.2	optixpp_namespace.h	133
3.3	optixu.h File Reference	177
3.3.1	Define Documentation	178
3.3.2	Function Documentation	179
3.4	optixu.h	182
3.5	optixu_traversal.h File Reference	189
3.5.1	Detailed Description	190
3.5.2	Typedef Documentation	190
3.5.3	Enumeration Type Documentation	190

3.5.4	Function Documentation	192
3.6	optixu_traversal.h	196

1 Module Documentation

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU.

1.1.1 Detailed Description

The OptiX traversal API is demonstrated in the traversal sample within the OptiX SDK.

Files

- file [optixu_traversal.h](#)

Typedefs

- typedef struct RTUtraversal_api * [RTUtraversal](#)

Classes

- struct [RTUtraversalresult](#)
Structure encapsulating the result of a single ray query.

Enumerations

- enum [RTUquerytype](#) { [RTU_QUERY_TYPE_ANY_HIT](#) = 0, [RTU_QUERY_TYPE_CLOSEST_HIT](#), [RTU_QUERY_TYPE_COUNT](#) }
- enum [RTUrayformat](#) { [RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED](#) = 0, [RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED](#), [RTU_RAYFORMAT_COUNT](#) }
- enum [RTUtriformat](#) { [RTU_TRIFORMAT_MESH](#) = 0, [RTU_TRIFORMAT_TRIANGLE_SOUP](#), [RTU_TRIFORMAT_COUNT](#) }
- enum [RTUinitoptions](#) { [RTU_INITOPTION_NONE](#) = 0, [RTU_INITOPTION_GPU_ONLY](#) = 1 << 0, [RTU_INITOPTION_CPU_ONLY](#) = 1 << 1, [RTU_INITOPTION_CULL_BACKFACE](#) = 1 << 2 }
- enum [RTUoutput](#) { [RTU_OUTPUT_NONE](#) = 0, [RTU_OUTPUT_NORMAL](#) = 1 << 0, [RTU_OUTPUT_BARYCENTRIC](#) = 1 << 1, [RTU_OUTPUT_BACKFACING](#) = 1 << 2 }
- enum [RTUoption](#) { [RTU_OPTION_INT_NUM_THREADS](#) = 0 }

Functions

- RTresult RTAPI [rtuTraversalCreate](#) (RTUtraversal *traversal, RTUquerytype query_type, RTUrayformat ray_format, RTUtriformat tri_format, unsigned int outputs, unsigned int options, RTcontext context)
- RTresult RTAPI [rtuTraversalGetErrorString](#) (RTUtraversal traversal, RTresult code, const char **return_string)
- RTresult RTAPI [rtuTraversalSetOption](#) (RTUtraversal traversal, RTUoption option, void *value)
- RTresult RTAPI [rtuTraversalSetMesh](#) (RTUtraversal traversal, unsigned int num_verts, const float *verts, unsigned int num_tris, const unsigned *indices)
- RTresult RTAPI [rtuTraversalSetTriangles](#) (RTUtraversal traversal, unsigned int num_tris, const float *tris)
- RTresult RTAPI [rtuTraversalSetAccelData](#) (RTUtraversal traversal, const void *data, RTsize data_size)
- RTresult RTAPI [rtuTraversalGetAccelDataSize](#) (RTUtraversal traversal, RTsize *data_size)
- RTresult RTAPI [rtuTraversalGetAccelData](#) (RTUtraversal traversal, void *data)
- RTresult RTAPI [rtuTraversalMapRays](#) (RTUtraversal traversal, unsigned int num_rays, float **rays)
- RTresult RTAPI [rtuTraversalUnmapRays](#) (RTUtraversal traversal)
- RTresult RTAPI [rtuTraversalPreprocess](#) (RTUtraversal traversal)
- RTresult RTAPI [rtuTraversalTraverse](#) (RTUtraversal traversal)
- RTresult RTAPI [rtuTraversalMapResults](#) (RTUtraversal traversal, RTUtraversalresult **results)
- RTresult RTAPI [rtuTraversalUnmapResults](#) (RTUtraversal traversal)
- RTresult RTAPI [rtuTraversalMapOutput](#) (RTUtraversal traversal, RTUoutput which, void **output)
- RTresult RTAPI [rtuTraversalUnmapOutput](#) (RTUtraversal traversal, RTUoutput which)
- RTresult RTAPI [rtuTraversalDestroy](#) (RTUtraversal traversal)

1.1.2 Typedef Documentation

1.1.2.1 typedef struct RTUtraversal_api* RTUtraversal

Opaque type. Note that the *_api types should never be used directly. Only the typedef target names will be guaranteed to remain unchanged.

Definition at line 116 of file [optixu_traversal.h](#).

1.1.3 Enumeration Type Documentation

1.1.3.1 enum RTUinitoptions

Initialization options (static across life of traversal object).

The rtuTraverse API supports both running on the CPU and GPU. When RTU_INIT_OPTION_NONE is specified GPU context creation is attempted. If that fails (such as

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 3

when there isn't an NVIDIA GPU part present, the CPU code path is automatically chosen. Specifying `RTU_INITOPTION_GPU_ONLY` or `RTU_INITOPTION_CPU_ONLY` will only use the GPU or CPU modes without automatic transitions from one to the other.

`RTU_INITOPTION_CULL_BACKFACE` will enable back face culling during intersection.

Enumerator:

```
RTU_INITOPTION_NONE  
RTU_INITOPTION_GPU_ONLY  
RTU_INITOPTION_CPU_ONLY  
RTU_INITOPTION_CULL_BACKFACE
```

Definition at line 89 of file [optixu_traversal.h](#).

1.1.3.2 enum RTUoption

Runtime options (can be set multiple times for a given traversal object).

Enumerator:

```
RTU_OPTION_INT_NUM_THREADS
```

Definition at line 107 of file [optixu_traversal.h](#).

1.1.3.3 enum RTUoutput

Enumerator:

```
RTU_OUTPUT_NONE  
RTU_OUTPUT_NORMAL  
RTU_OUTPUT_BARYCENTRIC  
RTU_OUTPUT_BACKFACING
```

Definition at line 96 of file [optixu_traversal.h](#).

1.1.3.4 enum RTUquerytype

The type of ray query to be performed.

See OptiX Programming Guide for explanation of any vs. closest hit queries. Note that in the case of `RTU_QUERY_TYPE_ANY_HIT`, the `prim_id` and `t` intersection values in [RTUtraversalresult](#) will correspond to the first successful intersection. These values may not be indicative of the closest intersection, only that there was at least one.

Enumerator:

```
RTU_QUERY_TYPE_ANY_HIT Perform any hit calculation  
RTU_QUERY_TYPE_CLOSEST_HIT Perform closest hit calculation  
RTU_QUERY_TYPE_COUNT
```

Definition at line 49 of file [optixu_traversal.h](#).

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 4

1.1.3.5 enum RTUrayformat

The input format of the ray vector.

Enumerator:

RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED
RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED
RTU_RAYFORMAT_COUNT

Definition at line 58 of file [optixu_traversal.h](#).

1.1.3.6 enum RTUtriformat

The input format of the triangles.

TRIANGLE_SOUP implies future use of `rtuTraversalSetTriangles` while MESH implies use of `rtuTraversalSetMesh`.

Enumerator:

RTU_TRIFORMAT_MESH
RTU_TRIFORMAT_TRIANGLE_SOUP
RTU_TRIFORMAT_COUNT

Definition at line 70 of file [optixu_traversal.h](#).

1.1.4 Function Documentation

1.1.4.1 **RTresult RTAPI `rtuTraversalCreate` (`RTUtraversal * traversal`, `RTUquerytype query_type`, `RTUrayformat ray_format`, `RTUtriformat tri_format`, unsigned int `outputs`, unsigned int `options`, `RTcontext context`)**

Create a traversal state and associate a context with it. If context is a null pointer a new context will be created internally. The context should also not be used for any other launch commands from the OptiX host API, nor attached to multiple `RTUtraversal` objects at one time.

Parameters

<code>out</code>	<code>traversal</code>	Return pointer for traverse state handle
	<code>query_type</code>	Ray query type
	<code>ray_format</code>	Ray format
	<code>tri_format</code>	Triangle format
	<code>outputs</code>	OR'ed mask of requested RTUoutputs
	<code>options</code>	Bit vector of or'ed RTUinitoptions.
	<code>context</code>	RTcontext used for internal object creation

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 5

1.1.4.2 RResult RTAPI rtuTraversalDestroy (RTUtraversal *traversal*)

Clean up any internal memory associated with rtuTraversal operations. Includes destruction of result buffers returned via rtuTraversalGetResults. Invalidates traversal object.

Parameters

	<i>traversal</i>	Traversal state handle
--	------------------	------------------------

1.1.4.3 RResult RTAPI rtuTraversalGetAccelData (RTUtraversal *traversal*, void * *data*)

Retrieve acceleration data for current geometry. Will force acceleration build if necessary. The data parameter should be preallocated and its length should match return value of rtuTraversalGetAccelDataSize.

Parameters

	<i>traversal</i>	Traversal state handle
out	<i>data</i>	Acceleration data

1.1.4.4 RResult RTAPI rtuTraversalGetAccelDataSize (RTUtraversal *traversal*, RTsize * *data_size*)

Retrieve acceleration data size for current geometry. Will force acceleration build if necessary.

Parameters

	<i>traversal</i>	Traversal state handle
out	<i>data_size</i>	Size of acceleration data

1.1.4.5 RResult RTAPI rtuTraversalGetErrorString (RTUtraversal *traversal*, RResult *code*, const char ** *return_string*)

Returns the string associated with the error code and any additional information from the last error. If traversal is non-NULL return_string only remains valid while traversal is live.

Parameters

	<i>traversal</i>	Traversal state handle. Can be NULL.
	<i>code</i>	Error code from last error
out	<i>return_string</i>	Pointer to string with error message in it.

1.1.4.6 RResult RTAPI rtuTraversalMapOutput (RTUtraversal *traversal*, RTUoutput *which*, void ** *output*)

Retrieve user-specified output from last rtuTraversal call. Output can be copied from the pointer returned by rtuTraversalMapOutput and will have length 'num_rays' from as

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 6

prescribed from the previous call to `rtuTraversalSetRays`. For each `RTUoutput`, a single `rtuTraversalMapOutput` pointers can be outstanding. `rtuTraversalUnmapOutput` should be called when finished reading the output.

If requested output type was not turned on with a previous call to `rtuTraverseSetOutputs` an error will be returned. See `RTUoutput` enum for description of output data formats for various outputs.

Parameters

	<i>traversal</i>	Traversal state handle
	<i>which</i>	Output type to be specified
<i>out</i>	<i>output</i>	Pointer to output from last traverse

1.1.4.7 RTResult RTAPI `rtuTraversalMapRays` (`RTUtraversal traversal`, unsigned int `num_rays`, float ** `rays`)

Specify set of rays to be cast upon next call to `rtuTraversalTraverse`. `rtuTraversalMapRays` obtains a pointer which can be used to copy the ray data into. Rays should be packed in the format described in `rtuTraversalCreate` call. When copying is completed `rtuTraversalUnmapRays` should be called. Note that this call invalidates any existing results buffers until `rtuTraversalTraverse` is called again.

Parameters

	<i>traversal</i>	Traversal state handle
	<i>num_rays</i>	Number of rays to be traced
	<i>rays</i>	Pointer to ray data

1.1.4.8 RTResult RTAPI `rtuTraversalMapResults` (`RTUtraversal traversal`, `RTUtraversalresult ** results`)

Retrieve results of last `rtuTraversal` call. Results can be copied from the pointer returned by `rtuTraversalMapResults` and will have length '`num_rays`' as prescribed from the previous call to `rtuTraversalMapRays`. `rtuTraversalUnmapResults` should be called when finished reading the results. Returned primitive ID of -1 indicates a ray miss.

Parameters

	<i>traversal</i>	Traversal state handle
<i>out</i>	<i>results</i>	Pointer to results of last traverse

1.1.4.9 RTResult RTAPI `rtuTraversalPreprocess` (`RTUtraversal traversal`)

Perform any necessary preprocessing (eg, acceleration structure building, optix context compilation). It is not necessary to call this function as `rtuTraversalTraverse` will call this internally as necessary.

1.1 rtTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 7

Parameters

<i>traversal</i>	Traversal state handle
------------------	------------------------

1.1.4.10 RTresult RTAPI rtTraversalSetAccelData (RTUtraversal *traversal*, const void * *data*, RTsize *data_size*)

Specify acceleration data for current geometry. Input acceleration data should be result of rtTraversalGetAccelData or rtAccelerationGetData call.

Parameters

<i>traversal</i>	Traversal state handle
<i>data</i>	Acceleration data
<i>data_size</i>	Size of acceleration data

1.1.4.11 RTresult RTAPI rtTraversalSetMesh (RTUtraversal *traversal*, unsigned int *num_verts*, const float * *verts*, unsigned int *num_tris*, const unsigned * *indices*)

Specify triangle mesh to be intersected by the next call to rtTraversalLaunch. Only one geometry set may be active at a time. Subsequent calls to rtTraversalSetTriangles or rtTraversalSetMesh will override any previously specified geometry. No internal copies of the mesh data are made. The user should ensure that the mesh data remains valid until after rtTraversalTraverse has been called. Counter-clockwise winding is assumed for normal and backfacing computations.

Parameters

<i>traversal</i>	Traversal state handle
<i>num_verts</i>	Vertex count
<i>verts</i>	Vertices [v1_x, v1_y, v1_z, v2.x, ...]
<i>num_tris</i>	Triangle count
<i>indices</i>	Indices [tri1_index1, tr1_index2, ...]

1.1.4.12 RTresult RTAPI rtTraversalSetOption (RTUtraversal *traversal*, RTUoption *option*, void * *value*)

Set a runtime option. Unlike initialization options, these options may be set more than once for a given RTUtraversal instance.

Parameters

<i>traversal</i>	Traversal state handle
<i>option</i>	The option to be set
<i>value</i>	Value of the option

1.1 rtuTraversal: traversal API allowing batch raycasting queries utilizing either OptiX or the CPU. 8

1.1.4.13 RTresult RTAPI rtuTraversalSetTriangles (RTUtraversal *traversal*, unsigned int *num_tris*, const float * *tris*)

Specify triangle soup to be intersected by the next call to rtuTraversalLaunch. Only one geometry set may be active at a time. Subsequent calls to rtuTraversalSetTriangles or rtuTraversalSetMesh will override any previously specified geometry. No internal copies of the triangle data are made. The user should ensure that the triangle data remains valid until after rtuTraversalTraverse has been called. Counter-clockwise winding is assumed for normal and backfacing computations.

Parameters

<i>traversal</i>	Traversal state handle
<i>num_tris</i>	Triangle count
<i>tris</i>	Triangles [tri1_v1.x, tri1_v1.y, tri1_v1.z, tri1_v2.x, ...]

1.1.4.14 RTresult RTAPI rtuTraversalTraverse (RTUtraversal *traversal*)

Perform any necessary preprocessing (eg, acceleration structure building and kernel compilation) and cast current rays against current geometry.

Parameters

<i>traversal</i>	Traversal state handle
------------------	------------------------

1.1.4.15 RTresult RTAPI rtuTraversalUnmapOutput (RTUtraversal *traversal*, RTUOutput *which*)

See rtuTraversalMapOutput

1.1.4.16 RTresult RTAPI rtuTraversalUnmapRays (RTUtraversal *traversal*)

See rtuTraversalMapRays.

1.1.4.17 RTresult RTAPI rtuTraversalUnmapResults (RTUtraversal *traversal*)

See rtuTraversalMapResults

1.2 OptiXpp: C++ wrapper for the OptiX C API.

1.2.1 Detailed Description

OptiXpp wraps each OptiX C API opaque type in a C++ class. Most of the OptiXpp class member functions map directly to C API function calls:

- [VariableObj::getContext](#) -> rtVariableGetContext
- [ContextObj::createBuffer](#) -> rtBufferCreate

Many classes have convenience functions which encapsulate a related group of OptiX functions. For instance

```
ContextObj::createBuffer(unsigned int type, RTformat format, RTsize width)
```

provides the functionality of

- rtBufferCreate
- rtBufferSetFormat
- rtBufferSetSize1D

in a single call.

Manipulation of these classes is performed via reference counted [Handle](#) class. Rather than working with a [ContextObj](#) directly you would use a [Context](#) instead, which is simply a typedef for `Handle<ContextObj>`. The OptiX SDK has many examples of the use of OptiXpp. In particular, `sample5` and `sample5pp` are a good place to look when learning OptiXpp as they are nearly identical programs, one created with the C API and one with the C++ API.

Files

- file [optixpp_namespace.h](#)

Typedefs

- typedef `Handle< AccelerationObj >` [optix::Acceleration](#)
- typedef `Handle< BufferObj >` [optix::Buffer](#)
- typedef `Handle< ContextObj >` [optix::Context](#)
- typedef `Handle< GeometryObj >` [optix::Geometry](#)
- typedef `Handle< GeometryGroupObj >` [optix::GeometryGroup](#)
- typedef `Handle< GeometryInstanceObj >` [optix::GeometryInstance](#)
- typedef `Handle< GroupObj >` [optix::Group](#)
- typedef `Handle< MaterialObj >` [optix::Material](#)
- typedef `Handle< ProgramObj >` [optix::Program](#)
- typedef `Handle< SelectorObj >` [optix::Selector](#)
- typedef `Handle< TextureSamplerObj >` [optix::TextureSampler](#)
- typedef `Handle< TransformObj >` [optix::Transform](#)
- typedef `Handle< VariableObj >` [optix::Variable](#)

Classes

- class `optix::Handle< T >`
The `Handle` class is a reference counted handle class used to manipulate API objects.
- class `optix::Exception`
`Exception` class for error reporting from the OptiXpp API.
- class `optix::APIObj`
Base class for all reference counted wrappers around OptiX C API opaque types.
- class `optix::DestroyableObj`
Base class for all wrapper objects which can be destroyed and validated.
- class `optix::ScopedObj`
Base class for all objects which are OptiX variable containers.
- class `optix::VariableObj`
Variable object wraps OptiX C API RTvariable type and its related function set.
- class `optix::ContextObj`
Context object wraps the OptiX C API RTcontext opaque type and its associated function set.
- class `optix::ProgramObj`
Program object wraps the OptiX C API RTprogram opaque type and its associated function set.
- class `optix::GroupObj`
Group wraps the OptiX C API RTgroup opaque type and its associated function set.
- class `optix::GeometryGroupObj`
GeometryGroup wraps the OptiX C API RTgeometrygroup opaque type and its associated function set.
- class `optix::TransformObj`
Transform wraps the OptiX C API RTtransform opaque type and its associated function set.
- class `optix::SelectorObj`
Selector wraps the OptiX C API RTselector opaque type and its associated function set.
- class `optix::AccelerationObj`
Acceleration wraps the OptiX C API RTacceleration opaque type and its associated function set.
- class `optix::GeometryInstanceObj`
GeometryInstance wraps the OptiX C API RTgeometryinstance acceleration opaque type and its associated function set.
- class `optix::GeometryObj`
Geometry wraps the OptiX C API RTgeometry opaque type and its associated function set.
- class `optix::MaterialObj`
Material wraps the OptiX C API RTmaterial opaque type and its associated function set.
- class `optix::TextureSamplerObj`
TextureSampler wraps the OptiX C API RTtexturesampler opaque type and its associated function set.

- class `optix::BufferObj`

Buffer wraps the OptiX C API RTbuffer opaque type and its associated function set.

Functions

- static Exception `optix::Exception::makeException` (RTresult code, RTcontext context)
- static Exception `optix::APIObj::makeException` (RTresult code, RTcontext context)
- `Handle< VariableObj > optix::Handle< T >::operator[]` (const std::string &varname)
- `Handle< VariableObj > optix::Handle< T >::operator[]` (const char *varname)
- virtual void `optix::APIObj::checkError` (RTresult code) const
- virtual void `optix::APIObj::checkError` (RTresult code, Context context) const
- void `optix::APIObj::checkErrorNoGetContext` (RTresult code) const
- Context `optix::ContextObj::getContext` () const
- static unsigned int `optix::ContextObj::getDeviceCount` ()
- static std::string `optix::ContextObj::getDeviceName` (int ordinal)
- static void `optix::ContextObj::getDeviceAttribute` (int ordinal, RTdeviceattribute attrib, RTsize size, void *p)
- static Context `optix::ContextObj::create` ()
- void `optix::ContextObj::destroy` ()
- void `optix::ContextObj::validate` ()
- void `optix::ContextObj::compile` ()
- int `optix::ContextObj::getRunningState` () const
- RTcontext `optix::ContextObj::get` ()
- void `optix::ProgramObj::destroy` ()
- void `optix::ProgramObj::validate` ()
- Context `optix::ProgramObj::getContext` () const
- Variable `optix::ProgramObj::declareVariable` (const std::string &name)
- Variable `optix::ProgramObj::queryVariable` (const std::string &name) const
- void `optix::ProgramObj::removeVariable` (Variable v)
- unsigned int `optix::ProgramObj::getVariableCount` () const
- Variable `optix::ProgramObj::getVariable` (unsigned int index) const
- RTprogram `optix::ProgramObj::get` ()
- void `optix::GroupObj::destroy` ()
- void `optix::GroupObj::validate` ()
- Context `optix::GroupObj::getContext` () const
- void `optix::SelectorObj::destroy` ()
- void `optix::SelectorObj::validate` ()
- Context `optix::SelectorObj::getContext` () const
- RTselector `optix::SelectorObj::get` ()
- RTgroup `optix::GroupObj::get` ()
- void `optix::GeometryGroupObj::destroy` ()
- void `optix::GeometryGroupObj::validate` ()
- Context `optix::GeometryGroupObj::getContext` () const

- RTgeometrygroup [optix::GeometryGroupObj::get](#) ()
- void [optix::TransformObj::destroy](#) ()
- void [optix::TransformObj::validate](#) ()
- Context [optix::TransformObj::getContext](#) () const
- RTtransform [optix::TransformObj::get](#) ()
- void [optix::AccelerationObj::destroy](#) ()
- void [optix::AccelerationObj::validate](#) ()
- Context [optix::AccelerationObj::getContext](#) () const
- RTacceleration [optix::AccelerationObj::get](#) ()
- void [optix::GeometryInstanceObj::destroy](#) ()
- void [optix::GeometryInstanceObj::validate](#) ()
- Context [optix::GeometryInstanceObj::getContext](#) () const
- RTgeometryinstance [optix::GeometryInstanceObj::get](#) ()
- void [optix::GeometryObj::destroy](#) ()
- void [optix::GeometryObj::validate](#) ()
- Context [optix::GeometryObj::getContext](#) () const
- RTgeometry [optix::GeometryObj::get](#) ()
- void [optix::MaterialObj::destroy](#) ()
- void [optix::MaterialObj::validate](#) ()
- Context [optix::MaterialObj::getContext](#) () const
- RTmaterial [optix::MaterialObj::get](#) ()
- void [optix::TextureSamplerObj::destroy](#) ()
- void [optix::TextureSamplerObj::validate](#) ()
- Context [optix::TextureSamplerObj::getContext](#) () const
- RTtexturesampler [optix::TextureSamplerObj::get](#) ()
- void [optix::BufferObj::destroy](#) ()
- void [optix::BufferObj::validate](#) ()
- Context [optix::BufferObj::getContext](#) () const
- RTbuffer [optix::BufferObj::get](#) ()
- Context [optix::VariableObj::getContext](#) () const
- std::string [optix::VariableObj::getName](#) () const
- std::string [optix::VariableObj::getAnnotation](#) () const
- RTobjecttype [optix::VariableObj::getType](#) () const
- RTvariable [optix::VariableObj::get](#) ()
- RTsize [optix::VariableObj::getSize](#) () const

Float setters

Set variable to have a float value.

- void [optix::VariableObj::setFloat](#) (float f1)
- void [optix::VariableObj::setFloat](#) (optix::float2 f)
- void [optix::VariableObj::setFloat](#) (float f1, float f2)
- void [optix::VariableObj::setFloat](#) (optix::float3 f)
- void [optix::VariableObj::setFloat](#) (float f1, float f2, float f3)
- void [optix::VariableObj::setFloat](#) (optix::float4 f)

- void `optix::VariableObj::setFloat` (float f1, float f2, float f3, float f4)
- void `optix::VariableObj::set1fv` (const float *f)
- void `optix::VariableObj::set2fv` (const float *f)
- void `optix::VariableObj::set3fv` (const float *f)
- void `optix::VariableObj::set4fv` (const float *f)

Int setters

Set variable to have an int value.

- void `optix::VariableObj::setInt` (int i1)
- void `optix::VariableObj::setInt` (optix::int2 i)
- void `optix::VariableObj::setInt` (int i1, int i2)
- void `optix::VariableObj::setInt` (optix::int3 i)
- void `optix::VariableObj::setInt` (int i1, int i2, int i3)
- void `optix::VariableObj::setInt` (optix::int4 i)
- void `optix::VariableObj::setInt` (int i1, int i2, int i3, int i4)
- void `optix::VariableObj::set1iv` (const int *i)
- void `optix::VariableObj::set2iv` (const int *i)
- void `optix::VariableObj::set3iv` (const int *i)
- void `optix::VariableObj::set4iv` (const int *i)

Unsigned int setters

Set variable to have an unsigned int value.

- void `optix::VariableObj::setUInt` (unsigned int u1)
- void `optix::VariableObj::setUInt` (unsigned int u1, unsigned int u2)
- void `optix::VariableObj::setUInt` (unsigned int u1, unsigned int u2, unsigned int u3)
- void `optix::VariableObj::setUInt` (unsigned int u1, unsigned int u2, unsigned int u3, unsigned int u4)
- void `optix::VariableObj::setUInt` (optix::uint2 u)
- void `optix::VariableObj::setUInt` (optix::uint3 u)
- void `optix::VariableObj::setUInt` (optix::uint4 u)
- void `optix::VariableObj::set1uiv` (const unsigned int *u)
- void `optix::VariableObj::set2uiv` (const unsigned int *u)
- void `optix::VariableObj::set3uiv` (const unsigned int *u)
- void `optix::VariableObj::set4uiv` (const unsigned int *u)

Matrix setters

Set variable to have a Matrix value

- void `optix::VariableObj::setMatrix2x2fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix2x3fv` (bool transpose, const float *m)

- void `optix::VariableObj::setMatrix2x4fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix3x2fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix3x3fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix3x4fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix4x2fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix4x3fv` (bool transpose, const float *m)
- void `optix::VariableObj::setMatrix4x4fv` (bool transpose, const float *m)

Numeric value getters

Query value of a variable with scalar numeric value

- float `optix::VariableObj::getFloat` () const
- unsigned int `optix::VariableObj::getUInt` () const
- int `optix::VariableObj::getInt` () const

OptiX API object setters

Set variable to have an OptiX API object as its value

- void `optix::VariableObj::setBuffer` (Buffer buffer)
- void `optix::VariableObj::set` (Buffer buffer)
- void `optix::VariableObj::setTextureSampler` (TextureSampler texturesample)

OptiX API object getters

Retrieve OptiX API object value from a variable

- Buffer `optix::VariableObj::getBuffer` () const
- TextureSampler `optix::VariableObj::getTextureSampler` () const
- Program `optix::VariableObj::getProgram` () const

User data variable accessors

- void `optix::VariableObj::setUserData` (RTsize size, const void *ptr)
- void `optix::VariableObj::getUserData` (RTsize size, void *ptr) const
- void `optix::ContextObj::checkError` (RTresult code) const
- std::string `optix::ContextObj::getErrorString` (RTresult code) const
- Acceleration `optix::ContextObj::createAcceleration` (const char *builder, const char *traverser)
- Buffer `optix::ContextObj::createBuffer` (unsigned int type)
- Buffer `optix::ContextObj::createBuffer` (unsigned int type, RTformat format)

- Buffer `optix::ContextObj::createBuffer` (unsigned int type, RTformat format, RTsize width)
- Buffer `optix::ContextObj::createBuffer` (unsigned int type, RTformat format, RTsize width, RTsize height)
- Buffer `optix::ContextObj::createBuffer` (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)
- Buffer `optix::ContextObj::createBufferForCUDA` (unsigned int type)
- Buffer `optix::ContextObj::createBufferForCUDA` (unsigned int type, RTformat format)
- Buffer `optix::ContextObj::createBufferForCUDA` (unsigned int type, RTformat format, RTsize width)
- Buffer `optix::ContextObj::createBufferForCUDA` (unsigned int type, RTformat format, RTsize width, RTsize height)
- Buffer `optix::ContextObj::createBufferForCUDA` (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)
- Buffer `optix::ContextObj::createBufferFromGLBO` (unsigned int type, unsigned int vbo)
- TextureSampler `optix::ContextObj::createTextureSamplerFromGLImage` (unsigned int id, RTgltarget target)
- Geometry `optix::ContextObj::createGeometry` ()
- GeometryInstance `optix::ContextObj::createGeometryInstance` ()
- `template<class Iterator >`
GeometryInstance `optix::ContextObj::createGeometryInstance` (Geometry geometry, Iterator matlbegin, Iterator matlend)
- Group `optix::ContextObj::createGroup` ()
- `template<class Iterator >`
Group `optix::ContextObj::createGroup` (Iterator childbegin, Iterator childend)
- GeometryGroup `optix::ContextObj::createGeometryGroup` ()
- `template<class Iterator >`
GeometryGroup `optix::ContextObj::createGeometryGroup` (Iterator childbegin, - Iterator childend)
- Transform `optix::ContextObj::createTransform` ()
- Material `optix::ContextObj::createMaterial` ()
- Program `optix::ContextObj::createProgramFromPTXFile` (const std::string &ptx, const std::string &program_name)
- Program `optix::ContextObj::createProgramFromPTXString` (const std::string &ptx, const std::string &program_name)
- Selector `optix::ContextObj::createSelector` ()
- TextureSampler `optix::ContextObj::createTextureSampler` ()

- `template<class Iterator >`
void `optix::ContextObj::setDevices` (Iterator begin, Iterator end)
- `std::vector< int >` `optix::ContextObj::getEnabledDevices` () const
- unsigned int `optix::ContextObj::getEnabledDeviceCount` () const

- int `optix::ContextObj::getMaxTextureCount` () const
- int `optix::ContextObj::getCPUNumThreads` () const

- RTsize [optix::ContextObj::getUsedHostMemory](#) () const
- int [optix::ContextObj::getGPUPagingActive](#) () const
- int [optix::ContextObj::getGPUPagingForcedOff](#) () const
- RTsize [optix::ContextObj::getAvailableDeviceMemory](#) (int ordinal) const

- void [optix::ContextObj::setCPUNumThreads](#) (int cpu_num_threads)
- void [optix::ContextObj::setGPUPagingForcedOff](#) (int gpu_paging_forced_off)

- void [optix::ContextObj::setStackSize](#) (RTsize stack_size_bytes)
- RTsize [optix::ContextObj::getStackSize](#) () const
- void [optix::ContextObj::setTimeoutCallback](#) (RTtimeoutcallback callback, double min_polling_seconds)
- void [optix::ContextObj::setEntryPointCount](#) (unsigned int num_entry_points)
- unsigned int [optix::ContextObj::getEntryPointCount](#) () const
- void [optix::ContextObj::setRayTypeCount](#) (unsigned int num_ray_types)
- unsigned int [optix::ContextObj::getRayTypeCount](#) () const

- void [optix::ContextObj::setRayGenerationProgram](#) (unsigned int entry_point_index, Program program)
- Program [optix::ContextObj::getRayGenerationProgram](#) (unsigned int entry_point_index) const
- void [optix::ContextObj::setExceptionProgram](#) (unsigned int entry_point_index, Program program)
- Program [optix::ContextObj::getExceptionProgram](#) (unsigned int entry_point_index) const
- void [optix::ContextObj::setExceptionEnabled](#) (RTexception exception, bool enabled)
- bool [optix::ContextObj::getExceptionEnabled](#) (RTexception exception) const
- void [optix::ContextObj::setMissProgram](#) (unsigned int ray_type_index, Program program)
- Program [optix::ContextObj::getMissProgram](#) (unsigned int ray_type_index) const

- void [optix::ContextObj::launch](#) (unsigned int entry_point_index, RTsize image_width)
- void [optix::ContextObj::launch](#) (unsigned int entry_point_index, RTsize image_width, RTsize image_height)
- void [optix::ContextObj::launch](#) (unsigned int entry_point_index, RTsize image_width, RTsize image_height, RTsize image_depth)

- void [optix::ContextObj::setPrintEnabled](#) (bool enabled)
- bool [optix::ContextObj::getPrintEnabled](#) () const
- void [optix::ContextObj::setPrintBufferSize](#) (RTsize buffer_size_bytes)
- RTsize [optix::ContextObj::getPrintBufferSize](#) () const
- void [optix::ContextObj::setPrintLaunchIndex](#) (int x, int y=-1, int z=-1)
- optix::int3 [optix::ContextObj::getPrintLaunchIndex](#) () const

- Variable [optix::ContextObj::declareVariable](#) (const std::string &name)

- Variable `optix::ContextObj::queryVariable` (const std::string &name) const
- void `optix::ContextObj::removeVariable` (Variable v)
- unsigned int `optix::ContextObj::getVariableCount` () const
- Variable `optix::ContextObj::getVariable` (unsigned int index) const

- void `optix::GroupObj::setAcceleration` (Acceleration acceleration)
- Acceleration `optix::GroupObj::getAcceleration` () const

- void `optix::GroupObj::setChildCount` (unsigned int count)
- unsigned int `optix::GroupObj::getChildCount` () const
- template<typename T >
void `optix::GroupObj::setChild` (unsigned int index, T child)
- template<typename T >
T `optix::GroupObj::getChild` (unsigned int index) const

- void `optix::GeometryGroupObj::setAcceleration` (Acceleration acceleration)
- Acceleration `optix::GeometryGroupObj::getAcceleration` () const

- void `optix::GeometryGroupObj::setChildCount` (unsigned int count)
- unsigned int `optix::GeometryGroupObj::getChildCount` () const
- void `optix::GeometryGroupObj::setChild` (unsigned int index, GeometryInstance geometryinstance)
- GeometryInstance `optix::GeometryGroupObj::getChild` (unsigned int index) const

- template<typename T >
void `optix::TransformObj::setChild` (T child)
- template<typename T >
T `optix::TransformObj::getChild` () const

- void `optix::TransformObj::setMatrix` (bool transpose, const float *matrix, const float *inverse_matrix)
- void `optix::TransformObj::getMatrix` (bool transpose, float *matrix, float *inverse_matrix) const

- void `optix::SelectorObj::setVisitProgram` (Program program)
- Program `optix::SelectorObj::getVisitProgram` () const

- void `optix::SelectorObj::setChildCount` (unsigned int count)
- unsigned int `optix::SelectorObj::getChildCount` () const
- template<typename T >
void `optix::SelectorObj::setChild` (unsigned int index, T child)
- template<typename T >
T `optix::SelectorObj::getChild` (unsigned int index) const

- Variable `optix::SelectorObj::declareVariable` (const std::string &name)
- Variable `optix::SelectorObj::queryVariable` (const std::string &name) const

- void `optix::SelectorObj::removeVariable` (Variable v)
- unsigned int `optix::SelectorObj::getVariableCount` () const
- Variable `optix::SelectorObj::getVariable` (unsigned int index) const

- void `optix::AccelerationObj::markDirty` ()
- bool `optix::AccelerationObj::isDirty` () const

- void `optix::AccelerationObj::setProperty` (const std::string &name, const std::string &value)
- std::string `optix::AccelerationObj::getProperty` (const std::string &name) const
- void `optix::AccelerationObj::setBuilder` (const std::string &builder)
- std::string `optix::AccelerationObj::getBuilder` () const
- void `optix::AccelerationObj::setTraverser` (const std::string &traverser)
- std::string `optix::AccelerationObj::getTraverser` () const

- RTsize `optix::AccelerationObj::getDataSize` () const
- void `optix::AccelerationObj::getData` (void *data) const
- void `optix::AccelerationObj::setData` (const void *data, RTsize size)

- void `optix::GeometryInstanceObj::setGeometry` (Geometry geometry)
- Geometry `optix::GeometryInstanceObj::getGeometry` () const
- void `optix::GeometryInstanceObj::setMaterialCount` (unsigned int count)
- unsigned int `optix::GeometryInstanceObj::getMaterialCount` () const
- void `optix::GeometryInstanceObj::setMaterial` (unsigned int idx, Material material)
- Material `optix::GeometryInstanceObj::getMaterial` (unsigned int idx) const
- unsigned int `optix::GeometryInstanceObj::addMaterial` (Material material)

- Variable `optix::GeometryInstanceObj::declareVariable` (const std::string &name)
- Variable `optix::GeometryInstanceObj::queryVariable` (const std::string &name) const
- void `optix::GeometryInstanceObj::removeVariable` (Variable v)
- unsigned int `optix::GeometryInstanceObj::getVariableCount` () const
- Variable `optix::GeometryInstanceObj::getVariable` (unsigned int index) const

- void `optix::GeometryObj::markDirty` ()
- bool `optix::GeometryObj::isDirty` () const

- void `optix::GeometryObj::setPrimitiveCount` (unsigned int num_primitives)
- unsigned int `optix::GeometryObj::getPrimitiveCount` () const

- void `optix::GeometryObj::setBoundingBoxProgram` (Program program)
- Program `optix::GeometryObj::getBoundingBoxProgram` () const
- void `optix::GeometryObj::setIntersectionProgram` (Program program)
- Program `optix::GeometryObj::getIntersectionProgram` () const

- Variable `optix::GeometryObj::declareVariable` (const std::string &name)

- Variable `optix::GeometryObj::queryVariable` (const std::string &name) const
- void `optix::GeometryObj::removeVariable` (Variable v)
- unsigned int `optix::GeometryObj::getVariableCount` () const
- Variable `optix::GeometryObj::getVariable` (unsigned int index) const

- void `optix::MaterialObj::setClosestHitProgram` (unsigned int ray_type_index, - Program program)
- Program `optix::MaterialObj::getClosestHitProgram` (unsigned int ray_type_index) const
- void `optix::MaterialObj::setAnyHitProgram` (unsigned int ray_type_index, - Program program)
- Program `optix::MaterialObj::getAnyHitProgram` (unsigned int ray_type_index) const

- Variable `optix::MaterialObj::declareVariable` (const std::string &name)
- Variable `optix::MaterialObj::queryVariable` (const std::string &name) const
- void `optix::MaterialObj::removeVariable` (Variable v)
- unsigned int `optix::MaterialObj::getVariableCount` () const
- Variable `optix::MaterialObj::getVariable` (unsigned int index) const

- void `optix::TextureSamplerObj::setMipLevelCount` (unsigned int num_mip_levels)
- unsigned int `optix::TextureSamplerObj::getMipLevelCount` () const
- void `optix::TextureSamplerObj::setArraySize` (unsigned int num_textures_in_array)
- unsigned int `optix::TextureSamplerObj::getArraySize` () const
- void `optix::TextureSamplerObj::setWrapMode` (unsigned int dim, RTwrapmode wrapmode)
- RTwrapmode `optix::TextureSamplerObj::getWrapMode` (unsigned int dim) const
- void `optix::TextureSamplerObj::setFilteringModes` (RTfiltermode minification, RTfiltermode magnification, RTfiltermode mipmapping)
- void `optix::TextureSamplerObj::getFilteringModes` (RTfiltermode &minification, RTfiltermode &magnification, RTfiltermode &mipmapping) const
- void `optix::TextureSamplerObj::setMaxAnisotropy` (float value)
- float `optix::TextureSamplerObj::getMaxAnisotropy` () const
- void `optix::TextureSamplerObj::setReadMode` (RTtexturereadmode readmode)
- RTtexturereadmode `optix::TextureSamplerObj::getReadMode` () const
- void `optix::TextureSamplerObj::setIndexingMode` (RTtextureindexmode indexmode)
- RTtextureindexmode `optix::TextureSamplerObj::getIndexingMode` () const

- int `optix::TextureSamplerObj::getIdx` () const

- void `optix::TextureSamplerObj::setBuffer` (unsigned int texture_array_idx, unsigned int mip_level, Buffer buffer)
- Buffer `optix::TextureSamplerObj::getBuffer` (unsigned int texture_array_idx, unsigned int mip_level) const

- void [optix::TextureSamplerObj::registerGLTexture](#) ()
- void [optix::TextureSamplerObj::unregisterGLTexture](#) ()

- void [optix::BufferObj::setFormat](#) (RTformat format)
- RTformat [optix::BufferObj::getFormat](#) () const
- void [optix::BufferObj::setElementSize](#) (RTsize size_of_element)
- RTsize [optix::BufferObj::getElementSize](#) () const
- void [optix::BufferObj::getDevicePointer](#) (unsigned int optix_device_number, C-Udeviceptr *device_pointer)
- void [optix::BufferObj::setDevicePointer](#) (unsigned int optix_device_number, C-Udeviceptr device_pointer)
- void [optix::BufferObj::markDirty](#) ()
- void [optix::BufferObj::setSize](#) (RTsize width)
- void [optix::BufferObj::getSize](#) (RTsize &width) const
- void [optix::BufferObj::setSize](#) (RTsize width, RTsize height)
- void [optix::BufferObj::getSize](#) (RTsize &width, RTsize &height) const
- void [optix::BufferObj::setSize](#) (RTsize width, RTsize height, RTsize depth)
- void [optix::BufferObj::getSize](#) (RTsize &width, RTsize &height, RTsize &depth) const
- void [optix::BufferObj::setSize](#) (unsigned int dimensionality, const RTsize *dims)
- void [optix::BufferObj::getSize](#) (unsigned int dimensionality, RTsize *dims) const
- unsigned int [optix::BufferObj::getDimensionality](#) () const

- unsigned int [optix::BufferObj::getGLBOld](#) () const
- void [optix::BufferObj::registerGLBuffer](#) ()
- void [optix::BufferObj::unregisterGLBuffer](#) ()

- void * [optix::BufferObj::map](#) ()
- void [optix::BufferObj::unmap](#) ()

1.2.2 Typedef Documentation

1.2.2.1 typedef Handle<AccelerationObj> optix::Acceleration

Use this to manipulate RTacceleration objects.

Definition at line 211 of file [optixpp_namespace.h](#).

1.2.2.2 typedef Handle<BufferObj> optix::Buffer

Use this to manipulate RTbuffer objects.

Definition at line 212 of file [optixpp_namespace.h](#).

1.2.2.3 typedef Handle<ContextObj> optix::Context

Use this to manipulate RTcontext objects.

Definition at line 213 of file [optixpp_namespace.h](#).

1.2.2.4 typedef Handle<GeometryObj> optix::Geometry

Use this to manipulate RTgeometry objects.

Definition at line 214 of file [optixpp_namespace.h](#).

1.2.2.5 typedef Handle<GeometryGroupObj> optix::GeometryGroup

Use this to manipulate RTgeometrygroup objects.

Definition at line 215 of file [optixpp_namespace.h](#).

1.2.2.6 typedef Handle<GeometryInstanceObj> optix::GeometryInstance

Use this to manipulate RTgeometryinstance objects.

Definition at line 216 of file [optixpp_namespace.h](#).

1.2.2.7 typedef Handle<GroupObj> optix::Group

Use this to manipulate RTgroup objects.

Definition at line 217 of file [optixpp_namespace.h](#).

1.2.2.8 typedef Handle<MaterialObj> optix::Material

Use this to manipulate RTmaterial objects.

Definition at line 218 of file [optixpp_namespace.h](#).

1.2.2.9 typedef Handle<ProgramObj> optix::Program

Use this to manipulate RTprogram objects.

Definition at line 219 of file [optixpp_namespace.h](#).

1.2.2.10 typedef Handle<SelectorObj> optix::Selector

Use this to manipulate RTselector objects.

Definition at line 220 of file [optixpp_namespace.h](#).

1.2.2.11 typedef Handle<TextureSamplerObj> optix::TextureSampler

Use this to manipulate RTtexturesampler objects.

Definition at line 221 of file [optixpp_namespace.h](#).

1.2.2.12 typedef Handle<TransformObj> optix::Transform

Use this to manipulate RTtransform objects.

Definition at line 222 of file [optixpp_namespace.h](#).

1.2.2.13 typedef Handle<VariableObj> optix::Variable

Use this to manipulate RTvariable objects.

Definition at line 223 of file [optixpp_namespace.h](#).

1.2.3 Function Documentation

1.2.3.1 `unsigned int optix::GeometryInstanceObj::addMaterial (Material material)`
[inline]

Adds the provided material and returns the index to newly added material; increases material count by one.

Definition at line 2631 of file [optixpp_namespace.h](#).

1.2.3.2 `void optix::APIObj::checkError (RResult code) const` [inline, virtual]

Check the given result code and throw an error with appropriate message if the code is not RTsuccess

Reimplemented in [optix::ContextObj](#).

Definition at line 1523 of file [optixpp_namespace.h](#).

1.2.3.3 `void optix::APIObj::checkError (RResult code, Context context) const`
[inline, virtual]

Definition at line 1531 of file [optixpp_namespace.h](#).

1.2.3.4 `void optix::ContextObj::checkError (RResult code) const` [inline, virtual]

See [APIObj::checkError](#)

Reimplemented from [optix::APIObj](#).

Definition at line 1551 of file [optixpp_namespace.h](#).

1.2.3.5 `void optix::APIObj::checkErrorNoGetContext (RResult code) const`
[inline]

Definition at line 1539 of file [optixpp_namespace.h](#).

1.2.3.6 `void optix::ContextObj::compile ()` [inline]

See [rtContextCompile](#).

Definition at line 2061 of file [optixpp_namespace.h](#).

1.2.3.7 `Context optix::ContextObj::create ()` [inline, static]

Creates a Context object. See [rtContextCreate](#).

Definition at line 1582 of file [optixpp_namespace.h](#).

1.2.3.8 Acceleration `optix::ContextObj::createAcceleration (const char * builder, const char * traverser)` `[inline]`

See `rtAccelerationCreate`

Definition at line 1602 of file `optixpp_namespace.h`.

1.2.3.9 Buffer `optix::ContextObj::createBuffer (unsigned int type)` `[inline]`

Create a buffer with given RTbuffertype. See `rtBufferCreate`.

Definition at line 1612 of file `optixpp_namespace.h`.

1.2.3.10 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format)` `[inline]`

Create a buffer with given RTbuffertype and RTformat. See `rtBufferCreate`, `rtBufferSetFormat`.

Definition at line 1619 of file `optixpp_namespace.h`.

1.2.3.11 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize1D`.

Definition at line 1627 of file `optixpp_namespace.h`.

1.2.3.12 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width, RTsize height)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize2D`.

Definition at line 1636 of file `optixpp_namespace.h`.

1.2.3.13 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize3D`.

Definition at line 1645 of file `optixpp_namespace.h`.

1.2.3.14 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type)` `[inline]`

Create a buffer for CUDA with given RTbuffertype. See `rtBufferCreate`.

Definition at line 1654 of file `optixpp_namespace.h`.

1.2.3.15 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format) [inline]`

Create a buffer for CUDA with given RTbuffertype and RTformat. See `rtBufferCreate`, `rtBufferSetFormat`.

Definition at line 1661 of file [optixpp_namespace.h](#).

1.2.3.16 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width) [inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize1D`.

Definition at line 1669 of file [optixpp_namespace.h](#).

1.2.3.17 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width, RTsize height) [inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize2D`.

Definition at line 1678 of file [optixpp_namespace.h](#).

1.2.3.18 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth) [inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize3D`.

Definition at line 1687 of file [optixpp_namespace.h](#).

1.2.3.19 Buffer `optix::ContextObj::createBufferFromGLBO (unsigned int type, unsigned int vbo) [inline]`

Create buffer from GL buffer object. See `rtBufferCreateFromGLBO`.

Definition at line 1696 of file [optixpp_namespace.h](#).

1.2.3.20 Geometry `optix::ContextObj::createGeometry () [inline]`

See `rtGeometryCreate`.

Definition at line 1771 of file [optixpp_namespace.h](#).

1.2.3.21 GeometryGroup `optix::ContextObj::createGeometryGroup () [inline]`

See `rtGeometryGroupCreate`.

Definition at line 1821 of file [optixpp_namespace.h](#).

1.2.3.22 `template<class Iterator > GeometryGroup optix::ContextObj::createGeometryGroup (Iterator childbegin, Iterator childend)`
[inline]

Create a GeometryGroup with a set of child nodes. See `rtGeometryGroupCreate`, `rtGeometryGroupSetChildCount` and `rtGeometryGroupSetChild`

Definition at line 1829 of file `optixpp_namespace.h`.

1.2.3.23 `GeometryInstance optix::ContextObj::createGeometryInstance ()`
[inline]

See `rtGeometryInstanceCreate`.

Definition at line 1778 of file `optixpp_namespace.h`.

1.2.3.24 `template<class Iterator > GeometryInstance optix::ContextObj::createGeometryInstance (Geometry geometry, Iterator matlbegin, Iterator matlend)`

Create a geometry instance with a Geometry object and a set of associated materials. See `rtGeometryInstanceCreate`, `rtGeometryInstanceSetMaterialCount`, and `rtGeometryInstanceSetMaterial`

Definition at line 1786 of file `optixpp_namespace.h`.

1.2.3.25 `Group optix::ContextObj::createGroup ()` [inline]

See `rtGroupCreate`.

Definition at line 1800 of file `optixpp_namespace.h`.

1.2.3.26 `template<class Iterator > Group optix::ContextObj::createGroup (Iterator childbegin, Iterator childend)` [inline]

Create a Group with a set of child nodes. See `rtGroupCreate`, `rtGroupSetChildCount` and `rtGroupSetChild`

Definition at line 1808 of file `optixpp_namespace.h`.

1.2.3.27 `Material optix::ContextObj::createMaterial ()` [inline]

See `rtMaterialCreate`.

Definition at line 1849 of file `optixpp_namespace.h`.

1.2.3.28 `Program optix::ContextObj::createProgramFromPTXFile (const std::string & ptx, const std::string & program_name)` [inline]

See `rtProgramCreateFromPTXFile`.

Definition at line 1856 of file `optixpp_namespace.h`.

1.2.3.29 Program `optix::ContextObj::createProgramFromPTXString (const std::string & ptx, const std::string & program_name)` `[inline]`

See `rtProgramCreateFromPTXString`.

Definition at line 1863 of file `optixpp_namespace.h`.

1.2.3.30 Selector `optix::ContextObj::createSelector ()` `[inline]`

See `rtSelectorCreate`.

Definition at line 1870 of file `optixpp_namespace.h`.

1.2.3.31 TextureSampler `optix::ContextObj::createTextureSampler ()` `[inline]`

See `rtTextureSamplerCreate`.

Definition at line 1877 of file `optixpp_namespace.h`.

1.2.3.32 TextureSampler `optix::ContextObj::createTextureSamplerFromGLImage (unsigned int id, RTgltarget target)` `[inline]`

Create TextureSampler from GL image. See `rtTextureSamplerCreateFromGLImage`.

Definition at line 1764 of file `optixpp_namespace.h`.

1.2.3.33 Transform `optix::ContextObj::createTransform ()` `[inline]`

See `rtTransformCreate`.

Definition at line 1842 of file `optixpp_namespace.h`.

1.2.3.34 Variable `optix::ContextObj::declareVariable (const std::string & name)` `[inline, virtual]`

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2125 of file `optixpp_namespace.h`.

1.2.3.35 Variable `optix::ProgramObj::declareVariable (const std::string & name)` `[inline, virtual]`

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2183 of file `optixpp_namespace.h`.

1.2.3.36 Variable `optix::SelectorObj::declareVariable (const std::string & name)` `[inline]`

Definition at line 2297 of file `optixpp_namespace.h`.

1.2.3.37 Variable `optix::GeometryInstanceObj::declareVariable (const std::string & name)` [inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2639 of file `optixpp_namespace.h`.

1.2.3.38 Variable `optix::GeometryObj::declareVariable (const std::string & name)` [inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2732 of file `optixpp_namespace.h`.

1.2.3.39 Variable `optix::MaterialObj::declareVariable (const std::string & name)` [inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2825 of file `optixpp_namespace.h`.

1.2.3.40 void `optix::ContextObj::destroy ()` [inline, virtual]

Destroy Context and all of its associated objects. See `rtContextDestroy`.

Implements `optix::DestroyableObj`.

Definition at line 1591 of file `optixpp_namespace.h`.

1.2.3.41 void `optix::ProgramObj::destroy ()` [inline, virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2164 of file `optixpp_namespace.h`.

1.2.3.42 void `optix::GroupObj::destroy ()` [inline, virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2221 of file `optixpp_namespace.h`.

1.2.3.43 void `optix::GeometryGroupObj::destroy ()` [inline, virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2378 of file [optixpp_namespace.h](#).

1.2.3.44 `void optix::TransformObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2438 of file [optixpp_namespace.h](#).

1.2.3.45 `void optix::SelectorObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2240 of file [optixpp_namespace.h](#).

1.2.3.46 `void optix::AccelerationObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2486 of file [optixpp_namespace.h](#).

1.2.3.47 `void optix::GeometryInstanceObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2575 of file [optixpp_namespace.h](#).

1.2.3.48 `void optix::GeometryObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2677 of file [optixpp_namespace.h](#).

1.2.3.49 `void optix::MaterialObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2782 of file [optixpp_namespace.h](#).

1.2.3.50 `void optix::TextureSamplerObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2863 of file [optixpp_namespace.h](#).

1.2.3.51 void `optix::BufferObj::destroy` () [`inline`, `virtual`]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 3032 of file `optixpp_namespace.h`.

1.2.3.52 RTvariable `optix::VariableObj::get` () [`inline`]

Get the OptiX C API object wrapped by this instance.

Definition at line 3550 of file `optixpp_namespace.h`.

1.2.3.53 RTcontext `optix::ContextObj::get` () [`inline`]

Return the OptiX C API RTcontext object.

Definition at line 2159 of file `optixpp_namespace.h`.

1.2.3.54 RTprogram `optix::ProgramObj::get` () [`inline`]

Definition at line 2216 of file `optixpp_namespace.h`.

1.2.3.55 RTgroup `optix::GroupObj::get` () [`inline`]

Get the underlying OptiX C API RTgroup opaque pointer.

Definition at line 2373 of file `optixpp_namespace.h`.

1.2.3.56 RTgeometrygroup `optix::GeometryGroupObj::get` () [`inline`]

Get the underlying OptiX C API RTgeometrygroup opaque pointer.

Definition at line 2433 of file `optixpp_namespace.h`.

1.2.3.57 RTtransform `optix::TransformObj::get` () [`inline`]

Get the underlying OptiX C API RTtransform opaque pointer.

Definition at line 2481 of file `optixpp_namespace.h`.

1.2.3.58 RTselector `optix::SelectorObj::get` () [`inline`]

Get the underlying OptiX C API RTselector opaque pointer.

Definition at line 2330 of file `optixpp_namespace.h`.

1.2.3.59 RTacceleration `optix::AccelerationObj::get` () [`inline`]

Get the underlying OptiX C API RTacceleration opaque pointer.

Definition at line 2570 of file `optixpp_namespace.h`.

1.2.3.60 RTgeometryinstance `optix::GeometryInstanceObj::get` () [`inline`]

Get the underlying OptiX C API RTgeometryinstance opaque pointer.

Definition at line 2672 of file [optixpp_namespace.h](#).

1.2.3.61 `RTgeometry optix::GeometryObj::get () [inline]`

Get the underlying OptiX C API RTgeometry opaque pointer.

Definition at line 2777 of file [optixpp_namespace.h](#).

1.2.3.62 `RTmaterial optix::MaterialObj::get () [inline]`

Get the underlying OptiX C API RTmaterial opaque pointer.

Definition at line 2858 of file [optixpp_namespace.h](#).

1.2.3.63 `RTtexturesampler optix::TextureSamplerObj::get () [inline]`

Get the underlying OptiX C API RTtexturesampler opaque pointer.

Definition at line 2983 of file [optixpp_namespace.h](#).

1.2.3.64 `RTbuffer optix::BufferObj::get () [inline]`

Get the underlying OptiX C API RTbuffer opaque pointer.

Definition at line 3222 of file [optixpp_namespace.h](#).

1.2.3.65 `Acceleration optix::GroupObj::getAcceleration () const [inline]`

Query the Acceleration structure for this group. See `rtGroupGetAcceleration`.

Definition at line 2340 of file [optixpp_namespace.h](#).

1.2.3.66 `Acceleration optix::GeometryGroupObj::getAcceleration () const [inline]`

Query the Acceleration structure for this group. See `rtGeometryGroupGetAcceleration`.

Definition at line 2402 of file [optixpp_namespace.h](#).

1.2.3.67 `std::string optix::VariableObj::getAnnotation () const [inline]`

Retrieve the annotation associated with the variable.

Definition at line 3536 of file [optixpp_namespace.h](#).

1.2.3.68 `Program optix::MaterialObj::getAnyHitProgram (unsigned int ray_type_index) const [inline]`

Get any hit program for this material at the given `ray_type` index. See `rtMaterialGetAnyHitProgram`.

Definition at line 2818 of file [optixpp_namespace.h](#).

1.2.3.69 `unsigned int optix::TextureSamplerObj::getArraySize () const [inline]`

Query the texture array size for this sampler. See `rtTextureSamplerGetArraySize`.

Definition at line [2899](#) of file [optixpp_namespace.h](#).

1.2.3.70 `RTsize optix::ContextObj::getAvailableDeviceMemory (int ordinal) const`
`[inline]`

See `rtContextGetAttribute`.

Definition at line [1949](#) of file [optixpp_namespace.h](#).

1.2.3.71 `Program optix::GeometryObj::getBoundingBoxProgram () const`
`[inline]`

Get the bounding box program for this geometry. See `rtGeometryGetBoundingBoxProgram`.

Definition at line [2713](#) of file [optixpp_namespace.h](#).

1.2.3.72 `Buffer optix::VariableObj::getBuffer () const` `[inline]`

Definition at line [3521](#) of file [optixpp_namespace.h](#).

1.2.3.73 `Buffer optix::TextureSamplerObj::getBuffer (unsigned int texture_array_idx,
unsigned int mip_level) const` `[inline]`

Get the underlying buffer used for texture storage. `rtTextureSamplerGetBuffer`.

Definition at line [2976](#) of file [optixpp_namespace.h](#).

1.2.3.74 `std::string optix::AccelerationObj::getBuilder () const` `[inline]`

Query the acceleration structure builder. See `rtAccelerationGetBuilder`.

Definition at line [2534](#) of file [optixpp_namespace.h](#).

1.2.3.75 `template<typename T > T optix::GroupObj::getChild (unsigned int index)
const` `[inline]`

Query an indexed child within this group. See `rtGroupGetChild`.

Definition at line [2366](#) of file [optixpp_namespace.h](#).

1.2.3.76 `GeometryInstance optix::GeometryGroupObj::getChild (unsigned int index)
const` `[inline]`

Query an indexed `GeometryInstance` within this group. See `rtGeometryGroupGetChild`.

Definition at line [2426](#) of file [optixpp_namespace.h](#).

1.2.3.77 `template<typename T > T optix::TransformObj::getChild () const`
`[inline]`

Set the child node of this transform. See `rtTransformGetChild`.

Definition at line [2464](#) of file [optixpp_namespace.h](#).

1.2.3.78 `template<typename T > T optix::SelectorObj::getChild (unsigned int index)
const [inline]`

Query an indexed child within this group. See `rtSelectorGetChild`.

Definition at line 2290 of file `optixpp_namespace.h`.

1.2.3.79 `unsigned int optix::GroupObj::getChildCount () const [inline]`

Query the number of children for this group. See `rtGroupGetChildCount`.

Definition at line 2352 of file `optixpp_namespace.h`.

1.2.3.80 `unsigned int optix::GeometryGroupObj::getChildCount () const
[inline]`

Query the number of children for this group. See `rtGeometryGroupGetChildCount`.

Definition at line 2414 of file `optixpp_namespace.h`.

1.2.3.81 `unsigned int optix::SelectorObj::getChildCount () const [inline]`

Query the number of children for this group. See `rtSelectorGetChildCount`.

Definition at line 2276 of file `optixpp_namespace.h`.

1.2.3.82 `Program optix::MaterialObj::getClosestHitProgram (unsigned int
ray_type_index) const [inline]`

Get closest hit program for this material at the given `ray_type` index. See `rtMaterialGetClosestHitProgram`.

Definition at line 2806 of file `optixpp_namespace.h`.

1.2.3.83 `Context optix::VariableObj::getContext () const [inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 3227 of file `optixpp_namespace.h`.

1.2.3.84 `Context optix::ContextObj::getContext () const [inline, virtual]`

Retrieve the Context object associated with this APIObject. In this case, simply returns itself.

Implements `optix::APIObj`.

Definition at line 1546 of file `optixpp_namespace.h`.

1.2.3.85 `Context optix::ProgramObj::getContext () const [inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2176 of file [optixpp_namespace.h](#).

1.2.3.86 Context `optix::GroupObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2233 of file [optixpp_namespace.h](#).

1.2.3.87 Context `optix::GeometryGroupObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2390 of file [optixpp_namespace.h](#).

1.2.3.88 Context `optix::TransformObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2450 of file [optixpp_namespace.h](#).

1.2.3.89 Context `optix::SelectorObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2252 of file [optixpp_namespace.h](#).

1.2.3.90 Context `optix::AccelerationObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2498 of file [optixpp_namespace.h](#).

1.2.3.91 Context `optix::GeometryInstanceObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2587 of file [optixpp_namespace.h](#).

1.2.3.92 Context `optix::GeometryObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2689 of file [optixpp_namespace.h](#).

1.2.3.93 Context `optix::MaterialObj::getContext () const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2794 of file [optixpp_namespace.h](#).

1.2.3.94 Context `optix::TextureSamplerObj::getContext () const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 2875 of file [optixpp_namespace.h](#).

1.2.3.95 Context `optix::BufferObj::getContext () const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements [optix::APIObj](#).

Definition at line 3044 of file [optixpp_namespace.h](#).

1.2.3.96 `int optix::ContextObj::getCPUNumThreads () const` `[inline]`

See `rtContextGetAttribute`.

Definition at line 1921 of file [optixpp_namespace.h](#).

1.2.3.97 `void optix::AccelerationObj::getData (void * data) const` `[inline]`

Get the marshalled acceleration data. See `rtAccelerationGetData`.

Definition at line 2560 of file [optixpp_namespace.h](#).

1.2.3.98 `RTsize optix::AccelerationObj::getDataSize () const` `[inline]`

Query the size of the marshalled acceleration data. See `rtAccelerationGetDataSize`.

Definition at line 2553 of file [optixpp_namespace.h](#).

1.2.3.99 `void optix::ContextObj::getDeviceAttribute (int ordinal, RTdeviceattribute attrib, RTsize size, void * p)` `[inline, static]`

Call `rtDeviceGetAttribute` and return the desired attribute value.

Definition at line 1576 of file [optixpp_namespace.h](#).

1.2.3.100 `unsigned int optix::ContextObj::getDeviceCount () [inline, static]`

Call `rtDeviceGetDeviceCount` and returns number of valid devices.

Definition at line 1557 of file [optixpp_namespace.h](#).

1.2.3.101 `std::string optix::ContextObj::getDeviceName (int ordinal) [inline, static]`

Call `rtDeviceGetAttribute` and return the name of the device.

Definition at line 1566 of file [optixpp_namespace.h](#).

1.2.3.102 `void optix::BufferObj::getDevicePointer (unsigned int optix_device_number, CUdeviceptr * device_pointer) [inline]`

Get the pointer to buffer memory on a specific device. See `rtBufferGetDevicePointer`.

Definition at line 3075 of file [optixpp_namespace.h](#).

1.2.3.103 `unsigned int optix::BufferObj::getDimensionality () const [inline]`

Query dimensionality of buffer. See `rtBufferGetDimensionality`.

Definition at line 3130 of file [optixpp_namespace.h](#).

1.2.3.104 `RTsize optix::BufferObj::getElementSize () const [inline]`

Query the data element size for user format buffers. See `rtBufferGetElementSize`.

Definition at line 3068 of file [optixpp_namespace.h](#).

1.2.3.105 `unsigned int optix::ContextObj::getEnabledDeviceCount () const [inline]`

See `rtContextGetDeviceCount`. As opposed to `getDeviceCount`, this returns only the number of enabled devices.

Definition at line 1907 of file [optixpp_namespace.h](#).

1.2.3.106 `std::vector< int > optix::ContextObj::getEnabledDevices () const [inline]`

See `rtContextGetDevices`. This returns the list of currently enabled devices.

Definition at line 1899 of file [optixpp_namespace.h](#).

1.2.3.107 `unsigned int optix::ContextObj::getEntryPointCount () const [inline]`

See `rtContextgetEntryPointCount`.

Definition at line 1990 of file [optixpp_namespace.h](#).

1.2.3.108 `std::string optix::ContextObj::getErrorString (RResult code) const`
[inline]

See `rtContextGetErrorString`.

Definition at line 1884 of file [optixpp_namespace.h](#).

1.2.3.109 `bool optix::ContextObj::getExceptionEnabled (RException exception)`
`const` [inline]

See `rtContextGetExceptionEnabled`.

Definition at line 2029 of file [optixpp_namespace.h](#).

1.2.3.110 `Program optix::ContextObj::getExceptionProgram (unsigned int`
`entry_point_index) const` [inline]

See `rtContextGetExceptionProgram`.

Definition at line 2016 of file [optixpp_namespace.h](#).

1.2.3.111 `void optix::TextureSamplerObj::getFilteringModes (RTfiltermode &`
`minification, RTfiltermode & magnification, RTfiltermode & mipmapping) const`
[inline]

Query filtering modes for this sampler. See `rtTextureSamplerGetFilteringModes`.

Definition at line 2923 of file [optixpp_namespace.h](#).

1.2.3.112 `float optix::VariableObj::getFloat () const` [inline]

Definition at line 3445 of file [optixpp_namespace.h](#).

1.2.3.113 `RTformat optix::BufferObj::getFormat () const` [inline]

Query the data format for the buffer. See `rtBufferGetFormat`.

Definition at line 3056 of file [optixpp_namespace.h](#).

1.2.3.114 `Geometry optix::GeometryInstanceObj::getGeometry () const`
[inline]

Get the geometry object associated with this instance. See `rtGeometryInstanceGetGeometry`.

Definition at line 2599 of file [optixpp_namespace.h](#).

1.2.3.115 `unsigned int optix::BufferObj::getGLBOld () const` [inline]

Queries the OpenGL Buffer Object ID associated with this buffer. See `rtBufferGetGLBOld`.

Definition at line 3137 of file [optixpp_namespace.h](#).

1.2.3.116 `int optix::ContextObj::getGPUPagingActive() const [inline]`

See `rtContextGetAttribute`.

Definition at line 1935 of file `optixpp_namespace.h`.

1.2.3.117 `int optix::ContextObj::getGPUPagingForcedOff() const [inline]`

See `rtContextGetAttribute`.

Definition at line 1942 of file `optixpp_namespace.h`.

1.2.3.118 `int optix::TextureSamplerObj::getId() const [inline]`

Returns the device-side ID of this sampler.

Definition at line 2940 of file `optixpp_namespace.h`.

1.2.3.119 `RTtextureindexmode optix::TextureSamplerObj::getIndexingMode() const [inline]`

Query texture indexing mode for this sampler. See `rtTextureSamplerGetIndexingMode`.

Definition at line 2964 of file `optixpp_namespace.h`.

1.2.3.120 `int optix::VariableObj::getInt() const [inline]`

Definition at line 3459 of file `optixpp_namespace.h`.

1.2.3.121 `Program optix::GeometryObj::getIntersectionProgram() const [inline]`

Get the intersection program for this geometry. See `rtGeometryGetIntersectionProgram`.

Definition at line 2725 of file `optixpp_namespace.h`.

1.2.3.122 `Material optix::GeometryInstanceObj::getMaterial(unsigned int idx) const [inline]`

Get the material at given index. See `rtGeometryInstanceGetMaterial`.

Definition at line 2623 of file `optixpp_namespace.h`.

1.2.3.123 `unsigned int optix::GeometryInstanceObj::getMaterialCount() const [inline]`

Query the number of materials associated with this instance. See `rtGeometryInstanceGetMaterialCount`.

Definition at line 2611 of file `optixpp_namespace.h`.

1.2.3.124 `void optix::TransformObj::getMatrix (bool transpose, float * matrix, float * inverse_matrix) const` [inline]

Get the transform matrix for this node. See `rtTransformGetMatrix`.

Definition at line 2476 of file `optixpp_namespace.h`.

1.2.3.125 `float optix::TextureSamplerObj::getMaxAnisotropy () const` [inline]

Query maximum anisotropy for this sampler. See `rtTextureSamplerGetMaxAnisotropy`.

Definition at line 2933 of file `optixpp_namespace.h`.

1.2.3.126 `int optix::ContextObj::getMaxTextureCount () const` [inline]

See `rtContextGetAttribute`

Definition at line 1914 of file `optixpp_namespace.h`.

1.2.3.127 `unsigned int optix::TextureSamplerObj::getMipLevelCount () const` [inline]

Query the number of mip levels for this sampler. See `rtTextureSamplerGetMipLevelCount`.

Definition at line 2887 of file `optixpp_namespace.h`.

1.2.3.128 `Program optix::ContextObj::getMissProgram (unsigned int ray_type_index) const` [inline]

See `rtContextGetMissProgram`.

Definition at line 2054 of file `optixpp_namespace.h`.

1.2.3.129 `std::string optix::VariableObj::getName () const` [inline]

Retrieve the name of the variable.

Definition at line 3529 of file `optixpp_namespace.h`.

1.2.3.130 `unsigned int optix::GeometryObj::getPrimitiveCount () const` [inline]

Query the number of primitives in this geometry objects (eg, number of triangles in mesh). See `rtGeometryGetPrimitiveCount`

Definition at line 2701 of file `optixpp_namespace.h`.

1.2.3.131 `RTsize optix::ContextObj::getPrintBufferSize () const` [inline]

See `rtContextGetPrintBufferSize`.

Definition at line 2106 of file `optixpp_namespace.h`.

1.2.3.132 `bool optix::ContextObj::getPrintEnabled () const [inline]`

See `rtContextGetPrintEnabled`.

Definition at line 2094 of file `optixpp_namespace.h`.

1.2.3.133 `optix::int3 optix::ContextObj::getPrintLaunchIndex () const [inline]`

See `rtContextGetPrintLaunchIndex`.

Definition at line 2118 of file `optixpp_namespace.h`.

1.2.3.134 `optix::Program optix::VariableObj::getProgram () const [inline]`

Definition at line 3570 of file `optixpp_namespace.h`.

1.2.3.135 `std::string optix::AccelerationObj::getProperty (const std::string & name) const [inline]`

Query properties specifying Acceleration builder/traverser behavior. See `rtAccelerationGetProperty`.

Definition at line 2522 of file `optixpp_namespace.h`.

1.2.3.136 `Program optix::ContextObj::getRayGenerationProgram (unsigned int entry_point_index) const [inline]`

See `rtContextGetRayGenerationProgram`.

Definition at line 2003 of file `optixpp_namespace.h`.

1.2.3.137 `unsigned int optix::ContextObj::getRayTypeCount () const [inline]`

See `rtContextGetRayTypeCount`.

Definition at line 2042 of file `optixpp_namespace.h`.

1.2.3.138 `RTtexturereadmode optix::TextureSamplerObj::getReadMode () const [inline]`

Query texture read mode for this sampler. See `rtTextureSamplerGetReadMode`.

Definition at line 2952 of file `optixpp_namespace.h`.

1.2.3.139 `int optix::ContextObj::getRunningState () const [inline]`

See `rtContextGetRunningState`.

Definition at line 2082 of file `optixpp_namespace.h`.

1.2.3.140 `RTsize optix::VariableObj::getSize () const [inline]`

Get the size of the variable data in bytes (eg, float4 returns 4*sizeof(float))

Definition at line 3555 of file `optixpp_namespace.h`.

1.2.3.141 `void optix::BufferObj::getSize (RTsize & width) const [inline]`

Query 1D buffer dimension. See `rtBufferGetSize1D`.

Definition at line 3095 of file `optixpp_namespace.h`.

1.2.3.142 `void optix::BufferObj::getSize (RTsize & width, RTsize & height) const [inline]`

Query 2D buffer dimension. See `rtBufferGetSize2D`.

Definition at line 3105 of file `optixpp_namespace.h`.

1.2.3.143 `void optix::BufferObj::getSize (RTsize & width, RTsize & height, RTsize & depth) const [inline]`

Query 3D buffer dimension. See `rtBufferGetSize3D`.

Definition at line 3115 of file `optixpp_namespace.h`.

1.2.3.144 `void optix::BufferObj::getSize (unsigned int dimensionality, RTsize * dims) const [inline]`

Query dimensions of buffer. See `rtBufferGetSize`.

Definition at line 3125 of file `optixpp_namespace.h`.

1.2.3.145 `RTsize optix::ContextObj::getStackSize () const [inline]`

See `rtContextGetStackSize`.

Definition at line 1973 of file `optixpp_namespace.h`.

1.2.3.146 `optix::TextureSampler optix::VariableObj::getTextureSampler () const [inline]`

Definition at line 3562 of file `optixpp_namespace.h`.

1.2.3.147 `std::string optix::AccelerationObj::getTraverser () const [inline]`

Query the acceleration structure traverser. See `rtAccelerationGetTraverser`.

Definition at line 2546 of file `optixpp_namespace.h`.

1.2.3.148 `RObjectType optix::VariableObj::getType () const [inline]`

Query the object type of the variable.

Definition at line 3543 of file `optixpp_namespace.h`.

1.2.3.149 `unsigned int optix::VariableObj::getUInt () const [inline]`

Definition at line 3452 of file `optixpp_namespace.h`.

1.2.3.150 `RTsize optix::ContextObj::getUsedHostMemory () const [inline]`

See `rtContextGetAttribute`.

Definition at line 1928 of file `optixpp_namespace.h`.

1.2.3.151 `void optix::VariableObj::getUserData (RTsize size, void * ptr) const [inline]`

Retrieve a user defined type given the sizeof the user object.

Definition at line 3481 of file `optixpp_namespace.h`.

1.2.3.152 `Variable optix::ContextObj::getVariable (unsigned int index) const [inline, virtual]`

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2151 of file `optixpp_namespace.h`.

1.2.3.153 `Variable optix::ProgramObj::getVariable (unsigned int index) const [inline, virtual]`

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2209 of file `optixpp_namespace.h`.

1.2.3.154 `Variable optix::SelectorObj::getVariable (unsigned int index) const [inline]`

Definition at line 2323 of file `optixpp_namespace.h`.

1.2.3.155 `Variable optix::GeometryInstanceObj::getVariable (unsigned int index) const [inline, virtual]`

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2665 of file `optixpp_namespace.h`.

1.2.3.156 `Variable optix::GeometryObj::getVariable (unsigned int index) const [inline, virtual]`

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2758 of file `optixpp_namespace.h`.

1.2.3.157 Variable `optix::MaterialObj::getVariable (unsigned int index) const`
[inline, virtual]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2851 of file `optixpp_namespace.h`.

1.2.3.158 `unsigned int optix::ContextObj::getVariableCount () const` [inline, virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2144 of file `optixpp_namespace.h`.

1.2.3.159 `unsigned int optix::ProgramObj::getVariableCount () const` [inline, virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2202 of file `optixpp_namespace.h`.

1.2.3.160 `unsigned int optix::SelectorObj::getVariableCount () const` [inline]

Definition at line 2316 of file `optixpp_namespace.h`.

1.2.3.161 `unsigned int optix::GeometryInstanceObj::getVariableCount () const`
[inline, virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2658 of file `optixpp_namespace.h`.

1.2.3.162 `unsigned int optix::GeometryObj::getVariableCount () const` [inline, virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2751 of file `optixpp_namespace.h`.

1.2.3.163 `unsigned int optix::MaterialObj::getVariableCount () const` [`inline`, `virtual`]

Query the number of variables associated with this object. Used along with [ScopedObj::getVariable](#) to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`. Implements [optix::ScopedObj](#).

Definition at line 2844 of file [optixpp_namespace.h](#).

1.2.3.164 `Program optix::SelectorObj::getVisitProgram () const` [`inline`]

Get the visitor program for this selector. See `rtSelectorGetVisitProgram`.

Definition at line 2264 of file [optixpp_namespace.h](#).

1.2.3.165 `RTwrapmode optix::TextureSamplerObj::getWrapMode (unsigned int dim) const` [`inline`]

Query the texture wrap mode for this sampler. See `rtTextureSamplerGetWrapMode`.

Definition at line 2911 of file [optixpp_namespace.h](#).

1.2.3.166 `bool optix::AccelerationObj::isDirty () const` [`inline`]

Query if the acceleration needs a rebuild. See `rtAccelerationIsDirty`.

Definition at line 2510 of file [optixpp_namespace.h](#).

1.2.3.167 `bool optix::GeometryObj::isDirty () const` [`inline`]

Query whether this geometry has been marked dirty. See `rtGeometryIsDirty`.

Definition at line 2770 of file [optixpp_namespace.h](#).

1.2.3.168 `void optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width)` [`inline`]

See `rtContextLaunch1D`.

Definition at line 2066 of file [optixpp_namespace.h](#).

1.2.3.169 `void optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width, RTsize image_height)` [`inline`]

See `rtContextLaunch2D`.

Definition at line 2071 of file [optixpp_namespace.h](#).

1.2.3.170 `void optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width, RTsize image_height, RTsize image_depth)` [`inline`]

See `rtContextLaunch3D`.

Definition at line 2076 of file [optixpp_namespace.h](#).

1.2.3.171 Exception `optix::Exception::makeException (RResult code, RTcontext context)` [`inline, static`]

Helper for creating exceptions from an RResult code origination from an OptiX C API function call.

Definition at line 262 of file [optixpp_namespace.h](#).

1.2.3.172 Exception `optix::APIObj::makeException (RResult code, RTcontext context)` [`inline, static`]

For backwards compatability. Use [Exception::makeException](#) instead.

Definition at line 317 of file [optixpp_namespace.h](#).

1.2.3.173 `void * optix::BufferObj::map ()` [`inline`]

Maps a buffer object for host access. See [rtBufferMap](#).

Definition at line 3209 of file [optixpp_namespace.h](#).

1.2.3.174 `void optix::AccelerationObj::markDirty ()` [`inline`]

Mark the acceleration as needing a rebuild. See [rtAccelerationMarkDirty](#).

Definition at line 2505 of file [optixpp_namespace.h](#).

1.2.3.175 `void optix::GeometryObj::markDirty ()` [`inline`]

Mark this geometry as dirty, causing rebuild of parent groups acceleration. See [rtGeometryMarkDirty](#).

Definition at line 2765 of file [optixpp_namespace.h](#).

1.2.3.176 `void optix::BufferObj::markDirty ()` [`inline`]

Mark the buffer dirty.

Definition at line 3085 of file [optixpp_namespace.h](#).

1.2.3.177 `template<class T > Handle< VariableObj > optix::Handle< T >::operator[] (const std::string & varname)`

Variable access operator. This operator will query the API object for a variable with the given name, creating a new variable instance if necessary. Only valid for [ScopedObjs](#).

Definition at line 598 of file [optixpp_namespace.h](#).

1.2.3.178 `template<class T > Handle< VariableObj > optix::Handle< T >::operator[] (const char * varname)`

Variable access operator. Identical to [operator\[\]\(const std::string& varname\)](#)

Explicitly define `char*` version to avoid ambiguities between builtin `operator[](int, char*)` and `Handle::operator[](std::string)`. The problem lies in that a [Handle](#) can be cast to a `bool` then to an `int` which implies that:


```
Context context;
context["var"];
```

can be interpreted as either

```
l["var"]; // Strange but legal way to index into a string (same as "var"[1]
)
```

or

```
context[ std::string("var") ];
```

Definition at line 607 of file [optixpp_namespace.h](#).

1.2.3.179 Variable `optix::ContextObj::queryVariable (const std::string & name) const` `[inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2132 of file [optixpp_namespace.h](#).

1.2.3.180 Variable `optix::ProgramObj::queryVariable (const std::string & name) const` `[inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2190 of file [optixpp_namespace.h](#).

1.2.3.181 Variable `optix::SelectorObj::queryVariable (const std::string & name) const` `[inline]`

Definition at line 2304 of file [optixpp_namespace.h](#).

1.2.3.182 Variable `optix::GeometryInstanceObj::queryVariable (const std::string & name) const` `[inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2646 of file [optixpp_namespace.h](#).

1.2.3.183 Variable `optix::GeometryObj::queryVariable (const std::string & name) const` `[inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2739 of file [optixpp_namespace.h](#).

1.2.3.184 Variable `optix::MaterialObj::queryVariable (const std::string & name) const`
[inline, virtual]

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2832 of file [optixpp_namespace.h](#).

1.2.3.185 `void optix::BufferObj::registerGLBuffer ()` [inline]

Declare the buffer as mutable and inaccessible by OptiX. See `rtTextureSamplerGLRegister`.

Definition at line 3144 of file [optixpp_namespace.h](#).

1.2.3.186 `void optix::TextureSamplerObj::registerGLTexture ()` [inline]

Declare the texture's buffer as mutable and inaccessible by OptiX. See `rtTextureSamplerGLRegister`.

Definition at line 2988 of file [optixpp_namespace.h](#).

1.2.3.187 `void optix::ContextObj::removeVariable (Variable v)` [inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2139 of file [optixpp_namespace.h](#).

1.2.3.188 `void optix::ProgramObj::removeVariable (Variable v)` [inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2197 of file [optixpp_namespace.h](#).

1.2.3.189 `void optix::SelectorObj::removeVariable (Variable v)` [inline]

Definition at line 2311 of file [optixpp_namespace.h](#).

1.2.3.190 `void optix::GeometryInstanceObj::removeVariable (Variable v)`
[inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2653 of file [optixpp_namespace.h](#).

1.2.3.191 `void optix::GeometryObj::removeVariable (Variable v)` [inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2746 of file [optixpp_namespace.h](#).

1.2.3.192 `void optix::MaterialObj::removeVariable (Variable v)` [inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2839 of file [optixpp_namespace.h](#).

1.2.3.193 `void optix::VariableObj::set (Buffer buffer)` [inline]

Definition at line 3471 of file [optixpp_namespace.h](#).

1.2.3.194 `void optix::VariableObj::set1fv (const float * f)` [inline]

Set variable value to a scalar float.

Definition at line 3369 of file [optixpp_namespace.h](#).

1.2.3.195 `void optix::VariableObj::set1iv (const int * i)` [inline]

Definition at line 3425 of file [optixpp_namespace.h](#).

1.2.3.196 `void optix::VariableObj::set1uiv (const unsigned int * u)` [inline]

Definition at line 3269 of file [optixpp_namespace.h](#).

1.2.3.197 `void optix::VariableObj::set2fv (const float * f)` [inline]

Set variable value to a float2.

Definition at line 3374 of file [optixpp_namespace.h](#).

1.2.3.198 `void optix::VariableObj::set2iv (const int * i)` [inline]

Definition at line 3430 of file [optixpp_namespace.h](#).

1.2.3.199 `void optix::VariableObj::set2uiv (const unsigned int * u)` [inline]

Definition at line 3274 of file [optixpp_namespace.h](#).

1.2.3.200 `void optix::VariableObj::set3fv (const float * f)` [inline]

Set variable value to a float3.

Definition at line 3379 of file [optixpp_namespace.h](#).

1.2.3.201 `void optix::VariableObj::set3iv (const int * i) [inline]`

Definition at line 3435 of file [optixpp_namespace.h](#).

1.2.3.202 `void optix::VariableObj::set3uiv (const unsigned int * u) [inline]`

Definition at line 3279 of file [optixpp_namespace.h](#).

1.2.3.203 `void optix::VariableObj::set4fv (const float * f) [inline]`

Set variable value to a float4.

Definition at line 3384 of file [optixpp_namespace.h](#).

1.2.3.204 `void optix::VariableObj::set4iv (const int * i) [inline]`

Definition at line 3440 of file [optixpp_namespace.h](#).

1.2.3.205 `void optix::VariableObj::set4uiv (const unsigned int * u) [inline]`

Definition at line 3284 of file [optixpp_namespace.h](#).

1.2.3.206 `void optix::GroupObj::setAcceleration (Acceleration acceleration) [inline]`

Set the Acceleration structure for this group. See `rtGroupSetAcceleration`.

Definition at line 2335 of file [optixpp_namespace.h](#).

1.2.3.207 `void optix::GeometryGroupObj::setAcceleration (Acceleration acceleration) [inline]`

Set the Acceleration structure for this group. See `rtGeometryGroupSetAcceleration`.

Definition at line 2397 of file [optixpp_namespace.h](#).

1.2.3.208 `void optix::MaterialObj::setAnyHitProgram (unsigned int ray_type_index, Program program) [inline]`

Set any hit program for this material at the given `ray_type` index. See `rtMaterialSetAnyHitProgram`.

Definition at line 2813 of file [optixpp_namespace.h](#).

1.2.3.209 `void optix::TextureSamplerObj::setArraySize (unsigned int num_textures_in_array) [inline]`

Set the texture array size for this sampler. See `rtTextureSamplerSetArraySize`.

Definition at line 2894 of file [optixpp_namespace.h](#).

1.2.3.210 `void optix::GeometryObj::setBoundingBoxProgram (Program program)`
[inline]

Set the bounding box program for this geometry. See `rtGeometrySetBoundingBoxProgram`.

Definition at line 2708 of file `optixpp_namespace.h`.

1.2.3.211 `void optix::VariableObj::setBuffer (Buffer buffer)` [inline]

Definition at line 3466 of file `optixpp_namespace.h`.

1.2.3.212 `void optix::TextureSamplerObj::setBuffer (unsigned int texture_array_idx,`
`unsigned int mip_level, Buffer buffer)` [inline]

Set the underlying buffer used for texture storage. `rtTextureSamplerSetBuffer`.

Definition at line 2971 of file `optixpp_namespace.h`.

1.2.3.213 `void optix::AccelerationObj::setBuilder (const std::string & builder)`
[inline]

Specify the acceleration structure builder. See `rtAccelerationSetBuilder`.

Definition at line 2529 of file `optixpp_namespace.h`.

1.2.3.214 `template<typename T> void optix::GroupObj::setChild (unsigned int index, T`
`child)` [inline]

Set an indexed child within this group. See `rtGroupSetChild`.

Definition at line 2360 of file `optixpp_namespace.h`.

1.2.3.215 `void optix::GeometryGroupObj::setChild (unsigned int index,`
`GeometryInstance geometryinstance)` [inline]

Set an indexed `GeometryInstance` child of this group. See `rtGeometryGroupSetChild`.

Definition at line 2421 of file `optixpp_namespace.h`.

1.2.3.216 `template<typename T> void optix::TransformObj::setChild (T child)`
[inline]

Set the child node of this transform. See `rtTransformSetChild`.

Definition at line 2458 of file `optixpp_namespace.h`.

1.2.3.217 `template<typename T> void optix::SelectorObj::setChild (unsigned int index,`
`T child)` [inline]

Set an indexed child child of this group. See `rtSelectorSetChild`.

Definition at line 2284 of file `optixpp_namespace.h`.

1.2.3.218 `void optix::GroupObj::setChildCount (unsigned int count) [inline]`

Set the number of children for this group. See `rtGroupSetChildCount`.

Definition at line 2347 of file `optixpp_namespace.h`.

1.2.3.219 `void optix::GeometryGroupObj::setChildCount (unsigned int count) [inline]`

Set the number of children for this group. See `rtGeometryGroupSetChildCount`.

Definition at line 2409 of file `optixpp_namespace.h`.

1.2.3.220 `void optix::SelectorObj::setChildCount (unsigned int count) [inline]`

Set the number of children for this group. See `rtSelectorSetChildCount`.

Definition at line 2271 of file `optixpp_namespace.h`.

1.2.3.221 `void optix::MaterialObj::setClosestHitProgram (unsigned int ray_type_index, Program program) [inline]`

Set closest hit program for this material at the given `ray_type` index. See `rtMaterialSetClosestHitProgram`.

Definition at line 2801 of file `optixpp_namespace.h`.

1.2.3.222 `void optix::ContextObj::setCPUNumThreads (int cpu_num_threads) [inline]`

See `rtContextSetAttribute`

Definition at line 1958 of file `optixpp_namespace.h`.

1.2.3.223 `void optix::AccelerationObj::setData (const void * data, RTsize size) [inline]`

Specify the acceleration structure via marshalled acceleration data. See `rtAccelerationSetData`.

Definition at line 2565 of file `optixpp_namespace.h`.

1.2.3.224 `void optix::BufferObj::setDevicePointer (unsigned int optix_device_number, CUdeviceptr device_pointer) [inline]`

Set the pointer to buffer memory on a specific device. See `rtBufferSetDevicePointer`.

Definition at line 3080 of file `optixpp_namespace.h`.

1.2.3.225 `template<class Iterator > void optix::ContextObj::setDevices (Iterator begin, Iterator end) [inline]`

See `rtContextSetDevices`

Definition at line 1892 of file `optixpp_namespace.h`.

1.2.3.226 `void optix::BufferObj::setElementSize (RTsize size_of_element)`
[inline]

Set the data element size for user format buffers. See `rtBufferSetElementSize`.

Definition at line 3063 of file `optixpp_namespace.h`.

1.2.3.227 `void optix::ContextObj::setEntryPointCount (unsigned int num_entry_points)`
[inline]

See `rtContextSetEntryPointCount`.

Definition at line 1985 of file `optixpp_namespace.h`.

1.2.3.228 `void optix::ContextObj::setExceptionEnabled (RTextception exception, bool enabled)`
[inline]

See `rtContextSetExceptionEnabled`.

Definition at line 2024 of file `optixpp_namespace.h`.

1.2.3.229 `void optix::ContextObj::setExceptionProgram (unsigned int entry_point_index, Program program)`
[inline]

See `rtContextSetExceptionProgram`.

Definition at line 2011 of file `optixpp_namespace.h`.

1.2.3.230 `void optix::TextureSamplerObj::setFilteringModes (RTfiltermode minification, RTfiltermode magnification, RTfiltermode mipmapping)`
[inline]

Set filtering modes for this sampler. See `rtTextureSamplerSetFilteringModes`.

Definition at line 2918 of file `optixpp_namespace.h`.

1.2.3.231 `void optix::VariableObj::setFloat (float f1)` [inline]

Set variable value to a scalar float.

Definition at line 3334 of file `optixpp_namespace.h`.

1.2.3.232 `void optix::VariableObj::setFloat (optix::float2 f)` [inline]

Set variable value to a float2.

Definition at line 3339 of file `optixpp_namespace.h`.

1.2.3.233 `void optix::VariableObj::setFloat (float f1, float f2)` [inline]

Set variable value to a float2.

Definition at line 3344 of file `optixpp_namespace.h`.

1.2.3.234 `void optix::VariableObj::setFloat (optix::float3 f)` [inline]

Set variable value to a float3.

Definition at line 3349 of file [optixpp_namespace.h](#).

1.2.3.235 `void optix::VariableObj::setFloat (float f1, float f2, float f3) [inline]`

Set variable value to a float3.

Definition at line 3354 of file [optixpp_namespace.h](#).

1.2.3.236 `void optix::VariableObj::setFloat (optix::float4 f) [inline]`

Set variable value to a float4.

Definition at line 3359 of file [optixpp_namespace.h](#).

1.2.3.237 `void optix::VariableObj::setFloat (float f1, float f2, float f3, float f4)
[inline]`

Set variable value to a float4.

Definition at line 3364 of file [optixpp_namespace.h](#).

1.2.3.238 `void optix::BufferObj::setFormat (RTformat format) [inline]`

Set the data format for the buffer. See `rtBufferSetFormat`.

Definition at line 3051 of file [optixpp_namespace.h](#).

1.2.3.239 `void optix::GeometryInstanceObj::setGeometry (Geometry geometry)
[inline]`

Set the geometry object associated with this instance. See `rtGeometryInstanceSetGeometry`.

Definition at line 2594 of file [optixpp_namespace.h](#).

1.2.3.240 `void optix::ContextObj::setGPUPagingForcedOff (int gpu_paging_forced_off
) [inline]`

See `rtContextSetAttribute`.

Definition at line 1963 of file [optixpp_namespace.h](#).

1.2.3.241 `void optix::TextureSamplerObj::setIndexingMode (RTtextureindexmode
indexmode) [inline]`

Set texture indexing mode for this sampler. See `rtTextureSamplerSetIndexingMode`.

Definition at line 2959 of file [optixpp_namespace.h](#).

1.2.3.242 `void optix::VariableObj::setInt (int i1) [inline]`

Definition at line 3390 of file [optixpp_namespace.h](#).

1.2.3.243 `void optix::VariableObj::setInt (int i1, int i2) [inline]`

Definition at line 3400 of file [optixpp_namespace.h](#).

1.2.3.244 `void optix::VariableObj::setInt (optix::int2 i) [inline]`

Definition at line 3395 of file [optixpp_namespace.h](#).

1.2.3.245 `void optix::VariableObj::setInt (int i1, int i2, int i3) [inline]`

Definition at line 3410 of file [optixpp_namespace.h](#).

1.2.3.246 `void optix::VariableObj::setInt (optix::int3 i) [inline]`

Definition at line 3405 of file [optixpp_namespace.h](#).

1.2.3.247 `void optix::VariableObj::setInt (int i1, int i2, int i3, int i4) [inline]`

Definition at line 3420 of file [optixpp_namespace.h](#).

1.2.3.248 `void optix::VariableObj::setInt (optix::int4 i) [inline]`

Definition at line 3415 of file [optixpp_namespace.h](#).

1.2.3.249 `void optix::GeometryObj::setIntersectionProgram (Program program) [inline]`

Set the intersection program for this geometry. See `rtGeometrySetIntersectionProgram`.

Definition at line 2720 of file [optixpp_namespace.h](#).

1.2.3.250 `void optix::GeometryInstanceObj::setMaterial (unsigned int idx, Material material) [inline]`

Set the material at given index. See `rtGeometryInstanceSetMaterial`.

Definition at line 2618 of file [optixpp_namespace.h](#).

1.2.3.251 `void optix::GeometryInstanceObj::setMaterialCount (unsigned int count) [inline]`

Set the number of materials associated with this instance. See `rtGeometryInstanceSetMaterialCount`.

Definition at line 2606 of file [optixpp_namespace.h](#).

1.2.3.252 `void optix::TransformObj::setMatrix (bool transpose, const float * matrix, const float * inverse_matrix) [inline]`

Set the transform matrix for this node. See `rtTransformSetMatrix`.

Definition at line 2471 of file [optixpp_namespace.h](#).

1.2.3.253 `void optix::VariableObj::setMatrix2x2fv (bool transpose, const float * m) [inline]`

Definition at line 3289 of file [optixpp_namespace.h](#).

1.2.3.254 `void optix::VariableObj::setMatrix2x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3294 of file [optixpp_namespace.h](#).

1.2.3.255 `void optix::VariableObj::setMatrix2x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3299 of file [optixpp_namespace.h](#).

1.2.3.256 `void optix::VariableObj::setMatrix3x2fv (bool transpose, const float * m)`
[inline]

Definition at line 3304 of file [optixpp_namespace.h](#).

1.2.3.257 `void optix::VariableObj::setMatrix3x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3309 of file [optixpp_namespace.h](#).

1.2.3.258 `void optix::VariableObj::setMatrix3x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3314 of file [optixpp_namespace.h](#).

1.2.3.259 `void optix::VariableObj::setMatrix4x2fv (bool transpose, const float * m)`
[inline]

Definition at line 3319 of file [optixpp_namespace.h](#).

1.2.3.260 `void optix::VariableObj::setMatrix4x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3324 of file [optixpp_namespace.h](#).

1.2.3.261 `void optix::VariableObj::setMatrix4x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3329 of file [optixpp_namespace.h](#).

1.2.3.262 `void optix::TextureSamplerObj::setMaxAnisotropy (float value)`
[inline]

Set maximum anisotropy for this sampler. See `rtTextureSamplerSetMaxAnisotropy`.

Definition at line 2928 of file [optixpp_namespace.h](#).

1.2.3.263 `void optix::TextureSamplerObj::setMipLevelCount (unsigned int num_mip_levels)` [inline]

Set the number of mip levels for this sampler. See `rtTextureSamplerSetMipLevelCount`.

Definition at line 2882 of file [optixpp_namespace.h](#).

1.2.3.264 `void optix::ContextObj::setMissProgram (unsigned int ray_type_index,
Program program) [inline]`

See `rtContextSetMissProgram`.

Definition at line 2049 of file `optixpp_namespace.h`.

1.2.3.265 `void optix::GeometryObj::setPrimitiveCount (unsigned int num_primitives)
[inline]`

Set the number of primitives in this geometry objects (eg, number of triangles in mesh).

See `rtGeometrySetPrimitiveCount`

Definition at line 2696 of file `optixpp_namespace.h`.

1.2.3.266 `void optix::ContextObj::setPrintBufferSize (RTsize buffer_size_bytes)
[inline]`

See `rtContextSetPrintBufferSize`.

Definition at line 2101 of file `optixpp_namespace.h`.

1.2.3.267 `void optix::ContextObj::setPrintEnabled (bool enabled) [inline]`

See `rtContextSetPrintEnabled`

Definition at line 2089 of file `optixpp_namespace.h`.

1.2.3.268 `void optix::ContextObj::setPrintLaunchIndex (int x, int y = -1, int z = -1)
[inline]`

See `rtContextSetPrintLaunchIndex`.

Definition at line 2113 of file `optixpp_namespace.h`.

1.2.3.269 `void optix::AccelerationObj::setProperty (const std::string & name, const
std::string & value) [inline]`

Set properties specifying Acceleration builder/traverser behavior. See `rtAcceleration-SetProperty`.

Definition at line 2517 of file `optixpp_namespace.h`.

1.2.3.270 `void optix::ContextObj::setRayGenerationProgram (unsigned int
entry_point_index, Program program) [inline]`

See `rtContextSetRayGenerationProgram`

Definition at line 1998 of file `optixpp_namespace.h`.

1.2.3.271 `void optix::ContextObj::setRayTypeCount (unsigned int num_ray_types)
[inline]`

See `rtContextSetRayTypeCount`.

Definition at line 2037 of file `optixpp_namespace.h`.

1.2.3.272 `void optix::TextureSamplerObj::setReadMode (RTtexturereadmode readmode) [inline]`

Set texture read mode for this sampler. See `rtTextureSamplerSetReadMode`.

Definition at line 2947 of file `optixpp_namespace.h`.

1.2.3.273 `void optix::BufferObj::setSize (RTsize width) [inline]`

Set buffer dimensionality to one and buffer width to specified width. See `rtBufferSetSize1D`.

Definition at line 3090 of file `optixpp_namespace.h`.

1.2.3.274 `void optix::BufferObj::setSize (RTsize width, RTsize height) [inline]`

Set buffer dimensionality to two and buffer dimensions to specified width,height. See `rtBufferSetSize2D`.

Definition at line 3100 of file `optixpp_namespace.h`.

1.2.3.275 `void optix::BufferObj::setSize (RTsize width, RTsize height, RTsize depth) [inline]`

Set buffer dimensionality to three and buffer dimensions to specified width,height,depth. See `rtBufferSetSize3D`.

Definition at line 3110 of file `optixpp_namespace.h`.

1.2.3.276 `void optix::BufferObj::setSize (unsigned int dimensionality, const RTsize * dims) [inline]`

Set buffer dimensionality and dimensions to specified values. See `rtBufferSetSize`.

Definition at line 3120 of file `optixpp_namespace.h`.

1.2.3.277 `void optix::ContextObj::setStackSize (RTsize stack_size_bytes) [inline]`

See `rtContextSetStackSize`

Definition at line 1968 of file `optixpp_namespace.h`.

1.2.3.278 `void optix::VariableObj::setTextureSampler (TextureSampler texturesample) [inline]`

Definition at line 3486 of file `optixpp_namespace.h`.

1.2.3.279 `void optix::ContextObj::setTimeoutCallback (RTtimeoutcallback callback, double min_polling_seconds) [inline]`

See `rtContextSetTimeoutCallback` `RTtimeoutcallback` is defined as `typedef int (*RTtimeoutcallback)(void)`.

Definition at line 1980 of file `optixpp_namespace.h`.

1.2.3.280 `void optix::AccelerationObj::setTraverser (const std::string & traverser)`
[inline]

Specify the acceleration structure traverser. See `rtAccelerationSetTraverser`.

Definition at line 2541 of file `optixpp_namespace.h`.

1.2.3.281 `void optix::VariableObj::setUInt (unsigned int u1)` [inline]

Definition at line 3234 of file `optixpp_namespace.h`.

1.2.3.282 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2)`
[inline]

Definition at line 3239 of file `optixpp_namespace.h`.

1.2.3.283 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2, unsigned int u3)` [inline]

Definition at line 3244 of file `optixpp_namespace.h`.

1.2.3.284 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2, unsigned int u3, unsigned int u4)` [inline]

Definition at line 3249 of file `optixpp_namespace.h`.

1.2.3.285 `void optix::VariableObj::setUInt (optix::uint2 u)` [inline]

Definition at line 3254 of file `optixpp_namespace.h`.

1.2.3.286 `void optix::VariableObj::setUInt (optix::uint3 u)` [inline]

Definition at line 3259 of file `optixpp_namespace.h`.

1.2.3.287 `void optix::VariableObj::setUInt (optix::uint4 u)` [inline]

Definition at line 3264 of file `optixpp_namespace.h`.

1.2.3.288 `void optix::VariableObj::setUserData (RTsize size, const void * ptr)`
[inline]

Set the variable to a user defined type given the sizeof the user object.

Definition at line 3476 of file `optixpp_namespace.h`.

1.2.3.289 `void optix::SelectorObj::setVisitProgram (Program program)`
[inline]

Set the visitor program for this selector. See `rtSelectorSetVisitProgram`

Definition at line 2259 of file `optixpp_namespace.h`.

1.2.3.290 `void optix::TextureSamplerObj::setWrapMode (unsigned int dim,
RTwrapmode wrapmode) [inline]`

Set the texture wrap mode for this sampler. See `rtTextureSamplerSetWrapMode`.

Definition at line 2906 of file `optixpp_namespace.h`.

1.2.3.291 `void optix::BufferObj::unmap () [inline]`

Unmaps a buffer object. See `rtBufferUnmap`.

Definition at line 3216 of file `optixpp_namespace.h`.

1.2.3.292 `void optix::BufferObj::unregisterGLBuffer () [inline]`

Unregister the buffer, re-enabling OptiX operations. See `rtTextureSamplerGLUnregister`.

Definition at line 3149 of file `optixpp_namespace.h`.

1.2.3.293 `void optix::TextureSamplerObj::unregisterGLTexture () [inline]`

Unregister the texture's buffer, re-enabling OptiX operations. See `rtTextureSamplerGLUnregister`.

Definition at line 2993 of file `optixpp_namespace.h`.

1.2.3.294 `void optix::ContextObj::validate () [inline, virtual]`

See `rtContextValidate`.

Implements `optix::DestroyableObj`.

Definition at line 1597 of file `optixpp_namespace.h`.

1.2.3.295 `void optix::ProgramObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2171 of file `optixpp_namespace.h`.

1.2.3.296 `void optix::GroupObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2228 of file `optixpp_namespace.h`.

1.2.3.297 `void optix::GeometryGroupObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2385 of file `optixpp_namespace.h`.

1.2.3.298 void `optix::TransformObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2445 of file `optixpp_namespace.h`.

1.2.3.299 void `optix::SelectorObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2247 of file `optixpp_namespace.h`.

1.2.3.300 void `optix::AccelerationObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2493 of file `optixpp_namespace.h`.

1.2.3.301 void `optix::GeometryInstanceObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2582 of file `optixpp_namespace.h`.

1.2.3.302 void `optix::GeometryObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2684 of file `optixpp_namespace.h`.

1.2.3.303 void `optix::MaterialObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2789 of file `optixpp_namespace.h`.

1.2.3.304 void `optix::TextureSamplerObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2870 of file `optixpp_namespace.h`.

1.2.3.305 void `optix::BufferObj::validate` () [`inline, virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

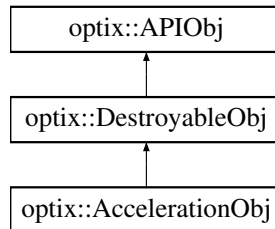
Definition at line [3039](#) of file [optixpp_namespace.h](#).

2 Class Documentation

2.1 optix::AccelerationObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for optix::AccelerationObj:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTacceleration](#) [get](#) ()

Friends

- class [Handle](#)< [AccelerationObj](#) >
- void [markDirty](#) ()
- bool [isDirty](#) () const
- void [setProperty](#) (const std::string &name, const std::string &value)
- std::string [getProperty](#) (const std::string &name) const
- void [setBuilder](#) (const std::string &builder)
- std::string [getBuilder](#) () const
- void [setTraverser](#) (const std::string &traverser)
- std::string [getTraverser](#) () const
- [RTsize](#) [getDataSize](#) () const
- void [getData](#) (void *data) const
- void [setData](#) (const void *data, [RTsize](#) size)

2.1.1 Detailed Description

Acceleration wraps the OptiX C API [RTacceleration](#) opaque type and its associated function set.

Definition at line [1101](#) of file [optixpp_namespace.h](#).

2.1.2 Member Function Documentation

2.1.2.1 `void optix::AccelerationObj::destroy ()` `[inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2486 of file `optixpp_namespace.h`.

2.1.2.2 `RTacceleration optix::AccelerationObj::get ()` `[inline]`

Get the underlying OptiX C API `RTacceleration` opaque pointer.

Definition at line 2570 of file `optixpp_namespace.h`.

2.1.2.3 `std::string optix::AccelerationObj::getBuilder () const` `[inline]`

Query the acceleration structure builder. See `rtAccelerationGetBuilder`.

Definition at line 2534 of file `optixpp_namespace.h`.

2.1.2.4 `Context optix::AccelerationObj::getContext () const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2498 of file `optixpp_namespace.h`.

2.1.2.5 `void optix::AccelerationObj::getData (void * data) const` `[inline]`

Get the marshalled acceleration data. See `rtAccelerationGetData`.

Definition at line 2560 of file `optixpp_namespace.h`.

2.1.2.6 `RTsize optix::AccelerationObj::getDataSize () const` `[inline]`

Query the size of the marshalled acceleration data. See `rtAccelerationGetDataSize`.

Definition at line 2553 of file `optixpp_namespace.h`.

2.1.2.7 `std::string optix::AccelerationObj::getProperty (const std::string & name) const` `[inline]`

Query properties specifying Acceleration builder/traverser behavior. See `rtAccelerationGetProperty`.

Definition at line 2522 of file `optixpp_namespace.h`.

2.1.2.8 `std::string optix::AccelerationObj::getTraverser () const` `[inline]`

Query the acceleration structure traverser. See `rtAccelerationGetTraverser`.

Definition at line 2546 of file `optixpp_namespace.h`.

2.1.2.9 `bool optix::AccelerationObj::isDirty() const` `[inline]`

Query if the acceleration needs a rebuild. See `rtAccelerationIsDirty`.

Definition at line 2510 of file `optixpp_namespace.h`.

2.1.2.10 `void optix::AccelerationObj::markDirty()` `[inline]`

Mark the acceleration as needing a rebuild. See `rtAccelerationMarkDirty`.

Definition at line 2505 of file `optixpp_namespace.h`.

2.1.2.11 `void optix::AccelerationObj::setBuilder(const std::string & builder)`
`[inline]`

Specify the acceleration structure builder. See `rtAccelerationSetBuilder`.

Definition at line 2529 of file `optixpp_namespace.h`.

2.1.2.12 `void optix::AccelerationObj::setData(const void * data, RTsize size)`
`[inline]`

Specify the acceleration structure via marshalled acceleration data. See `rtAccelerationSetData`.

Definition at line 2565 of file `optixpp_namespace.h`.

2.1.2.13 `void optix::AccelerationObj::setProperty(const std::string & name, const std::string & value)` `[inline]`

Set properties specifying Acceleration builder/traverser behavior. See `rtAccelerationSetProperty`.

Definition at line 2517 of file `optixpp_namespace.h`.

2.1.2.14 `void optix::AccelerationObj::setTraverser(const std::string & traverser)`
`[inline]`

Specify the acceleration structure traverser. See `rtAccelerationSetTraverser`.

Definition at line 2541 of file `optixpp_namespace.h`.

2.1.2.15 `void optix::AccelerationObj::validate()` `[inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2493 of file `optixpp_namespace.h`.

2.1.3 Friends And Related Function Documentation

2.1.3.1 `friend class Handle< AccelerationObj >` `[friend]`

Definition at line 1149 of file `optixpp_namespace.h`.

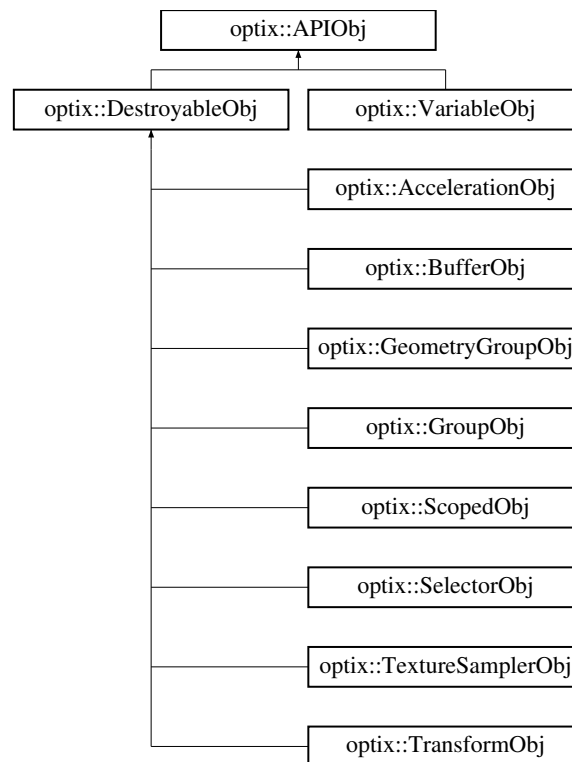
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.2 optix::APIObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for optix::APIObj:



Public Member Functions

- [APIObj \(\)](#)
- virtual [~APIObj \(\)](#)
- void [addReference \(\)](#)
- int [removeReference \(\)](#)
- virtual [Context getContext \(\) const =0](#)
- virtual void [checkError \(RTresult code\) const](#)
- virtual void [checkError \(RTresult code, Context context\) const](#)
- void [checkErrorNoGetContext \(RTresult code\) const](#)

Static Public Member Functions

- static [Exception](#) `makeException` (RTresult code, RTcontext context)

2.2.1 Detailed Description

Base class for all reference counted wrappers around OptiX C API opaque types.

Wraps:

- RTcontext
- RTbuffer
- RTgeometry
- RTgeometryinstance
- RTgeometrygroup
- RTgroup
- RTmaterial
- RTprogram
- RTselector
- RTtexturesampler
- RTtransform
- RTvariable

Definition at line 291 of file [optixpp_namespace.h](#).

2.2.2 Constructor & Destructor Documentation

2.2.2.1 `optix::APIObj::APIObj()` [inline]

Definition at line 293 of file [optixpp_namespace.h](#).

2.2.2.2 `virtual optix::APIObj::~~APIObj()` [inline, virtual]

Definition at line 294 of file [optixpp_namespace.h](#).

2.2.3 Member Function Documentation

2.2.3.1 `void optix::APIObj::addReference()` [inline]

Increment the reference count for this object.

Definition at line 297 of file [optixpp_namespace.h](#).

2.2.3.2 `void optix::APIObj::checkError (RResult code) const` [`inline`, `virtual`]

Check the given result code and throw an error with appropriate message if the code is not `RTsuccess`

Reimplemented in [optix::ContextObj](#).

Definition at line 1523 of file [optixpp_namespace.h](#).

2.2.3.3 `void optix::APIObj::checkError (RResult code, Context context) const` [`inline`, `virtual`]

Definition at line 1531 of file [optixpp_namespace.h](#).

2.2.3.4 `void optix::APIObj::checkErrorNoGetContext (RResult code) const` [`inline`]

Definition at line 1539 of file [optixpp_namespace.h](#).

2.2.3.5 `virtual Context optix::APIObj::getContext () const` [`pure virtual`]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implemented in [optix::BufferObj](#), [optix::TextureSamplerObj](#), [optix::MaterialObj](#), [optix::GeometryObj](#), [optix::GeometryInstanceObj](#), [optix::AccelerationObj](#), [optix::SelectorObj](#), [optix::TransformObj](#), [optix::GeometryGroupObj](#), [optix::GroupObj](#), [optix::ProgramObj](#), [optix::ContextObj](#), and [optix::VariableObj](#).

2.2.3.6 `Exception optix::APIObj::makeException (RResult code, RTcontext context)` [`inline`, `static`]

For backwards compatability. Use [Exception::makeException](#) instead.

Definition at line 317 of file [optixpp_namespace.h](#).

2.2.3.7 `int optix::APIObj::removeReference ()` [`inline`]

Decrement the reference count for this object.

Definition at line 299 of file [optixpp_namespace.h](#).

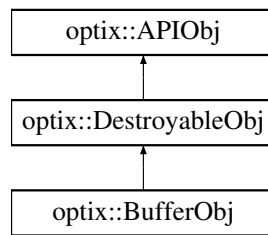
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.3 `optix::BufferObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::BufferObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTbuffer](#) [get](#) ()

Friends

- class [Handle](#)< [BufferObj](#) >
- void [setFormat](#) ([RTformat](#) format)
- [RTformat](#) [getFormat](#) () const
- void [setElementSize](#) ([RTsize](#) size_of_element)
- [RTsize](#) [getElementSize](#) () const
- void [getDevicePointer](#) (unsigned int optix_device_number, [CUdeviceptr](#) *device_pointer)
- void [setDevicePointer](#) (unsigned int optix_device_number, [CUdeviceptr](#) device_pointer)
- void [markDirty](#) ()
- void [setSize](#) ([RTsize](#) width)
- void [getSize](#) ([RTsize](#) &width) const
- void [setSize](#) ([RTsize](#) width, [RTsize](#) height)
- void [getSize](#) ([RTsize](#) &width, [RTsize](#) &height) const
- void [setSize](#) ([RTsize](#) width, [RTsize](#) height, [RTsize](#) depth)
- void [getSize](#) ([RTsize](#) &width, [RTsize](#) &height, [RTsize](#) &depth) const
- void [setSize](#) (unsigned int dimensionality, const [RTsize](#) *dims)
- void [getSize](#) (unsigned int dimensionality, [RTsize](#) *dims) const
- unsigned int [getDimensionality](#) () const
- unsigned int [getGLBOid](#) () const
- void [registerGLBuffer](#) ()
- void [unregisterGLBuffer](#) ()
- void * [map](#) ()
- void [unmap](#) ()

2.3.1 Detailed Description

Buffer wraps the OptiX C API `RTbuffer` opaque type and its associated function set.

Definition at line 1415 of file `optixpp_namespace.h`.

2.3.2 Member Function Documentation

2.3.2.1 `void optix::BufferObj::destroy() [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 3032 of file `optixpp_namespace.h`.

2.3.2.2 `RTbuffer optix::BufferObj::get() [inline]`

Get the underlying OptiX C API `RTbuffer` opaque pointer.

Definition at line 3222 of file `optixpp_namespace.h`.

2.3.2.3 `Context optix::BufferObj::getContext() const [inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 3044 of file `optixpp_namespace.h`.

2.3.2.4 `void optix::BufferObj::getDevicePointer(unsigned int optix_device_number, CUdeviceptr * device_pointer) [inline]`

Get the pointer to buffer memory on a specific device. See `rtBufferGetDevicePointer`.

Definition at line 3075 of file `optixpp_namespace.h`.

2.3.2.5 `unsigned int optix::BufferObj::getDimensionality() const [inline]`

Query dimensionality of buffer. See `rtBufferGetDimensionality`.

Definition at line 3130 of file `optixpp_namespace.h`.

2.3.2.6 `RTsize optix::BufferObj::getElementSize() const [inline]`

Query the data element size for user format buffers. See `rtBufferGetElementSize`.

Definition at line 3068 of file `optixpp_namespace.h`.

2.3.2.7 `RTformat optix::BufferObj::getFormat() const [inline]`

Query the data format for the buffer. See `rtBufferGetFormat`.

Definition at line 3056 of file `optixpp_namespace.h`.

2.3.2.8 `unsigned int optix::BufferObj::getGLBOid () const [inline]`

Queries the OpenGL Buffer Object ID associated with this buffer. See `rtBufferGetGLBOld`.

Definition at line 3137 of file `optixpp_namespace.h`.

2.3.2.9 `void optix::BufferObj::getSize (RTsize & width) const [inline]`

Query 1D buffer dimension. See `rtBufferGetSize1D`.

Definition at line 3095 of file `optixpp_namespace.h`.

2.3.2.10 `void optix::BufferObj::getSize (RTsize & width, RTsize & height) const [inline]`

Query 2D buffer dimension. See `rtBufferGetSize2D`.

Definition at line 3105 of file `optixpp_namespace.h`.

2.3.2.11 `void optix::BufferObj::getSize (RTsize & width, RTsize & height, RTsize & depth) const [inline]`

Query 3D buffer dimension. See `rtBufferGetSize3D`.

Definition at line 3115 of file `optixpp_namespace.h`.

2.3.2.12 `void optix::BufferObj::getSize (unsigned int dimensionality, RTsize * dims) const [inline]`

Query dimensions of buffer. See `rtBufferGetSizev`.

Definition at line 3125 of file `optixpp_namespace.h`.

2.3.2.13 `void * optix::BufferObj::map () [inline]`

Maps a buffer object for host access. See `rtBufferMap`.

Definition at line 3209 of file `optixpp_namespace.h`.

2.3.2.14 `void optix::BufferObj::markDirty () [inline]`

Mark the buffer dirty.

Definition at line 3085 of file `optixpp_namespace.h`.

2.3.2.15 `void optix::BufferObj::registerGLBuffer () [inline]`

Declare the buffer as mutable and inaccessible by OptiX. See `rtTextureSamplerGLRegister`.

Definition at line 3144 of file `optixpp_namespace.h`.

2.3.2.16 `void optix::BufferObj::setDevicePointer (unsigned int optix_device_number, CUdeviceptr device_pointer) [inline]`

Set the pointer to buffer memory on a specific device. See `rtBufferSetDevicePointer`.

Definition at line 3080 of file `optixpp_namespace.h`.

2.3.2.17 `void optix::BufferObj::setElementSize (RTsize size_of_element) [inline]`

Set the data element size for user format buffers. See `rtBufferSetElementSize`.

Definition at line 3063 of file `optixpp_namespace.h`.

2.3.2.18 `void optix::BufferObj::setFormat (RTformat format) [inline]`

Set the data format for the buffer. See `rtBufferSetFormat`.

Definition at line 3051 of file `optixpp_namespace.h`.

2.3.2.19 `void optix::BufferObj::setSize (RTsize width) [inline]`

Set buffer dimensionality to one and buffer width to specified width. See `rtBufferSetSize1D`.

Definition at line 3090 of file `optixpp_namespace.h`.

2.3.2.20 `void optix::BufferObj::setSize (RTsize width, RTsize height) [inline]`

Set buffer dimensionality to two and buffer dimensions to specified width,height. See `rtBufferSetSize2D`.

Definition at line 3100 of file `optixpp_namespace.h`.

2.3.2.21 `void optix::BufferObj::setSize (RTsize width, RTsize height, RTsize depth) [inline]`

Set buffer dimensionality to three and buffer dimensions to specified width,height,depth. See `rtBufferSetSize3D`.

Definition at line 3110 of file `optixpp_namespace.h`.

2.3.2.22 `void optix::BufferObj::setSize (unsigned int dimensionality, const RTsize * dims) [inline]`

Set buffer dimensionality and dimensions to specified values. See `rtBufferSetSizev`.

Definition at line 3120 of file `optixpp_namespace.h`.

2.3.2.23 `void optix::BufferObj::unmap () [inline]`

Unmaps a buffer object. See `rtBufferUnmap`.

Definition at line 3216 of file `optixpp_namespace.h`.

2.3.2.24 void `optix::BufferObj::unregisterGLBuffer ()` [inline]

Unregister the buffer, re-enabling OptiX operations. See `rtTextureSamplerGLUnregister`.

Definition at line 3149 of file `optixpp_namespace.h`.

2.3.2.25 void `optix::BufferObj::validate ()` [inline, virtual]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 3039 of file `optixpp_namespace.h`.

2.3.3 Friends And Related Function Documentation

2.3.3.1 friend class `Handle< BufferObj >` [friend]

Definition at line 1516 of file `optixpp_namespace.h`.

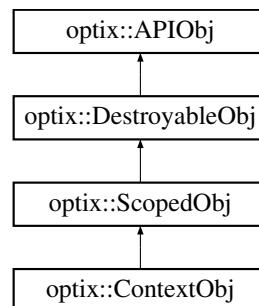
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.4 optix::ContextObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::ContextObj`:



Public Member Functions

- void `destroy ()`
- void `validate ()`
- `Context getContext ()` const
- void `compile ()`
- int `getRunningState ()` const
- `RTcontext get ()`

Static Public Member Functions

- static unsigned int [getDeviceCount](#) ()
- static std::string [getDeviceName](#) (int ordinal)
- static void [getDeviceAttribute](#) (int ordinal, RTdeviceattribute attrib, RTsize size, void *p)
- static [Context](#) [create](#) ()

Friends

- class [Handle](#)< [ContextObj](#) >
- void [checkError](#) (RTresult code) const
- std::string [getErrorString](#) (RTresult code) const
- [Acceleration](#) [createAcceleration](#) (const char *builder, const char *traverser)
- [Buffer](#) [createBuffer](#) (unsigned int type)
- [Buffer](#) [createBuffer](#) (unsigned int type, RTformat format)
- [Buffer](#) [createBuffer](#) (unsigned int type, RTformat format, RTsize width)
- [Buffer](#) [createBuffer](#) (unsigned int type, RTformat format, RTsize width, RTsize height)
- [Buffer](#) [createBuffer](#) (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)
- [Buffer](#) [createBufferForCUDA](#) (unsigned int type)
- [Buffer](#) [createBufferForCUDA](#) (unsigned int type, RTformat format)
- [Buffer](#) [createBufferForCUDA](#) (unsigned int type, RTformat format, RTsize width)
- [Buffer](#) [createBufferForCUDA](#) (unsigned int type, RTformat format, RTsize width, RTsize height)
- [Buffer](#) [createBufferForCUDA](#) (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)
- [Buffer](#) [createBufferFromGLBO](#) (unsigned int type, unsigned int vbo)
- [TextureSampler](#) [createTextureSamplerFromGLImage](#) (unsigned int id, RTgltarget target)
- [Geometry](#) [createGeometry](#) ()
- [GeometryInstance](#) [createGeometryInstance](#) ()
- template<class Iterator >
[GeometryInstance](#) [createGeometryInstance](#) ([Geometry](#) geometry, Iterator matlbegin, Iterator matlend)
- [Group](#) [createGroup](#) ()
- template<class Iterator >
[Group](#) [createGroup](#) (Iterator childbegin, Iterator childend)
- [GeometryGroup](#) [createGeometryGroup](#) ()
- template<class Iterator >
[GeometryGroup](#) [createGeometryGroup](#) (Iterator childbegin, Iterator childend)
- [Transform](#) [createTransform](#) ()
- [Material](#) [createMaterial](#) ()

- [Program](#) [createProgramFromPTXFile](#) (const std::string &ptx, const std::string &program_name)
- [Program](#) [createProgramFromPTXString](#) (const std::string &ptx, const std::string &program_name)
- [Selector](#) [createSelector](#) ()
- [TextureSampler](#) [createTextureSampler](#) ()

- template<class Iterator >
void [setDevices](#) (Iterator begin, Iterator end)
- std::vector< int > [getEnabledDevices](#) () const
- unsigned int [getEnabledDeviceCount](#) () const

- int [getMaxTextureCount](#) () const
- int [getCPUNumThreads](#) () const
- RTsize [getUsedHostMemory](#) () const
- int [getGPUPagingActive](#) () const
- int [getGPUPagingForcedOff](#) () const
- RTsize [getAvailableDeviceMemory](#) (int ordinal) const

- void [setCPUNumThreads](#) (int cpu_num_threads)
- void [setGPUPagingForcedOff](#) (int gpu_paging_forced_off)

- void [setStackSize](#) (RTsize stack_size_bytes)
- RTsize [getStackSize](#) () const
- void [setTimeoutCallback](#) (RTtimeoutcallback callback, double min_polling_seconds)
- void [setEntryPointCount](#) (unsigned int num_entry_points)
- unsigned int [getEntryPointCount](#) () const
- void [setRayTypeCount](#) (unsigned int num_ray_types)
- unsigned int [getRayTypeCount](#) () const

- void [setRayGenerationProgram](#) (unsigned int entry_point_index, [Program](#) program)
- [Program](#) [getRayGenerationProgram](#) (unsigned int entry_point_index) const
- void [setExceptionProgram](#) (unsigned int entry_point_index, [Program](#) program)
- [Program](#) [getExceptionProgram](#) (unsigned int entry_point_index) const
- void [setExceptionEnabled](#) (RTexception exception, bool enabled)
- bool [getExceptionEnabled](#) (RTexception exception) const
- void [setMissProgram](#) (unsigned int ray_type_index, [Program](#) program)
- [Program](#) [getMissProgram](#) (unsigned int ray_type_index) const

- void [launch](#) (unsigned int entry_point_index, RTsize image_width)
- void [launch](#) (unsigned int entry_point_index, RTsize image_width, RTsize image_height)
- void [launch](#) (unsigned int entry_point_index, RTsize image_width, RTsize image_height, RTsize image_depth)

- void [setPrintEnabled](#) (bool enabled)
- bool [getPrintEnabled](#) () const
- void [setPrintBufferSize](#) (RTsize buffer_size_bytes)
- RTsize [getPrintBufferSize](#) () const
- void [setPrintLaunchIndex](#) (int x, int y=-1, int z=-1)
- optix::int3 [getPrintLaunchIndex](#) () const

- [Variable declareVariable](#) (const std::string &name)
- [Variable queryVariable](#) (const std::string &name) const
- void [removeVariable](#) ([Variable](#) v)
- unsigned int [getVariableCount](#) () const
- [Variable getVariable](#) (unsigned int index) const

2.4.1 Detailed Description

Context object wraps the OptiX C API RTcontext opaque type and its associated function set.

Definition at line 619 of file [optixpp_namespace.h](#).

2.4.2 Member Function Documentation

2.4.2.1 void [optix::ContextObj::checkError](#) (RTresult *code*) const [[inline](#), [virtual](#)]

See [APIObj::checkError](#)

Reimplemented from [optix::APIObj](#).

Definition at line 1551 of file [optixpp_namespace.h](#).

2.4.2.2 void [optix::ContextObj::compile](#) () [[inline](#)]

See [rtContextCompile](#).

Definition at line 2061 of file [optixpp_namespace.h](#).

2.4.2.3 Context [optix::ContextObj::create](#) () [[inline](#), [static](#)]

Creates a Context object. See [rtContextCreate](#).

Definition at line 1582 of file [optixpp_namespace.h](#).

2.4.2.4 Acceleration [optix::ContextObj::createAcceleration](#) (const char * *builder*, const char * *traverser*) [[inline](#)]

See [rtAccelerationCreate](#)

Definition at line 1602 of file [optixpp_namespace.h](#).

2.4.2.5 Buffer `optix::ContextObj::createBuffer (unsigned int type)` `[inline]`

Create a buffer with given RTbuffertype. See `rtBufferCreate`.

Definition at line 1612 of file `optixpp_namespace.h`.

2.4.2.6 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format)` `[inline]`

Create a buffer with given RTbuffertype and RTformat. See `rtBufferCreate`, `rtBufferSetFormat`.

Definition at line 1619 of file `optixpp_namespace.h`.

2.4.2.7 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize1D`.

Definition at line 1627 of file `optixpp_namespace.h`.

2.4.2.8 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width, RTsize height)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize2D`.

Definition at line 1636 of file `optixpp_namespace.h`.

2.4.2.9 Buffer `optix::ContextObj::createBuffer (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)` `[inline]`

Create a buffer with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize3D`.

Definition at line 1645 of file `optixpp_namespace.h`.

2.4.2.10 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type)` `[inline]`

Create a buffer for CUDA with given RTbuffertype. See `rtBufferCreate`.

Definition at line 1654 of file `optixpp_namespace.h`.

2.4.2.11 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format)` `[inline]`

Create a buffer for CUDA with given RTbuffertype and RTformat. See `rtBufferCreate`, `rtBufferSetFormat`.

Definition at line 1661 of file `optixpp_namespace.h`.

2.4.2.12 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width)` `[inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize1D`.

Definition at line 1669 of file `optixpp_namespace.h`.

2.4.2.13 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width, RTsize height)` `[inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize2D`.

Definition at line 1678 of file `optixpp_namespace.h`.

2.4.2.14 Buffer `optix::ContextObj::createBufferForCUDA (unsigned int type, RTformat format, RTsize width, RTsize height, RTsize depth)` `[inline]`

Create a buffer for CUDA with given RTbuffertype, RTformat and dimension. See `rtBufferCreate`, `rtBufferSetFormat` and `rtBufferSetSize3D`.

Definition at line 1687 of file `optixpp_namespace.h`.

2.4.2.15 Buffer `optix::ContextObj::createBufferFromGLBO (unsigned int type, unsigned int vbo)` `[inline]`

Create buffer from GL buffer object. See `rtBufferCreateFromGLBO`.

Definition at line 1696 of file `optixpp_namespace.h`.

2.4.2.16 Geometry `optix::ContextObj::createGeometry ()` `[inline]`

See `rtGeometryCreate`.

Definition at line 1771 of file `optixpp_namespace.h`.

2.4.2.17 GeometryGroup `optix::ContextObj::createGeometryGroup ()` `[inline]`

See `rtGeometryGroupCreate`.

Definition at line 1821 of file `optixpp_namespace.h`.

2.4.2.18 template<class Iterator > GeometryGroup `optix::ContextObj::createGeometryGroup (Iterator childbegin, Iterator childend)` `[inline]`

Create a GeometryGroup with a set of child nodes. See `rtGeometryGroupCreate`, `rtGeometryGroupSetChildCount` and `rtGeometryGroupSetChild`

Definition at line 1829 of file `optixpp_namespace.h`.

2.4.2.19 GeometryInstance `optix::ContextObj::createGeometryInstance ()`
[inline]

See `rtGeometryInstanceCreate`.

Definition at line 1778 of file `optixpp_namespace.h`.

2.4.2.20 `template<class Iterator > GeometryInstance optix::ContextObj::create-
GeometryInstance (Geometry geometry, Iterator matlbegin, Iterator matlend
)`

Create a geometry instance with a Geometry object and a set of associated materials. See `rtGeometryInstanceCreate`, `rtGeometryInstanceSetMaterialCount`, and `rtGeometryInstanceSetMaterial`

Definition at line 1786 of file `optixpp_namespace.h`.

2.4.2.21 Group `optix::ContextObj::createGroup ()` [inline]

See `rtGroupCreate`.

Definition at line 1800 of file `optixpp_namespace.h`.

2.4.2.22 `template<class Iterator > Group optix::ContextObj::createGroup (Iterator
childbegin, Iterator childend)` [inline]

Create a Group with a set of child nodes. See `rtGroupCreate`, `rtGroupSetChildCount` and `rtGroupSetChild`

Definition at line 1808 of file `optixpp_namespace.h`.

2.4.2.23 Material `optix::ContextObj::createMaterial ()` [inline]

See `rtMaterialCreate`.

Definition at line 1849 of file `optixpp_namespace.h`.

2.4.2.24 Program `optix::ContextObj::createProgramFromPTXFile (const std::string
& ptx, const std::string & program_name)` [inline]

See `rtProgramCreateFromPTXFile`.

Definition at line 1856 of file `optixpp_namespace.h`.

2.4.2.25 Program `optix::ContextObj::createProgramFromPTXString (const
std::string & ptx, const std::string & program_name)` [inline]

See `rtProgramCreateFromPTXString`.

Definition at line 1863 of file `optixpp_namespace.h`.

2.4.2.26 Selector `optix::ContextObj::createSelector ()` [inline]

See `rtSelectorCreate`.

Definition at line 1870 of file `optixpp_namespace.h`.

2.4.2.27 TextureSampler `optix::ContextObj::createTextureSampler ()`
[inline]

See `rtTextureSamplerCreate`.

Definition at line 1877 of file `optixpp_namespace.h`.

2.4.2.28 TextureSampler `optix::ContextObj::createTextureSamplerFromGLImage`
(unsigned int *id*, RTgltarget *target*) [inline]

Create TextureSampler from GL image. See `rtTextureSamplerCreateFromGLImage`.

Definition at line 1764 of file `optixpp_namespace.h`.

2.4.2.29 Transform `optix::ContextObj::createTransform ()` [inline]

See `rtTransformCreate`.

Definition at line 1842 of file `optixpp_namespace.h`.

2.4.2.30 Variable `optix::ContextObj::declareVariable (const std::string & name)`
[inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2125 of file `optixpp_namespace.h`.

2.4.2.31 void `optix::ContextObj::destroy ()` [inline, virtual]

Destroy Context and all of its associated objects. See `rtContextDestroy`.

Implements `optix::DestroyableObj`.

Definition at line 1591 of file `optixpp_namespace.h`.

2.4.2.32 RTcontext `optix::ContextObj::get ()` [inline]

Return the OptiX C API RTcontext object.

Definition at line 2159 of file `optixpp_namespace.h`.

2.4.2.33 RTsize `optix::ContextObj::getAvailableDeviceMemory (int ordinal) const`
[inline]

See `rtContextGetAttribute`.

Definition at line 1949 of file `optixpp_namespace.h`.

2.4.2.34 Context `optix::ContextObj::getContext () const` [inline, virtual]

Retrieve the Context object associated with this APIObject. In this case, simply returns itself.

Implements [optix::APIObj](#).

Definition at line 1546 of file [optixpp_namespace.h](#).

2.4.2.35 `int optix::ContextObj::getCPUNumThreads () const [inline]`

See [rtContextGetAttribute](#).

Definition at line 1921 of file [optixpp_namespace.h](#).

2.4.2.36 `void optix::ContextObj::getDeviceAttribute (int ordinal, RTdeviceattribute attrib, RTsize size, void * p) [inline, static]`

Call [rtDeviceGetAttribute](#) and return the desired attribute value.

Definition at line 1576 of file [optixpp_namespace.h](#).

2.4.2.37 `unsigned int optix::ContextObj::getDeviceCount () [inline, static]`

Call [rtDeviceGetDeviceCount](#) and returns number of valid devices.

Definition at line 1557 of file [optixpp_namespace.h](#).

2.4.2.38 `std::string optix::ContextObj::getDeviceName (int ordinal) [inline, static]`

Call [rtDeviceGetAttribute](#) and return the name of the device.

Definition at line 1566 of file [optixpp_namespace.h](#).

2.4.2.39 `unsigned int optix::ContextObj::getEnabledDeviceCount () const [inline]`

See [rtContextGetDeviceCount](#). As opposed to [getDeviceCount](#), this returns only the number of enabled devices.

Definition at line 1907 of file [optixpp_namespace.h](#).

2.4.2.40 `std::vector< int > optix::ContextObj::getEnabledDevices () const [inline]`

See [rtContextGetDevices](#). This returns the list of currently enabled devices.

Definition at line 1899 of file [optixpp_namespace.h](#).

2.4.2.41 `unsigned int optix::ContextObj::getEntryPointCount () const [inline]`

See [rtContextgetEntryPointCount](#).

Definition at line 1990 of file [optixpp_namespace.h](#).

2.4.2.42 `std::string optix::ContextObj::getErrorString (RTresult code) const [inline]`

See [rtContextGetErrroString](#).

Definition at line 1884 of file [optixpp_namespace.h](#).

2.4.2.43 `bool optix::ContextObj::getExceptionEnabled (RException exception) const [inline]`

See `rtContextGetExceptionEnabled`.

Definition at line 2029 of file [optixpp_namespace.h](#).

2.4.2.44 `Program optix::ContextObj::getExceptionProgram (unsigned int entry_point_index) const [inline]`

See `rtContextGetExceptionProgram`.

Definition at line 2016 of file [optixpp_namespace.h](#).

2.4.2.45 `int optix::ContextObj::getGPUPagingActive () const [inline]`

See `rtContextGetAttribute`.

Definition at line 1935 of file [optixpp_namespace.h](#).

2.4.2.46 `int optix::ContextObj::getGPUPagingForcedOff () const [inline]`

See `rtContextGetAttribute`.

Definition at line 1942 of file [optixpp_namespace.h](#).

2.4.2.47 `int optix::ContextObj::getMaxTextureCount () const [inline]`

See `rtContextGetAttribute`

Definition at line 1914 of file [optixpp_namespace.h](#).

2.4.2.48 `Program optix::ContextObj::getMissProgram (unsigned int ray_type_index) const [inline]`

See `rtContextGetMissProgram`.

Definition at line 2054 of file [optixpp_namespace.h](#).

2.4.2.49 `RTsize optix::ContextObj::getPrintBufferSize () const [inline]`

See `rtContextGetPrintBufferSize`.

Definition at line 2106 of file [optixpp_namespace.h](#).

2.4.2.50 `bool optix::ContextObj::getPrintEnabled () const [inline]`

See `rtContextGetPrintEnabled`.

Definition at line 2094 of file [optixpp_namespace.h](#).

2.4.2.51 `optix::int3 optix::ContextObj::getPrintLaunchIndex () const [inline]`

See `rtContextGetPrintLaunchIndex`.

Definition at line 2118 of file `optixpp_namespace.h`.

2.4.2.52 Program `optix::ContextObj::getRayGenerationProgram (unsigned int entry_point_index) const` `[inline]`

See `rtContextGetRayGenerationProgram`.

Definition at line 2003 of file `optixpp_namespace.h`.

2.4.2.53 unsigned int `optix::ContextObj::getRayTypeCount () const` `[inline]`

See `rtContextGetRayTypeCount`.

Definition at line 2042 of file `optixpp_namespace.h`.

2.4.2.54 int `optix::ContextObj::getRunningState () const` `[inline]`

See `rtContextGetRunningState`.

Definition at line 2082 of file `optixpp_namespace.h`.

2.4.2.55 RTsize `optix::ContextObj::getStackSize () const` `[inline]`

See `rtContextGetStackSize`.

Definition at line 1973 of file `optixpp_namespace.h`.

2.4.2.56 RTsize `optix::ContextObj::getUsedHostMemory () const` `[inline]`

See `rtContextGetAttribute`.

Definition at line 1928 of file `optixpp_namespace.h`.

2.4.2.57 Variable `optix::ContextObj::getVariable (unsigned int index) const` `[inline, virtual]`

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2151 of file `optixpp_namespace.h`.

2.4.2.58 unsigned int `optix::ContextObj::getVariableCount () const` `[inline, virtual]`

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2144 of file `optixpp_namespace.h`.

2.4.2.59 void `optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width)` `[inline]`

See `rtContextLaunch1D`

Definition at line 2066 of file `optixpp_namespace.h`.

2.4.2.60 `void optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width, RTsize image_height) [inline]`

See `rtContextLaunch2D`.

Definition at line 2071 of file `optixpp_namespace.h`.

2.4.2.61 `void optix::ContextObj::launch (unsigned int entry_point_index, RTsize image_width, RTsize image_height, RTsize image_depth) [inline]`

See `rtContextLaunch3D`.

Definition at line 2076 of file `optixpp_namespace.h`.

2.4.2.62 `Variable optix::ContextObj::queryVariable (const std::string & name) const [inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2132 of file `optixpp_namespace.h`.

2.4.2.63 `void optix::ContextObj::removeVariable (Variable v) [inline, virtual]`

Remove a variable associated with this object.

Implements `optix::ScopedObj`.

Definition at line 2139 of file `optixpp_namespace.h`.

2.4.2.64 `void optix::ContextObj::setCPUNumThreads (int cpu_num_threads) [inline]`

See `rtContextSetAttribute`

Definition at line 1958 of file `optixpp_namespace.h`.

2.4.2.65 `template<class Iterator > void optix::ContextObj::setDevices (Iterator begin, Iterator end) [inline]`

See `rtContextSetDevices`

Definition at line 1892 of file `optixpp_namespace.h`.

2.4.2.66 `void optix::ContextObj::setEntryPointCount (unsigned int num_entry_points) [inline]`

See `rtContextSetEntryPointCount`.

Definition at line 1985 of file `optixpp_namespace.h`.

2.4.2.67 `void optix::ContextObj::setExceptionEnabled (RException exception, bool enabled) [inline]`

See `rtContextSetExceptionEnabled`.

Definition at line 2024 of file `optixpp_namespace.h`.

2.4.2.68 `void optix::ContextObj::setExceptionProgram (unsigned int entry_point_index, Program program) [inline]`

See `rtContextSetExceptionProgram`.

Definition at line 2011 of file `optixpp_namespace.h`.

2.4.2.69 `void optix::ContextObj::setGPUPagingForcedOff (int gpu_paging_forced_off) [inline]`

See `rtContextSetAttribute`.

Definition at line 1963 of file `optixpp_namespace.h`.

2.4.2.70 `void optix::ContextObj::setMissProgram (unsigned int ray_type_index, Program program) [inline]`

See `rtContextSetMissProgram`.

Definition at line 2049 of file `optixpp_namespace.h`.

2.4.2.71 `void optix::ContextObj::setPrintBufferSize (RTsize buffer_size_bytes) [inline]`

See `rtContextSetPrintBufferSize`.

Definition at line 2101 of file `optixpp_namespace.h`.

2.4.2.72 `void optix::ContextObj::setPrintEnabled (bool enabled) [inline]`

See `rtContextSetPrintEnabled`

Definition at line 2089 of file `optixpp_namespace.h`.

2.4.2.73 `void optix::ContextObj::setPrintLaunchIndex (int x, int y = -1, int z = -1) [inline]`

See `rtContextSetPrintLaunchIndex`.

Definition at line 2113 of file `optixpp_namespace.h`.

2.4.2.74 `void optix::ContextObj::setRayGenerationProgram (unsigned int entry_point_index, Program program) [inline]`

See `rtContextSetRayGenerationProgram`

Definition at line 1998 of file `optixpp_namespace.h`.

2.4.2.75 `void optix::ContextObj::setRayTypeCount (unsigned int num_ray_types)`
[inline]

See `rtContextSetRayTypeCount`.

Definition at line 2037 of file `optixpp_namespace.h`.

2.4.2.76 `void optix::ContextObj::setStackSize (RTsize stack_size_bytes)` [inline]

See `rtContextSetStackSize`

Definition at line 1968 of file `optixpp_namespace.h`.

2.4.2.77 `void optix::ContextObj::setTimeoutCallback (RTtimeoutcallback callback,
double min_polling_seconds)` [inline]

See `rtContextSetTimeoutCallback` `RTtimeoutcallback` is defined as `typedef int (*RT-Timeoutcallback)(void)`.

Definition at line 1980 of file `optixpp_namespace.h`.

2.4.2.78 `void optix::ContextObj::validate ()` [inline, virtual]

See `rtContextValidate`.

Implements `optix::DestroyableObj`.

Definition at line 1597 of file `optixpp_namespace.h`.

2.4.3 Friends And Related Function Documentation

2.4.3.1 `friend class Handle< ContextObj >` [friend]

Definition at line 886 of file `optixpp_namespace.h`.

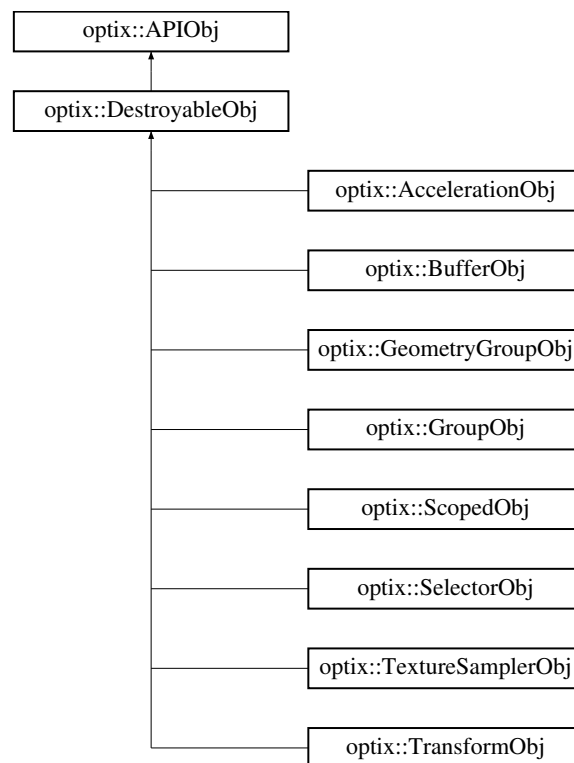
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.5 `optix::DestroyableObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::DestroyableObj`:



Public Member Functions

- virtual `~DestroyableObj ()`
- virtual void `destroy ()=0`
- virtual void `validate ()=0`

2.5.1 Detailed Description

Base class for all wrapper objects which can be destroyed and validated.

Wraps:

- `RTcontext`
- `RTgeometry`
- `RTgeometryinstance`
- `RTgeometrygroup`
- `RTgroup`
- `RTmaterial`
- `RTprogram`

- RTselector
- RTtexturesampler
- RTtransform

Definition at line 341 of file [optixpp_namespace.h](#).

2.5.2 Constructor & Destructor Documentation

2.5.2.1 virtual `optix::DestroyableObj::~~DestroyableObj()` [inline, virtual]

Definition at line 343 of file [optixpp_namespace.h](#).

2.5.3 Member Function Documentation

2.5.3.1 virtual void `optix::DestroyableObj::destroy()` [pure virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implemented in [optix::BufferObj](#), [optix::TextureSamplerObj](#), [optix::MaterialObj](#), [optix::GeometryObj](#), [optix::GeometryInstanceObj](#), [optix::AccelerationObj](#), [optix::SelectorObj](#), [optix::TransformObj](#), [optix::GeometryGroupObj](#), [optix::GroupObj](#), [optix::ProgramObj](#), and [optix::ContextObj](#).

2.5.3.2 virtual void `optix::DestroyableObj::validate()` [pure virtual]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implemented in [optix::BufferObj](#), [optix::TextureSamplerObj](#), [optix::MaterialObj](#), [optix::GeometryObj](#), [optix::GeometryInstanceObj](#), [optix::AccelerationObj](#), [optix::SelectorObj](#), [optix::TransformObj](#), [optix::GeometryGroupObj](#), [optix::GroupObj](#), [optix::ProgramObj](#), and [optix::ContextObj](#).

The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.6 optix::Exception Class Reference

```
#include <optixpp_namespace.h>
```

Public Member Functions

- [Exception](#) (const std::string &message, RTresult error_code=RT_ERROR_UNKNOWN)
- virtual `~Exception()` throw ()
- const std::string & `getErrorString()` const

- RTresult [getErrorCode](#) () const
- virtual const char * [what](#) () const throw ()

Static Public Member Functions

- static [Exception makeException](#) (RTresult code, RTcontext context)

2.6.1 Detailed Description

[Exception](#) class for error reporting from the OptiXpp API.

Encapsulates an error message, often the direct result of a failed OptiX C API function call and subsequent `rtContextGetErrorString` call.

Definition at line 235 of file [optixpp_namespace.h](#).

2.6.2 Constructor & Destructor Documentation

2.6.2.1 `optix::Exception::Exception (const std::string & message, RTresult error_code = RT_ERROR_UNKNOWN) [inline]`

Create exception.

Definition at line 238 of file [optixpp_namespace.h](#).

2.6.2.2 `virtual optix::Exception::~Exception () throw () [inline, virtual]`

Virtual destructor (needed for virtual function calls inherited from `std::exception`).

Definition at line 243 of file [optixpp_namespace.h](#).

2.6.3 Member Function Documentation

2.6.3.1 `RTresult optix::Exception::getErrorCode () const [inline]`

Retrieve the error code.

Definition at line 249 of file [optixpp_namespace.h](#).

2.6.3.2 `const std::string& optix::Exception::getErrorString () const [inline]`

Retrieve the error message.

Definition at line 246 of file [optixpp_namespace.h](#).

2.6.3.3 `Exception optix::Exception::makeException (RTresult code, RTcontext context) [inline, static]`

Helper for creating exceptions from an RTresult code origination from an OptiX C API function call.

Definition at line 262 of file [optixpp_namespace.h](#).

2.6.3.4 `virtual const char* optix::Exception::what () const throw () [inline, virtual]`

From `std::exception`.

Definition at line 256 of file [optixpp_namespace.h](#).

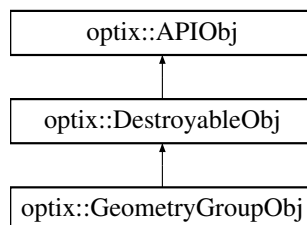
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.7 optix::GeometryGroupObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::GeometryGroupObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTgeometrygroup](#) [get](#) ()

Friends

- class [Handle](#)< [GeometryGroupObj](#) >
- void [setAcceleration](#) ([Acceleration](#) acceleration)
- [Acceleration](#) [getAcceleration](#) () const
- void [setChildCount](#) (unsigned int count)
- unsigned int [getChildCount](#) () const
- void [setChild](#) (unsigned int index, [GeometryInstance](#) geometryinstance)
- [GeometryInstance](#) [getChild](#) (unsigned int index) const

2.7.1 Detailed Description

`GeometryGroup` wraps the OptiX C API `RTgeometrygroup` opaque type and its associated function set.

Definition at line 969 of file `optixpp_namespace.h`.

2.7.2 Member Function Documentation

2.7.2.1 `void optix::GeometryGroupObj::destroy()` `[inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2378 of file `optixpp_namespace.h`.

2.7.2.2 `RTgeometrygroup optix::GeometryGroupObj::get()` `[inline]`

Get the underlying OptiX C API `RTgeometrygroup` opaque pointer.

Definition at line 2433 of file `optixpp_namespace.h`.

2.7.2.3 `Acceleration optix::GeometryGroupObj::getAcceleration()` `const` `[inline]`

Query the `Acceleration` structure for this group. See `rtGeometryGroupGetAcceleration`.

Definition at line 2402 of file `optixpp_namespace.h`.

2.7.2.4 `GeometryInstance optix::GeometryGroupObj::getChild(unsigned int index)` `const` `[inline]`

Query an indexed `GeometryInstance` within this group. See `rtGeometryGroupGetChild`.

Definition at line 2426 of file `optixpp_namespace.h`.

2.7.2.5 `unsigned int optix::GeometryGroupObj::getChildCount()` `const` `[inline]`

Query the number of children for this group. See `rtGeometryGroupGetChildCount`.

Definition at line 2414 of file `optixpp_namespace.h`.

2.7.2.6 `Context optix::GeometryGroupObj::getContext()` `const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2390 of file `optixpp_namespace.h`.

2.7.2.7 `void optix::GeometryGroupObj::setAcceleration (Acceleration acceleration)` [inline]

Set the Acceleration structure for this group. See `rtGeometryGroupSetAcceleration`.

Definition at line 2397 of file `optixpp_namespace.h`.

2.7.2.8 `void optix::GeometryGroupObj::setChild (unsigned int index, GeometryInstance geometryinstance)` [inline]

Set an indexed GeometryInstance child of this group. See `rtGeometryGroupSetChild`.

Definition at line 2421 of file `optixpp_namespace.h`.

2.7.2.9 `void optix::GeometryGroupObj::setChildCount (unsigned int count)` [inline]

Set the number of children for this group. See `rtGeometryGroupSetChildCount`.

Definition at line 2409 of file `optixpp_namespace.h`.

2.7.2.10 `void optix::GeometryGroupObj::validate ()` [inline, virtual]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2385 of file `optixpp_namespace.h`.

2.7.3 Friends And Related Function Documentation

2.7.3.1 `friend class Handle< GeometryGroupObj >` [friend]

Definition at line 1002 of file `optixpp_namespace.h`.

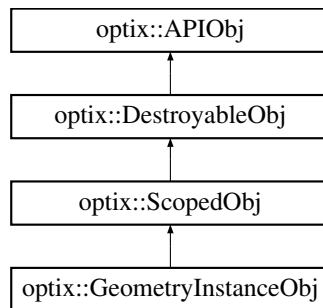
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.8 `optix::GeometryInstanceObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::GeometryInstanceObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTgeometryinstance](#) [get](#) ()

Friends

- class [Handle](#)< [GeometryInstanceObj](#) >
- void [setGeometry](#) ([Geometry](#) geometry)
- [Geometry](#) [getGeometry](#) () const
- void [setMaterialCount](#) (unsigned int count)
- unsigned int [getMaterialCount](#) () const
- void [setMaterial](#) (unsigned int idx, [Material](#) material)
- [Material](#) [getMaterial](#) (unsigned int idx) const
- unsigned int [addMaterial](#) ([Material](#) material)
- [Variable](#) [declareVariable](#) (const std::string &name)
- [Variable](#) [queryVariable](#) (const std::string &name) const
- void [removeVariable](#) ([Variable](#) v)
- unsigned int [getVariableCount](#) () const
- [Variable](#) [getVariable](#) (unsigned int index) const

2.8.1 Detailed Description

`GeometryInstance` wraps the OptiX C API `RTgeometryinstance` acceleration opaque type and its associated function set.

Definition at line 1160 of file [optixpp_namespace.h](#).

2.8.2 Member Function Documentation

2.8.2.1 `unsigned int optix::GeometryInstanceObj::addMaterial (Material material)` [inline]

Adds the provided material and returns the index to newly added material; increases material count by one.

Definition at line 2631 of file `optixpp_namespace.h`.

2.8.2.2 Variable `optix::GeometryInstanceObj::declareVariable (const std::string & name)` [inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2639 of file `optixpp_namespace.h`.

2.8.2.3 `void optix::GeometryInstanceObj::destroy ()` [inline, virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2575 of file `optixpp_namespace.h`.

2.8.2.4 `RTgeometryinstance optix::GeometryInstanceObj::get ()` [inline]

Get the underlying OptiX C API `RTgeometryinstance` opaque pointer.

Definition at line 2672 of file `optixpp_namespace.h`.

2.8.2.5 `Context optix::GeometryInstanceObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2587 of file `optixpp_namespace.h`.

2.8.2.6 `Geometry optix::GeometryInstanceObj::getGeometry () const` [inline]

Get the geometry object associated with this instance. See `rtGeometryInstanceGetGeometry`.

Definition at line 2599 of file `optixpp_namespace.h`.

2.8.2.7 `Material optix::GeometryInstanceObj::getMaterial (unsigned int idx) const` [inline]

Get the material at given index. See `rtGeometryInstanceGetMaterial`.

Definition at line 2623 of file [optixpp_namespace.h](#).

2.8.2.8 unsigned int `optix::GeometryInstanceObj::getMaterialCount () const`
[inline]

Query the number of materials associated with this instance. See `rtGeometryInstanceGetMaterialCount`.

Definition at line 2611 of file [optixpp_namespace.h](#).

2.8.2.9 Variable `optix::GeometryInstanceObj::getVariable (unsigned int index) const`
[inline, virtual]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements [optix::ScopedObj](#).

Definition at line 2665 of file [optixpp_namespace.h](#).

2.8.2.10 unsigned int `optix::GeometryInstanceObj::getVariableCount () const`
[inline, virtual]

Query the number of variables associated with this object. Used along with [ScopedObj::getVariable](#) to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements [optix::ScopedObj](#).

Definition at line 2658 of file [optixpp_namespace.h](#).

2.8.2.11 Variable `optix::GeometryInstanceObj::queryVariable (const std::string & name) const` [inline, virtual]

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2646 of file [optixpp_namespace.h](#).

2.8.2.12 void `optix::GeometryInstanceObj::removeVariable (Variable v)`
[inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2653 of file [optixpp_namespace.h](#).

2.8.2.13 void `optix::GeometryInstanceObj::setGeometry (Geometry geometry)`
[inline]

Set the geometry object associated with this instance. See `rtGeometryInstanceSetGeometry`.

Definition at line 2594 of file [optixpp_namespace.h](#).

2.8.2.14 `void optix::GeometryInstanceObj::setMaterial (unsigned int idx, Material material) [inline]`

Set the material at given index. See `rtGeometryInstanceSetMaterial`.

Definition at line 2618 of file `optixpp_namespace.h`.

2.8.2.15 `void optix::GeometryInstanceObj::setMaterialCount (unsigned int count) [inline]`

Set the number of materials associated with this instance. See `rtGeometryInstanceSetMaterialCount`.

Definition at line 2606 of file `optixpp_namespace.h`.

2.8.2.16 `void optix::GeometryInstanceObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2582 of file `optixpp_namespace.h`.

2.8.3 Friends And Related Function Documentation

2.8.3.1 `friend class Handle< GeometryInstanceObj > [friend]`

Definition at line 1202 of file `optixpp_namespace.h`.

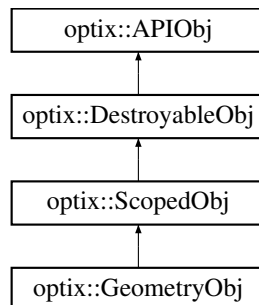
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.9 optix::GeometryObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::GeometryObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTgeometry](#) [get](#) ()

Friends

- class [Handle](#)< [GeometryObj](#) >
- void [markDirty](#) ()
- bool [isDirty](#) () const
- void [setPrimitiveCount](#) (unsigned int num_primitives)
- unsigned int [getPrimitiveCount](#) () const
- void [setBoundingBoxProgram](#) ([Program](#) program)
- [Program](#) [getBoundingBoxProgram](#) () const
- void [setIntersectionProgram](#) ([Program](#) program)
- [Program](#) [getIntersectionProgram](#) () const
- [Variable](#) [declareVariable](#) (const std::string &name)
- [Variable](#) [queryVariable](#) (const std::string &name) const
- void [removeVariable](#) ([Variable](#) v)
- unsigned int [getVariableCount](#) () const
- [Variable](#) [getVariable](#) (unsigned int index) const

2.9.1 Detailed Description

Geometry wraps the OptiX C API [RTgeometry](#) opaque type and its associated function set.

Definition at line [1212](#) of file [optixpp_namespace.h](#).

2.9.2 Member Function Documentation
2.9.2.1 Variable [optix::GeometryObj::declareVariable](#) (const std::string & name)
 [[inline](#), [virtual](#)]

Declare a variable associated with this object. See [rt\[ObjectType\]DeclareVariable](#). Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line [2732](#) of file [optixpp_namespace.h](#).

2.9.2.2 void `optix::GeometryObj::destroy ()` [`inline`, `virtual`]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2677 of file `optixpp_namespace.h`.

2.9.2.3 `RTgeometry` `optix::GeometryObj::get ()` [`inline`]

Get the underlying OptiX C API `RTgeometry` opaque pointer.

Definition at line 2777 of file `optixpp_namespace.h`.

2.9.2.4 Program `optix::GeometryObj::getBoundingBoxProgram ()` `const` [`inline`]

Get the bounding box program for this geometry. See `rtGeometryGetBoundingBoxProgram`.

Definition at line 2713 of file `optixpp_namespace.h`.

2.9.2.5 Context `optix::GeometryObj::getContext ()` `const` [`inline`, `virtual`]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2689 of file `optixpp_namespace.h`.

2.9.2.6 Program `optix::GeometryObj::getIntersectionProgram ()` `const` [`inline`]

Get the intersection program for this geometry. See `rtGeometryGetIntersectionProgram`.

Definition at line 2725 of file `optixpp_namespace.h`.

2.9.2.7 unsigned int `optix::GeometryObj::getPrimitiveCount ()` `const` [`inline`]

Query the number of primitives in this geometry objects (eg, number of triangles in mesh). See `rtGeometryGetPrimitiveCount`

Definition at line 2701 of file `optixpp_namespace.h`.

2.9.2.8 Variable `optix::GeometryObj::getVariable (unsigned int index)` `const` [`inline`, `virtual`]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2758 of file `optixpp_namespace.h`.

2.9.2.9 `unsigned int optix::GeometryObj::getVariableCount () const` `[inline, virtual]`

Query the number of variables associated with this object. Used along with [ScopedObj::getVariable](#) to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`. Implements [optix::ScopedObj](#).

Definition at line 2751 of file [optixpp_namespace.h](#).

2.9.2.10 `bool optix::GeometryObj::isDirty () const` `[inline]`

Query whether this geometry has been marked dirty. See `rtGeometryIsDirty`.

Definition at line 2770 of file [optixpp_namespace.h](#).

2.9.2.11 `void optix::GeometryObj::markDirty ()` `[inline]`

Mark this geometry as dirty, causing rebuild of parent groups acceleration. See `rtGeometryMarkDirty`.

Definition at line 2765 of file [optixpp_namespace.h](#).

2.9.2.12 `Variable optix::GeometryObj::queryVariable (const std::string & name) const` `[inline, virtual]`

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2739 of file [optixpp_namespace.h](#).

2.9.2.13 `void optix::GeometryObj::removeVariable (Variable v)` `[inline, virtual]`

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2746 of file [optixpp_namespace.h](#).

2.9.2.14 `void optix::GeometryObj::setBoundingBoxProgram (Program program)` `[inline]`

Set the bounding box program for this geometry. See `rtGeometrySetBoundingBoxProgram`.

Definition at line 2708 of file [optixpp_namespace.h](#).

2.9.2.15 `void optix::GeometryObj::setIntersectionProgram (Program program)` `[inline]`

Set the intersection program for this geometry. See `rtGeometrySetIntersectionProgram`.

Definition at line 2720 of file [optixpp_namespace.h](#).

2.9.2.16 void `optix::GeometryObj::setPrimitiveCount (unsigned int num_primitives)`
`[inline]`

Set the number of primitives in this geometry objects (eg, number of triangles in mesh).
 See `rtGeometrySetPrimitiveCount`

Definition at line 2696 of file `optixpp_namespace.h`.

2.9.2.17 void `optix::GeometryObj::validate ()` `[inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2684 of file `optixpp_namespace.h`.

2.9.3 Friends And Related Function Documentation

2.9.3.1 friend class `Handle< GeometryObj >` `[friend]`

Definition at line 1262 of file `optixpp_namespace.h`.

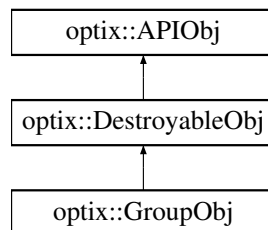
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.10 optix::GroupObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::GroupObj`:



Public Member Functions

- void `destroy ()`
- void `validate ()`
- `Context getContext () const`
- `RTgroup get ()`

Friends

- class `Handle< GroupObj >`
- void `setAcceleration (Acceleration acceleration)`
- `Acceleration getAcceleration () const`
- void `setChildCount (unsigned int count)`
- unsigned int `getChildCount () const`
- template<typename T >
void `setChild (unsigned int index, T child)`
- template<typename T >
T `getChild (unsigned int index) const`

2.10.1 Detailed Description

Group wraps the OptiX C API `RTgroup` opaque type and its associated function set.
Definition at line 925 of file `optixpp_namespace.h`.

2.10.2 Member Function Documentation

2.10.2.1 void `optix::GroupObj::destroy ()` [`inline`, `virtual`]

call `rt[ObjectType]Destroy` on the underlying OptiX C object
Implements `optix::DestroyableObj`.
Definition at line 2221 of file `optixpp_namespace.h`.

2.10.2.2 RTgroup `optix::GroupObj::get ()` [`inline`]

Get the underlying OptiX C API `RTgroup` opaque pointer.
Definition at line 2373 of file `optixpp_namespace.h`.

2.10.2.3 Acceleration `optix::GroupObj::getAcceleration () const` [`inline`]

Query the Acceleration structure for this group. See `rtGroupGetAcceleration`.
Definition at line 2340 of file `optixpp_namespace.h`.

2.10.2.4 template<typename T > T `optix::GroupObj::getChild (unsigned int index) const` [`inline`]

Query an indexed child within this group. See `rtGroupGetChild`.
Definition at line 2366 of file `optixpp_namespace.h`.

2.10.2.5 unsigned int `optix::GroupObj::getChildCount () const` [`inline`]

Query the number of children for this group. See `rtGroupGetChildCount`.

Definition at line 2352 of file [optixpp_namespace.h](#).

2.10.2.6 Context `optix::GroupObj::getContext () const` [`inline`, `virtual`]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2233 of file [optixpp_namespace.h](#).

2.10.2.7 `void optix::GroupObj::setAcceleration (Acceleration acceleration)` [`inline`]

Set the Acceleration structure for this group. See `rtGroupSetAcceleration`.

Definition at line 2335 of file [optixpp_namespace.h](#).

2.10.2.8 `template<typename T > void optix::GroupObj::setChild (unsigned int index, T child)` [`inline`]

Set an indexed child within this group. See `rtGroupSetChild`.

Definition at line 2360 of file [optixpp_namespace.h](#).

2.10.2.9 `void optix::GroupObj::setChildCount (unsigned int count)` [`inline`]

Set the number of children for this group. See `rtGroupSetChildCount`.

Definition at line 2347 of file [optixpp_namespace.h](#).

2.10.2.10 `void optix::GroupObj::validate ()` [`inline`, `virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2228 of file [optixpp_namespace.h](#).

2.10.3 Friends And Related Function Documentation

2.10.3.1 `friend class Handle< GroupObj >` [`friend`]

Definition at line 959 of file [optixpp_namespace.h](#).

The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.11 `optix::Handle< T >` Class Template Reference

```
#include <optixpp_namespace.h>
```


Public Member Functions

- [Handle](#) ()
- [Handle](#) (T *ptr)
- `template<class U >`
[Handle](#) (U *ptr)
- [Handle](#) (const [Handle](#)< T > ©)
- `template<class U >`
[Handle](#) (const [Handle](#)< U > ©)
- [Handle](#)< T > & [operator=](#) (const [Handle](#)< T > ©)
- `template<class U >`
[Handle](#)< T > & [operator=](#) (const [Handle](#)< U > ©)
- [~Handle](#) ()
- T * [operator->](#) ()
- const T * [operator->](#) () const
- T * [get](#) ()
- const T * [get](#) () const
- [operator bool](#) () const
- [Handle](#)< VariableObj > [operator\[\]](#) (const std::string &varname)
- [Handle](#)< VariableObj > [operator\[\]](#) (const char *varname)

Static Public Member Functions

- static [Handle](#)< T > [take](#) (typename T::api_t p)
- static [Handle](#)< T > [take](#) (RObject p)
- static [Handle](#)< T > [create](#) ()
- static unsigned int [getDeviceCount](#) ()

2.11.1 Detailed Description

```
template<class T>class optix::Handle< T >
```

The [Handle](#) class is a reference counted handle class used to manipulate API objects.

All interaction with API objects should be done via these handles and the associated typedefs rather than direct usage of the objects.

Definition at line 123 of file [optixpp_namespace.h](#).

2.11.2 Constructor & Destructor Documentation

2.11.2.1 `template<class T> optix::Handle< T >::Handle () [inline]`

Default constructor initializes handle to null pointer.

Definition at line 126 of file [optixpp_namespace.h](#).

2.11.2.2 `template<class T> optix::Handle< T >::Handle (T * ptr) [inline]`

Takes a raw pointer to an API object and creates a handle.

Definition at line 129 of file `optixpp_namespace.h`.

2.11.2.3 `template<class T> template<class U > optix::Handle< T >::Handle (U * ptr) [inline]`

Takes a raw pointer of arbitrary type and creates a handle.

Definition at line 133 of file `optixpp_namespace.h`.

2.11.2.4 `template<class T> optix::Handle< T >::Handle (const Handle< T > & copy) [inline]`

Takes a handle of the same type and creates a handle.

Definition at line 136 of file `optixpp_namespace.h`.

2.11.2.5 `template<class T> template<class U > optix::Handle< T >::Handle (const Handle< U > & copy) [inline]`

Takes a handle of some other type and creates a handle.

Definition at line 140 of file `optixpp_namespace.h`.

2.11.2.6 `template<class T> optix::Handle< T >::~~Handle () [inline]`

Decrements reference count on the handled object.

Definition at line 152 of file `optixpp_namespace.h`.

2.11.3 Member Function Documentation

2.11.3.1 `template<class T> static Handle<T> optix::Handle< T >::create () [inline, static]`

Static object creation. Only valid for contexts.

Definition at line 197 of file `optixpp_namespace.h`.

2.11.3.2 `template<class T> T* optix::Handle< T >::get () [inline]`

Retrieve the handled object.

Definition at line 165 of file `optixpp_namespace.h`.

2.11.3.3 `template<class T> const T* optix::Handle< T >::get () const [inline]`

Definition at line 166 of file `optixpp_namespace.h`.

2.11.3.4 `template<class T> static unsigned int optix::Handle< T >::getDeviceCount ()`
`[inline, static]`

Query the machine device count. Only valid for contexts.

Definition at line 200 of file [optixpp_namespace.h](#).

2.11.3.5 `template<class T> optix::Handle< T >::operator bool () const` `[inline]`

implicit bool cast based on NULLness of wrapped pointer

Definition at line 169 of file [optixpp_namespace.h](#).

2.11.3.6 `template<class T> T* optix::Handle< T >::operator-> ()` `[inline]`

Dereferences the handle.

Definition at line 161 of file [optixpp_namespace.h](#).

2.11.3.7 `template<class T> const T* optix::Handle< T >::operator-> () const`
`[inline]`

Definition at line 162 of file [optixpp_namespace.h](#).

2.11.3.8 `template<class T> Handle<T>& optix::Handle< T >::operator= (const`
`Handle< T > & copy)` `[inline]`

Assignment of handle with same underlying object type.

Definition at line 143 of file [optixpp_namespace.h](#).

2.11.3.9 `template<class T> template<class U > Handle<T>& optix::Handle< T`
`>::operator= (const Handle< U > & copy)` `[inline]`

Assignment of handle with different underlying object type.

Definition at line 148 of file [optixpp_namespace.h](#).

2.11.3.10 `template<class T > Handle< VariableObj > optix::Handle< T >::operator[] (`
`const std::string & varname)`

Variable access operator. This operator will query the API object for a variable with the given name, creating a new variable instance if necessary. Only valid for ScopedObjs.

Definition at line 598 of file [optixpp_namespace.h](#).

2.11.3.11 `template<class T > Handle< VariableObj > optix::Handle< T >::operator[] (`
`const char * varname)`

Variable access operator. Identical to [operator\[\]\(const std::string& varname\)](#)

Explicitly define char* version to avoid ambiguities between builtin `operator[](int, char*)` and `Handle::operator[](std::string)`. The problem lies in that a `Handle` can be cast to a `bool` then to an `int` which implies that:

```
Context context;
```

```
context["var"];
```

can be interpreted as either

```
l["var"]; // Strange but legal way to index into a string (same as "var"[1]
)
```

or

```
context[ std::string("var") ];
```

Definition at line 607 of file [optixpp_namespace.h](#).

2.11.3.12 `template<class T> static Handle<T> optix::Handle< T >::take (typename T::api_t p) [inline, static]`

Takes a base optix api opaque type and creates a handle to optixpp wrapper type.

Definition at line 155 of file [optixpp_namespace.h](#).

2.11.3.13 `template<class T> static Handle<T> optix::Handle< T >::take (RObject p) [inline, static]`

Special version that takes an RObject which must be cast up to the appropriate OptiX API opaque type.

Definition at line 158 of file [optixpp_namespace.h](#).

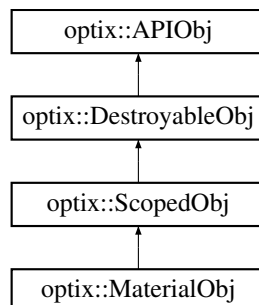
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.12 optix::MaterialObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for optix::MaterialObj:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTmaterial](#) [get](#) ()

Friends

- class [Handle](#)< [MaterialObj](#) >
- void [setClosestHitProgram](#) (unsigned int ray_type_index, [Program](#) program)
- [Program](#) [getClosestHitProgram](#) (unsigned int ray_type_index) const
- void [setAnyHitProgram](#) (unsigned int ray_type_index, [Program](#) program)
- [Program](#) [getAnyHitProgram](#) (unsigned int ray_type_index) const
- [Variable](#) [declareVariable](#) (const std::string &name)
- [Variable](#) [queryVariable](#) (const std::string &name) const
- void [removeVariable](#) ([Variable](#) v)
- unsigned int [getVariableCount](#) () const
- [Variable](#) [getVariable](#) (unsigned int index) const

2.12.1 Detailed Description

`Material` wraps the OptiX C API `RTmaterial` opaque type and its associated function set. Definition at line 1272 of file [optixpp_namespace.h](#).

2.12.2 Member Function Documentation**2.12.2.1 Variable `optix::MaterialObj::declareVariable (const std::string & name)` [inline, virtual]**

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function [Handle::operator\[\]](#).

Implements [optix::ScopedObj](#).

Definition at line 2825 of file [optixpp_namespace.h](#).

2.12.2.2 void `optix::MaterialObj::destroy ()` [inline, virtual]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2782 of file [optixpp_namespace.h](#).

2.12.2.3 RTmaterial `optix::MaterialObj::get ()` [inline]

Get the underlying OptiX C API RTmaterial opaque pointer.

Definition at line 2858 of file `optixpp_namespace.h`.

2.12.2.4 Program `optix::MaterialObj::getAnyHitProgram (unsigned int ray_type_index) const` [inline]

Get any hit program for this material at the given `ray_type` index. See `rtMaterialGetAnyHitProgram`.

Definition at line 2818 of file `optixpp_namespace.h`.

2.12.2.5 Program `optix::MaterialObj::getClosestHitProgram (unsigned int ray_type_index) const` [inline]

Get closest hit program for this material at the given `ray_type` index. See `rtMaterialGetClosestHitProgram`.

Definition at line 2806 of file `optixpp_namespace.h`.

2.12.2.6 Context `optix::MaterialObj::getContext () const` [inline, virtual]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2794 of file `optixpp_namespace.h`.

2.12.2.7 Variable `optix::MaterialObj::getVariable (unsigned int index) const` [inline, virtual]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2851 of file `optixpp_namespace.h`.

2.12.2.8 `unsigned int optix::MaterialObj::getVariableCount () const` [inline, virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2844 of file `optixpp_namespace.h`.

2.12.2.9 Variable `optix::MaterialObj::queryVariable (const std::string & name) const` [inline, virtual]

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements [optix::ScopedObj](#).

Definition at line 2832 of file [optixpp_namespace.h](#).

2.12.2.10 void [optix::MaterialObj::removeVariable](#) (Variable v) [inline, virtual]

Remove a variable associated with this object.

Implements [optix::ScopedObj](#).

Definition at line 2839 of file [optixpp_namespace.h](#).

2.12.2.11 void [optix::MaterialObj::setAnyHitProgram](#) (unsigned int ray_type_index, Program program) [inline]

Set any hit program for this material at the given *ray_type* index. See [rtMaterialSetAnyHitProgram](#).

Definition at line 2813 of file [optixpp_namespace.h](#).

2.12.2.12 void [optix::MaterialObj::setClosestHitProgram](#) (unsigned int ray_type_index, Program program) [inline]

Set closest hit program for this material at the given *ray_type* index. See [rtMaterialSetClosestHitProgram](#).

Definition at line 2801 of file [optixpp_namespace.h](#).

2.12.2.13 void [optix::MaterialObj::validate](#) () [inline, virtual]

call [rt\[ObjectType\]Validate](#) on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2789 of file [optixpp_namespace.h](#).

2.12.3 Friends And Related Function Documentation

2.12.3.1 friend class [Handle](#)< [MaterialObj](#) > [friend]

Definition at line 1305 of file [optixpp_namespace.h](#).

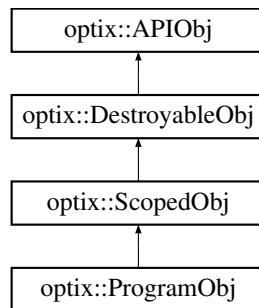
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.13 optix::ProgramObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for [optix::ProgramObj](#):



Public Member Functions

- void `destroy` ()
- void `validate` ()
- `Context` `getContext` () const
- `Variable` `declareVariable` (const std::string &name)
- `Variable` `queryVariable` (const std::string &name) const
- void `removeVariable` (`Variable` v)
- unsigned int `getVariableCount` () const
- `Variable` `getVariable` (unsigned int index) const
- `RTprogram` `get` ()

Friends

- class `Handle< ProgramObj >`

2.13.1 Detailed Description

Program object wraps the OptiX C API `RTprogram` opaque type and its associated function set.

Definition at line 896 of file `optixpp_namespace.h`.

2.13.2 Member Function Documentation

2.13.2.1 Variable `optix::ProgramObj::declareVariable` (const std::string & name) [inline, virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2183 of file `optixpp_namespace.h`.

2.13.2.2 `void optix::ProgramObj::destroy()` [`inline`, `virtual`]

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2164 of file `optixpp_namespace.h`.

2.13.2.3 `RTprogram optix::ProgramObj::get()` [`inline`]

Definition at line 2216 of file `optixpp_namespace.h`.

2.13.2.4 `Context optix::ProgramObj::getContext() const` [`inline`, `virtual`]

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2176 of file `optixpp_namespace.h`.

2.13.2.5 `Variable optix::ProgramObj::getVariable(unsigned int index) const` [`inline`, `virtual`]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implements `optix::ScopedObj`.

Definition at line 2209 of file `optixpp_namespace.h`.

2.13.2.6 `unsigned int optix::ProgramObj::getVariableCount() const` [`inline`, `virtual`]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implements `optix::ScopedObj`.

Definition at line 2202 of file `optixpp_namespace.h`.

2.13.2.7 `Variable optix::ProgramObj::queryVariable(const std::string & name) const` [`inline`, `virtual`]

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implements `optix::ScopedObj`.

Definition at line 2190 of file `optixpp_namespace.h`.

2.13.2.8 `void optix::ProgramObj::removeVariable(Variable v)` [`inline`, `virtual`]

Remove a variable associated with this object.

Implements `optix::ScopedObj`.

Definition at line 2197 of file `optixpp_namespace.h`.

2.13.2.9 `void optix::ProgramObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2171 of file [optixpp_namespace.h](#).

2.13.3 Friends And Related Function Documentation

2.13.3.1 `friend class Handle< ProgramObj > [friend]`

Definition at line 915 of file [optixpp_namespace.h](#).

The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.14 RTUtraversalresult Struct Reference

```
#include <optixu_traversal.h>
```

Public Attributes

- `int prim_id`
- `float t`

2.14.1 Detailed Description

Structure encapsulating the result of a single ray query.

Definition at line 35 of file [optixu_traversal.h](#).

2.14.2 Member Data Documentation

2.14.2.1 `int RTUtraversalresult::prim_id`

Index of the intereseected triangle, -1 for miss

Definition at line 36 of file [optixu_traversal.h](#).

2.14.2.2 `float RTUtraversalresult::t`

Ray t parameter of hit point

Definition at line 37 of file [optixu_traversal.h](#).

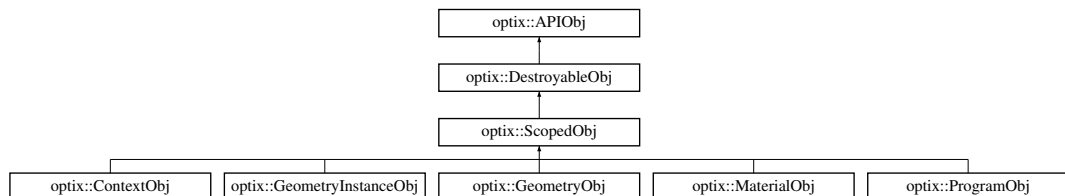
The documentation for this struct was generated from the following file:

- [optixu_traversal.h](#)

2.15 optix::ScopedObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for optix::ScopedObj:



Public Member Functions

- virtual `~ScopedObj()`
- virtual `Variable declareVariable(const std::string &name)=0`
- virtual `Variable queryVariable(const std::string &name) const =0`
- virtual `void removeVariable(Variable v)=0`
- virtual `unsigned int getVariableCount() const =0`
- virtual `Variable getVariable(unsigned int index) const =0`

2.15.1 Detailed Description

Base class for all objects which are OptiX variable containers.

Wraps:

- RTcontext
- RTgeometry
- RTgeometryinstance
- RTmaterial
- RTprogram

Definition at line 367 of file [optixpp_namespace.h](#).

2.15.2 Constructor & Destructor Documentation

2.15.2.1 virtual `optix::ScopedObj::~ScopedObj()` [`inline`, `virtual`]

Definition at line 369 of file [optixpp_namespace.h](#).

2.15.3 Member Function Documentation

2.15.3.1 virtual Variable optix::ScopedObj::declareVariable (const std::string & name)
[pure virtual]

Declare a variable associated with this object. See `rt[ObjectType]DeclareVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implemented in `optix::MaterialObj`, `optix::GeometryObj`, `optix::GeometryInstanceObj`, `optix::ProgramObj`, and `optix::ContextObj`.

2.15.3.2 virtual Variable optix::ScopedObj::getVariable (unsigned int index) const
[pure virtual]

Query variable by index. See `rt[ObjectType]GetVariable`.

Implemented in `optix::MaterialObj`, `optix::GeometryObj`, `optix::GeometryInstanceObj`, `optix::ProgramObj`, and `optix::ContextObj`.

2.15.3.3 virtual unsigned int optix::ScopedObj::getVariableCount () const [pure virtual]

Query the number of variables associated with this object. Used along with `ScopedObj::getVariable` to iterate over variables in an object. See `rt[ObjectType]GetVariableCount`

Implemented in `optix::MaterialObj`, `optix::GeometryObj`, `optix::GeometryInstanceObj`, `optix::ProgramObj`, and `optix::ContextObj`.

2.15.3.4 virtual Variable optix::ScopedObj::queryVariable (const std::string & name) const [pure virtual]

Query a variable associated with this object by name. See `rt[ObjectType]QueryVariable`. Note that this function is wrapped by the convenience function `Handle::operator[]`.

Implemented in `optix::MaterialObj`, `optix::GeometryObj`, `optix::GeometryInstanceObj`, `optix::ProgramObj`, and `optix::ContextObj`.

2.15.3.5 virtual void optix::ScopedObj::removeVariable (Variable v) [pure virtual]

Remove a variable associated with this object.

Implemented in `optix::MaterialObj`, `optix::GeometryObj`, `optix::GeometryInstanceObj`, `optix::ProgramObj`, and `optix::ContextObj`.

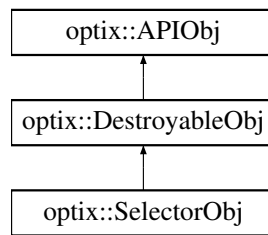
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.16 optix::SelectorObj Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::SelectorObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context getContext](#) () const
- [RTselector get](#) ()

Friends

- class [Handle< SelectorObj >](#)
- void [setVisitProgram](#) ([Program](#) program)
- [Program getVisitProgram](#) () const
- void [setChildCount](#) (unsigned int count)
- unsigned int [getChildCount](#) () const
- template<typename T >
void [setChild](#) (unsigned int index, T child)
- template<typename T >
T [getChild](#) (unsigned int index) const
- [Variable declareVariable](#) (const std::string &name)
- [Variable queryVariable](#) (const std::string &name) const
- void [removeVariable](#) ([Variable](#) v)
- unsigned int [getVariableCount](#) () const
- [Variable getVariable](#) (unsigned int index) const

2.16.1 Detailed Description

Selector wraps the OptiX C API RTselector opaque type and its associated function set.

Definition at line 1050 of file [optixpp_namespace.h](#).

2.16.2 Member Function Documentation

2.16.2.1 Variable [optix::SelectorObj::declareVariable](#) (const std::string & name) [inline]

Definition at line 2297 of file [optixpp_namespace.h](#).

2.16.2.2 `void optix::SelectorObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2240 of file `optixpp_namespace.h`.

2.16.2.3 `RTselector optix::SelectorObj::get () [inline]`

Get the underlying OptiX C API RTselector opaque pointer.

Definition at line 2330 of file `optixpp_namespace.h`.

2.16.2.4 `template<typename T > T optix::SelectorObj::getChild (unsigned int index) const [inline]`

Query an indexed child within this group. See `rtSelectorGetChild`.

Definition at line 2290 of file `optixpp_namespace.h`.

2.16.2.5 `unsigned int optix::SelectorObj::getChildCount () const [inline]`

Query the number of children for this group. See `rtSelectorGetChildCount`.

Definition at line 2276 of file `optixpp_namespace.h`.

2.16.2.6 `Context optix::SelectorObj::getContext () const [inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2252 of file `optixpp_namespace.h`.

2.16.2.7 `Variable optix::SelectorObj::getVariable (unsigned int index) const [inline]`

Definition at line 2323 of file `optixpp_namespace.h`.

2.16.2.8 `unsigned int optix::SelectorObj::getVariableCount () const [inline]`

Definition at line 2316 of file `optixpp_namespace.h`.

2.16.2.9 `Program optix::SelectorObj::getVisitProgram () const [inline]`

Get the visitor program for this selector. See `rtSelectorGetVisitProgram`.

Definition at line 2264 of file `optixpp_namespace.h`.

2.16.2.10 `Variable optix::SelectorObj::queryVariable (const std::string & name) const [inline]`

Definition at line 2304 of file `optixpp_namespace.h`.

2.16.2.11 `void optix::SelectorObj::removeVariable (Variable v) [inline]`

Definition at line 2311 of file `optixpp_namespace.h`.

2.16.2.12 `template<typename T> void optix::SelectorObj::setChild (unsigned int index, T child) [inline]`

Set an indexed child of this group. See `rtSelectorSetChild`.

Definition at line 2284 of file `optixpp_namespace.h`.

2.16.2.13 `void optix::SelectorObj::setChildCount (unsigned int count) [inline]`

Set the number of children for this group. See `rtSelectorSetChildCount`.

Definition at line 2271 of file `optixpp_namespace.h`.

2.16.2.14 `void optix::SelectorObj::setVisitProgram (Program program) [inline]`

Set the visitor program for this selector. See `rtSelectorSetVisitProgram`

Definition at line 2259 of file `optixpp_namespace.h`.

2.16.2.15 `void optix::SelectorObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2247 of file `optixpp_namespace.h`.

2.16.3 Friends And Related Function Documentation

2.16.3.1 `friend class Handle< SelectorObj > [friend]`

Definition at line 1091 of file `optixpp_namespace.h`.

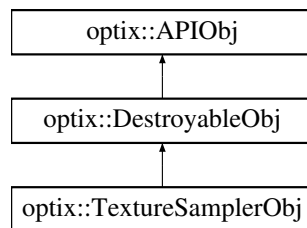
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.17 `optix::TextureSamplerObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::TextureSamplerObj`:



Public Member Functions

- void [destroy](#) ()
- void [validate](#) ()
- [Context](#) [getContext](#) () const
- [RTtexturesampler](#) [get](#) ()

Friends

- class [Handle](#)< [TextureSamplerObj](#) >
- void [setMipLevelCount](#) (unsigned int num_mip_levels)
- unsigned int [getMipLevelCount](#) () const
- void [setArraySize](#) (unsigned int num_textures_in_array)
- unsigned int [getArraySize](#) () const
- void [setWrapMode](#) (unsigned int dim, [RTwrapmode](#) wrapmode)
- [RTwrapmode](#) [getWrapMode](#) (unsigned int dim) const
- void [setFilteringModes](#) ([RTfiltermode](#) minification, [RTfiltermode](#) magnification, [RTfiltermode](#) mipmapping)
- void [getFilteringModes](#) ([RTfiltermode](#) &minification, [RTfiltermode](#) &magnification, [RTfiltermode](#) &mipmapping) const
- void [setMaxAnisotropy](#) (float value)
- float [getMaxAnisotropy](#) () const
- void [setReadMode](#) ([RTtexturereadmode](#) readmode)
- [RTtexturereadmode](#) [getReadMode](#) () const
- void [setIndexingMode](#) ([RTtextureindexmode](#) indexmode)
- [RTtextureindexmode](#) [getIndexingMode](#) () const
- int [getId](#) () const
- void [setBuffer](#) (unsigned int texture_array_idx, unsigned int mip_level, [Buffer](#) buffer)
- [Buffer](#) [getBuffer](#) (unsigned int texture_array_idx, unsigned int mip_level) const
- void [registerGLTexture](#) ()
- void [unregisterGLTexture](#) ()

2.17.1 Detailed Description

`TextureSampler` wraps the OptiX C API `RTtexturesampler` opaque type and its associated function set.

Definition at line 1315 of file `optixpp_namespace.h`.

2.17.2 Member Function Documentation

2.17.2.1 `void optix::TextureSamplerObj::destroy () [inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2863 of file `optixpp_namespace.h`.

2.17.2.2 `RTtexturesampler optix::TextureSamplerObj::get () [inline]`

Get the underlying OptiX C API `RTtexturesampler` opaque pointer.

Definition at line 2983 of file `optixpp_namespace.h`.

2.17.2.3 `unsigned int optix::TextureSamplerObj::getArraySize () const [inline]`

Query the texture array size for this sampler. See `rtTextureSamplerGetArraySize`.

Definition at line 2899 of file `optixpp_namespace.h`.

2.17.2.4 `Buffer optix::TextureSamplerObj::getBuffer (unsigned int texture_array_idx, unsigned int mip_level) const [inline]`

Get the underlying buffer used for texture storage. `rtTextureSamplerGetBuffer`.

Definition at line 2976 of file `optixpp_namespace.h`.

2.17.2.5 `Context optix::TextureSamplerObj::getContext () const [inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2875 of file `optixpp_namespace.h`.

2.17.2.6 `void optix::TextureSamplerObj::getFilteringModes (RTfiltermode & minification, RTfiltermode & magnification, RTfiltermode & mipmapping) const [inline]`

Query filtering modes for this sampler. See `rtTextureSamplerGetFilteringModes`.

Definition at line 2923 of file `optixpp_namespace.h`.

2.17.2.7 `int optix::TextureSamplerObj::getId () const` `[inline]`

Returns the device-side ID of this sampler.

Definition at line 2940 of file [optixpp_namespace.h](#).

2.17.2.8 `RTtextureindexmode optix::TextureSamplerObj::getIndexingMode () const` `[inline]`

Query texture indexing mode for this sampler. See `rtTextureSamplerGetIndexingMode`.

Definition at line 2964 of file [optixpp_namespace.h](#).

2.17.2.9 `float optix::TextureSamplerObj::getMaxAnisotropy () const` `[inline]`

Query maximum anisotropy for this sampler. See `rtTextureSamplerGetMaxAnisotropy`.

Definition at line 2933 of file [optixpp_namespace.h](#).

2.17.2.10 `unsigned int optix::TextureSamplerObj::getMipLevelCount () const` `[inline]`

Query the number of mip levels for this sampler. See `rtTextureSamplerGetMipLevelCount`.

Definition at line 2887 of file [optixpp_namespace.h](#).

2.17.2.11 `RTtexturereadmode optix::TextureSamplerObj::getReadMode () const` `[inline]`

Query texture read mode for this sampler. See `rtTextureSamplerGetReadMode`.

Definition at line 2952 of file [optixpp_namespace.h](#).

2.17.2.12 `RTwrapmode optix::TextureSamplerObj::getWrapMode (unsigned int dim) const` `[inline]`

Query the texture wrap mode for this sampler. See `rtTextureSamplerGetWrapMode`.

Definition at line 2911 of file [optixpp_namespace.h](#).

2.17.2.13 `void optix::TextureSamplerObj::registerGLTexture ()` `[inline]`

Declare the texture's buffer as mutable and inaccessible by OptiX. See `rtTextureSamplerGLRegister`.

Definition at line 2988 of file [optixpp_namespace.h](#).

2.17.2.14 `void optix::TextureSamplerObj::setArraySize (unsigned int num_textures_in_array)` `[inline]`

Set the texture array size for this sampler. See `rtTextureSamplerSetArraySize`.

Definition at line 2894 of file [optixpp_namespace.h](#).

2.17.2.15 `void optix::TextureSamplerObj::setBuffer (unsigned int texture_array_idx, unsigned int mip_level, Buffer buffer) [inline]`

Set the underlying buffer used for texture storage. `rtTextureSamplerSetBuffer`.

Definition at line 2971 of file `optixpp_namespace.h`.

2.17.2.16 `void optix::TextureSamplerObj::setFilteringModes (RTfiltermode minification, RTfiltermode magnification, RTfiltermode mipmapping) [inline]`

Set filtering modes for this sampler. See `rtTextureSamplerSetFilteringModes`.

Definition at line 2918 of file `optixpp_namespace.h`.

2.17.2.17 `void optix::TextureSamplerObj::setIndexingMode (RTtextureindexmode indexmode) [inline]`

Set texture indexing mode for this sampler. See `rtTextureSamplerSetIndexingMode`.

Definition at line 2959 of file `optixpp_namespace.h`.

2.17.2.18 `void optix::TextureSamplerObj::setMaxAnisotropy (float value) [inline]`

Set maximum anisotropy for this sampler. See `rtTextureSamplerSetMaxAnisotropy`.

Definition at line 2928 of file `optixpp_namespace.h`.

2.17.2.19 `void optix::TextureSamplerObj::setMipLevelCount (unsigned int num_mip_levels) [inline]`

Set the number of mip levels for this sampler. See `rtTextureSamplerSetMipLevelCount`.

Definition at line 2882 of file `optixpp_namespace.h`.

2.17.2.20 `void optix::TextureSamplerObj::setReadMode (RTtexturereadmode readmode) [inline]`

Set texture read mode for this sampler. See `rtTextureSamplerSetReadMode`.

Definition at line 2947 of file `optixpp_namespace.h`.

2.17.2.21 `void optix::TextureSamplerObj::setWrapMode (unsigned int dim, RTwrapmode wrapmode) [inline]`

Set the texture wrap mode for this sampler. See `rtTextureSamplerSetWrapMode`.

Definition at line 2906 of file `optixpp_namespace.h`.

2.17.2.22 `void optix::TextureSamplerObj::unregisterGLTexture () [inline]`

Unregister the texture's buffer, re-enabling OptiX operations. See `rtTextureSamplerGLUnregister`.

Definition at line 2993 of file `optixpp_namespace.h`.

2.17.2.23 `void optix::TextureSamplerObj::validate ()` [`inline`, `virtual`]

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements [optix::DestroyableObj](#).

Definition at line 2870 of file [optixpp_namespace.h](#).

2.17.3 Friends And Related Function Documentation

2.17.3.1 `friend class Handle< TextureSamplerObj >` [`friend`]

Definition at line 1405 of file [optixpp_namespace.h](#).

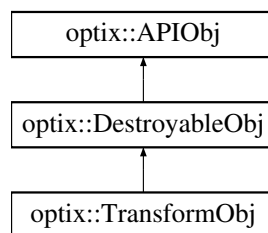
The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

2.18 `optix::TransformObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::TransformObj`:



Public Member Functions

- `void` [destroy](#) ()
- `void` [validate](#) ()
- `Context` [getContext](#) () `const`
- `RTtransform` [get](#) ()

Friends

- class [Handle< TransformObj >](#)
- `template<typename T >`
`void` [setChild](#) (T child)
- `template<typename T >`
T [getChild](#) () `const`

- void `setMatrix` (bool transpose, const float *matrix, const float *inverse_matrix)
- void `getMatrix` (bool transpose, float *matrix, float *inverse_matrix) const

2.18.1 Detailed Description

`Transform` wraps the OptiX C API `RTtransform` opaque type and its associated function set.

Definition at line 1012 of file `optixpp_namespace.h`.

2.18.2 Member Function Documentation

2.18.2.1 void `optix::TransformObj::destroy` () `[inline, virtual]`

call `rt[ObjectType]Destroy` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2438 of file `optixpp_namespace.h`.

2.18.2.2 `RTtransform` `optix::TransformObj::get` () `[inline]`

Get the underlying OptiX C API `RTtransform` opaque pointer.

Definition at line 2481 of file `optixpp_namespace.h`.

2.18.2.3 `template<typename T > T` `optix::TransformObj::getChild` () const `[inline]`

Set the child node of this transform. See `rtTransformGetChild`.

Definition at line 2464 of file `optixpp_namespace.h`.

2.18.2.4 `Context` `optix::TransformObj::getContext` () const `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 2450 of file `optixpp_namespace.h`.

2.18.2.5 void `optix::TransformObj::getMatrix` (bool *transpose*, float * *matrix*, float * *inverse_matrix*) const `[inline]`

Get the transform matrix for this node. See `rtTransformGetMatrix`.

Definition at line 2476 of file `optixpp_namespace.h`.

2.18.2.6 `template<typename T > void` `optix::TransformObj::setChild` (T *child*) `[inline]`

Set the child node of this transform. See `rtTransformSetChild`.

Definition at line 2458 of file `optixpp_namespace.h`.

2.18.2.7 `void optix::TransformObj::setMatrix (bool transpose, const float * matrix, const float * inverse_matrix) [inline]`

Set the transform matrix for this node. See `rtTransformSetMatrix`.

Definition at line 2471 of file `optixpp_namespace.h`.

2.18.2.8 `void optix::TransformObj::validate () [inline, virtual]`

call `rt[ObjectType]Validate` on the underlying OptiX C object

Implements `optix::DestroyableObj`.

Definition at line 2445 of file `optixpp_namespace.h`.

2.18.3 Friends And Related Function Documentation

2.18.3.1 `friend class Handle< TransformObj > [friend]`

Definition at line 1040 of file `optixpp_namespace.h`.

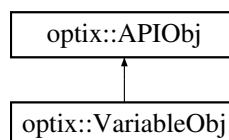
The documentation for this class was generated from the following file:

- `optixpp_namespace.h`

2.19 `optix::VariableObj` Class Reference

```
#include <optixpp_namespace.h>
```

Inheritance diagram for `optix::VariableObj`:



Public Member Functions

- `Context getContext () const`
- `std::string getName () const`
- `std::string getAnnotation () const`
- `RObjectType getType () const`
- `RTvariable get ()`
- `RTsize getSize () const`

Friends

- class [Handle](#)< [VariableObj](#) >

Float setters

Set variable to have a float value.

- void [setFloat](#) (float f1)
- void [setFloat](#) (optix::float2 f)
- void [setFloat](#) (float f1, float f2)
- void [setFloat](#) (optix::float3 f)
- void [setFloat](#) (float f1, float f2, float f3)
- void [setFloat](#) (optix::float4 f)
- void [setFloat](#) (float f1, float f2, float f3, float f4)
- void [set1fv](#) (const float *f)
- void [set2fv](#) (const float *f)
- void [set3fv](#) (const float *f)
- void [set4fv](#) (const float *f)

Int setters

Set variable to have an int value.

- void [setInt](#) (int i1)
- void [setInt](#) (int i1, int i2)
- void [setInt](#) (optix::int2 i)
- void [setInt](#) (int i1, int i2, int i3)
- void [setInt](#) (optix::int3 i)
- void [setInt](#) (int i1, int i2, int i3, int i4)
- void [setInt](#) (optix::int4 i)
- void [set1iv](#) (const int *i)
- void [set2iv](#) (const int *i)
- void [set3iv](#) (const int *i)
- void [set4iv](#) (const int *i)

Unsigned int setters

Set variable to have an unsigned int value.

- void [setUInt](#) (unsigned int u1)
- void [setUInt](#) (unsigned int u1, unsigned int u2)
- void [setUInt](#) (unsigned int u1, unsigned int u2, unsigned int u3)
- void [setUInt](#) (unsigned int u1, unsigned int u2, unsigned int u3, unsigned int u4)
- void [setUInt](#) (optix::uint2 u)

- void `setUint` (`optix::uint3` u)
- void `setUint` (`optix::uint4` u)
- void `set1uiv` (const unsigned int *u)
- void `set2uiv` (const unsigned int *u)
- void `set3uiv` (const unsigned int *u)
- void `set4uiv` (const unsigned int *u)

Matrix setters

Set variable to have a Matrix value

- void `setMatrix2x2fv` (bool transpose, const float *m)
- void `setMatrix2x3fv` (bool transpose, const float *m)
- void `setMatrix2x4fv` (bool transpose, const float *m)
- void `setMatrix3x2fv` (bool transpose, const float *m)
- void `setMatrix3x3fv` (bool transpose, const float *m)
- void `setMatrix3x4fv` (bool transpose, const float *m)
- void `setMatrix4x2fv` (bool transpose, const float *m)
- void `setMatrix4x3fv` (bool transpose, const float *m)
- void `setMatrix4x4fv` (bool transpose, const float *m)

Numeric value getters

Query value of a variable with scalar numeric value

- float `getFloat` () const
- unsigned int `getUint` () const
- int `getInt` () const

OptiX API object setters

Set variable to have an OptiX API object as its value

- void `setBuffer` (`Buffer` buffer)
- void `set` (`Buffer` buffer)
- void `setTextureSampler` (`TextureSampler` texturesample)
- void `set` (`TextureSampler` texturesample)
- void `set` (`GeometryGroup` group)
- void `set` (`Group` group)
- void `set` (`Program` program)
- void `set` (`Selector` selector)
- void `set` (`Transform` transform)

OptiX API object getters

Retrieve OptiX API object value from a variable

- [Buffer](#) `getBuffer () const`
- [TextureSampler](#) `getTextureSampler () const`
- [Program](#) `getProgram () const`

User data variable accessors

- void [setUserData](#) (RTsize size, const void *ptr)
- void [getUserData](#) (RTsize size, void *ptr) const

2.19.1 Detailed Description

Variable object wraps OptiX C API RTvariable type and its related function set.

See OptiX programming guide and API reference for complete description of the usage and behavior of RTvariable objects. Creation and querying of Variables can be performed via the [Handle::operator\[\]](#) function of the scope object associated with the variable. For example:

```
my_context["new_variable"]->setFloat( 1.0f );
```

will create a variable named `new_variable` on the object `my_context` if it does not already exist. It will then set the value of that variable to be a float 1.0f.

Definition at line 406 of file [optixpp_namespace.h](#).

2.19.2 Member Function Documentation

2.19.2.1 RTvariable `optix::VariableObj::get ()` [inline]

Get the OptiX C API object wrapped by this instance.

Definition at line 3550 of file [optixpp_namespace.h](#).

2.19.2.2 `std::string optix::VariableObj::getAnnotation () const` [inline]

Retrieve the annotation associated with the variable.

Definition at line 3536 of file [optixpp_namespace.h](#).

2.19.2.3 Buffer `optix::VariableObj::getBuffer () const` [inline]

Definition at line 3521 of file [optixpp_namespace.h](#).

2.19.2.4 `Context optix::VariableObj::getContext () const` `[inline, virtual]`

Retrieve the context this object is associated with. See `rt[ObjectType]GetContext`.

Implements `optix::APIObj`.

Definition at line 3227 of file `optixpp_namespace.h`.

2.19.2.5 `float optix::VariableObj::getFloat () const` `[inline]`

Definition at line 3445 of file `optixpp_namespace.h`.

2.19.2.6 `int optix::VariableObj::getInt () const` `[inline]`

Definition at line 3459 of file `optixpp_namespace.h`.

2.19.2.7 `std::string optix::VariableObj::getName () const` `[inline]`

Retrieve the name of the variable.

Definition at line 3529 of file `optixpp_namespace.h`.

2.19.2.8 `optix::Program optix::VariableObj::getProgram () const` `[inline]`

Definition at line 3570 of file `optixpp_namespace.h`.

2.19.2.9 `RTsize optix::VariableObj::getSize () const` `[inline]`

Get the size of the variable data in bytes (eg, `float4` returns `4*sizeof(float)`)

Definition at line 3555 of file `optixpp_namespace.h`.

2.19.2.10 `optix::TextureSampler optix::VariableObj::getTextureSampler () const` `[inline]`

Definition at line 3562 of file `optixpp_namespace.h`.

2.19.2.11 `RObjectType optix::VariableObj::getType () const` `[inline]`

Query the object type of the variable.

Definition at line 3543 of file `optixpp_namespace.h`.

2.19.2.12 `unsigned int optix::VariableObj::getUInt () const` `[inline]`

Definition at line 3452 of file `optixpp_namespace.h`.

2.19.2.13 `void optix::VariableObj::getUserData (RTsize size, void * ptr) const` `[inline]`

Retrieve a user defined type given the sizeof of the user object.

Definition at line 3481 of file `optixpp_namespace.h`.

2.19.2.14 void optix::VariableObj::set (Buffer *buffer*) [inline]

Definition at line 3471 of file [optixpp_namespace.h](#).

2.19.2.15 void optix::VariableObj::set (TextureSampler *texturesample*)

2.19.2.16 void optix::VariableObj::set (GeometryGroup *group*)

2.19.2.17 void optix::VariableObj::set (Group *group*)

2.19.2.18 void optix::VariableObj::set (Program *program*)

2.19.2.19 void optix::VariableObj::set (Selector *selector*)

2.19.2.20 void optix::VariableObj::set (Transform *transform*)

2.19.2.21 void optix::VariableObj::set1fv (const float * *f*) [inline]

Set variable value to a scalar float.

Definition at line 3369 of file [optixpp_namespace.h](#).

2.19.2.22 void optix::VariableObj::set1iv (const int * *i*) [inline]

Definition at line 3425 of file [optixpp_namespace.h](#).

2.19.2.23 void optix::VariableObj::set1uiv (const unsigned int * *u*) [inline]

Definition at line 3269 of file [optixpp_namespace.h](#).

2.19.2.24 void optix::VariableObj::set2fv (const float * *f*) [inline]

Set variable value to a float2.

Definition at line 3374 of file [optixpp_namespace.h](#).

2.19.2.25 void optix::VariableObj::set2iv (const int * *i*) [inline]

Definition at line 3430 of file [optixpp_namespace.h](#).

2.19.2.26 void optix::VariableObj::set2uiv (const unsigned int * *u*) [inline]

Definition at line 3274 of file [optixpp_namespace.h](#).

2.19.2.27 void optix::VariableObj::set3fv (const float * *f*) [inline]

Set variable value to a float3.

Definition at line 3379 of file [optixpp_namespace.h](#).

2.19.2.28 void optix::VariableObj::set3iv (const int * *i*) [inline]

Definition at line 3435 of file [optixpp_namespace.h](#).

2.19.2.29 `void optix::VariableObj::set3uiv (const unsigned int * u) [inline]`

Definition at line 3279 of file `optixpp_namespace.h`.

2.19.2.30 `void optix::VariableObj::set4fv (const float * f) [inline]`

Set variable value to a float4.

Definition at line 3384 of file `optixpp_namespace.h`.

2.19.2.31 `void optix::VariableObj::set4iv (const int * i) [inline]`

Definition at line 3440 of file `optixpp_namespace.h`.

2.19.2.32 `void optix::VariableObj::set4uiv (const unsigned int * u) [inline]`

Definition at line 3284 of file `optixpp_namespace.h`.

2.19.2.33 `void optix::VariableObj::setBuffer (Buffer buffer) [inline]`

Definition at line 3466 of file `optixpp_namespace.h`.

2.19.2.34 `void optix::VariableObj::setFloat (float f1) [inline]`

Set variable value to a scalar float.

Definition at line 3334 of file `optixpp_namespace.h`.

2.19.2.35 `void optix::VariableObj::setFloat (optix::float2 f) [inline]`

Set variable value to a float2.

Definition at line 3339 of file `optixpp_namespace.h`.

2.19.2.36 `void optix::VariableObj::setFloat (float f1, float f2) [inline]`

Set variable value to a float2.

Definition at line 3344 of file `optixpp_namespace.h`.

2.19.2.37 `void optix::VariableObj::setFloat (optix::float3 f) [inline]`

Set variable value to a float3.

Definition at line 3349 of file `optixpp_namespace.h`.

2.19.2.38 `void optix::VariableObj::setFloat (float f1, float f2, float f3) [inline]`

Set variable value to a float3.

Definition at line 3354 of file `optixpp_namespace.h`.

2.19.2.39 `void optix::VariableObj::setFloat (optix::float4 f) [inline]`

Set variable value to a float4.

Definition at line 3359 of file `optixpp_namespace.h`.

2.19.2.40 `void optix::VariableObj::setFloat (float f1, float f2, float f3, float f4)`
[inline]

Set variable value to a float4.

Definition at line 3364 of file `optixpp_namespace.h`.

2.19.2.41 `void optix::VariableObj::setInt (int i1)` [inline]

Definition at line 3390 of file `optixpp_namespace.h`.

2.19.2.42 `void optix::VariableObj::setInt (int i1, int i2)` [inline]

Definition at line 3400 of file `optixpp_namespace.h`.

2.19.2.43 `void optix::VariableObj::setInt (optix::int2 i)` [inline]

Definition at line 3395 of file `optixpp_namespace.h`.

2.19.2.44 `void optix::VariableObj::setInt (int i1, int i2, int i3)` [inline]

Definition at line 3410 of file `optixpp_namespace.h`.

2.19.2.45 `void optix::VariableObj::setInt (optix::int3 i)` [inline]

Definition at line 3405 of file `optixpp_namespace.h`.

2.19.2.46 `void optix::VariableObj::setInt (int i1, int i2, int i3, int i4)` [inline]

Definition at line 3420 of file `optixpp_namespace.h`.

2.19.2.47 `void optix::VariableObj::setInt (optix::int4 i)` [inline]

Definition at line 3415 of file `optixpp_namespace.h`.

2.19.2.48 `void optix::VariableObj::setMatrix2x2fv (bool transpose, const float * m)`
[inline]

Definition at line 3289 of file `optixpp_namespace.h`.

2.19.2.49 `void optix::VariableObj::setMatrix2x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3294 of file `optixpp_namespace.h`.

2.19.2.50 `void optix::VariableObj::setMatrix2x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3299 of file `optixpp_namespace.h`.

2.19.2.51 `void optix::VariableObj::setMatrix3x2fv (bool transpose, const float * m)`
[inline]

Definition at line 3304 of file `optixpp_namespace.h`.

2.19.2.52 `void optix::VariableObj::setMatrix3x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3309 of file `optixpp_namespace.h`.

2.19.2.53 `void optix::VariableObj::setMatrix3x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3314 of file `optixpp_namespace.h`.

2.19.2.54 `void optix::VariableObj::setMatrix4x2fv (bool transpose, const float * m)`
[inline]

Definition at line 3319 of file `optixpp_namespace.h`.

2.19.2.55 `void optix::VariableObj::setMatrix4x3fv (bool transpose, const float * m)`
[inline]

Definition at line 3324 of file `optixpp_namespace.h`.

2.19.2.56 `void optix::VariableObj::setMatrix4x4fv (bool transpose, const float * m)`
[inline]

Definition at line 3329 of file `optixpp_namespace.h`.

2.19.2.57 `void optix::VariableObj::setTextureSampler (TextureSampler texturesample)` [inline]

Definition at line 3486 of file `optixpp_namespace.h`.

2.19.2.58 `void optix::VariableObj::setUInt (unsigned int u1)` [inline]

Definition at line 3234 of file `optixpp_namespace.h`.

2.19.2.59 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2)`
[inline]

Definition at line 3239 of file `optixpp_namespace.h`.

2.19.2.60 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2, unsigned int u3)` [inline]

Definition at line 3244 of file `optixpp_namespace.h`.

2.19.2.61 `void optix::VariableObj::setUInt (unsigned int u1, unsigned int u2, unsigned int u3, unsigned int u4)` [inline]

Definition at line 3249 of file [optixpp_namespace.h](#).

2.19.2.62 `void optix::VariableObj::setUInt (optix::uint2 u)` [inline]

Definition at line 3254 of file [optixpp_namespace.h](#).

2.19.2.63 `void optix::VariableObj::setUInt (optix::uint3 u)` [inline]

Definition at line 3259 of file [optixpp_namespace.h](#).

2.19.2.64 `void optix::VariableObj::setUInt (optix::uint4 u)` [inline]

Definition at line 3264 of file [optixpp_namespace.h](#).

2.19.2.65 `void optix::VariableObj::setUserData (RTsize size, const void * ptr)` [inline]

Set the variable to a user defined type given the sizeof the user object.

Definition at line 3476 of file [optixpp_namespace.h](#).

2.19.3 Friends And Related Function Documentation

2.19.3.1 `friend class Handle< VariableObj >` [friend]

Definition at line 593 of file [optixpp_namespace.h](#).

The documentation for this class was generated from the following file:

- [optixpp_namespace.h](#)

3 File Documentation

3.1 optixpp_namespace.h File Reference

```
#include "../optix.h"    #include "../optix_gl_interop.h" ×
#include "../optix_cuda_interop.h"    #include <string> ×
#include <vector> #include <iterator> #include "optixu-
_vector_types.h"
```

Classes

- class [optix::Handle< T >](#)
The *Handle* class is a reference counted handle class used to manipulate API objects.
- class [optix::Exception](#)

Exception class for error reporting from the OptiXpp API.

- class `optix::APIObj`
Base class for all reference counted wrappers around OptiX C API opaque types.
- class `optix::DestroyableObj`
Base class for all wrapper objects which can be destroyed and validated.
- class `optix::ScopedObj`
Base class for all objects which are OptiX variable containers.
- class `optix::VariableObj`
Variable object wraps OptiX C API RTvariable type and its related function set.
- class `optix::ContextObj`
Context object wraps the OptiX C API RTcontext opaque type and its associated function set.
- class `optix::ProgramObj`
Program object wraps the OptiX C API RTprogram opaque type and its associated function set.
- class `optix::GroupObj`
Group wraps the OptiX C API RTgroup opaque type and its associated function set.
- class `optix::GeometryGroupObj`
GeometryGroup wraps the OptiX C API RTgeometrygroup opaque type and its associated function set.
- class `optix::TransformObj`
Transform wraps the OptiX C API RTtransform opaque type and its associated function set.
- class `optix::SelectorObj`
Selector wraps the OptiX C API RTselector opaque type and its associated function set.
- class `optix::AccelerationObj`
Acceleration wraps the OptiX C API RTacceleration opaque type and its associated function set.
- class `optix::GeometryInstanceObj`
GeometryInstance wraps the OptiX C API RTgeometryinstance acceleration opaque type and its associated function set.
- class `optix::GeometryObj`
Geometry wraps the OptiX C API RTgeometry opaque type and its associated function set.
- class `optix::MaterialObj`
Material wraps the OptiX C API RTmaterial opaque type and its associated function set.
- class `optix::TextureSamplerObj`
TextureSampler wraps the OptiX C API RTtexturesampler opaque type and its associated function set.
- class `optix::BufferObj`
Buffer wraps the OptiX C API RTbuffer opaque type and its associated function set.

Typedefs

- typedef Handle< AccelerationObj > [optix::Acceleration](#)
- typedef Handle< BufferObj > [optix::Buffer](#)
- typedef Handle< ContextObj > [optix::Context](#)
- typedef Handle< GeometryObj > [optix::Geometry](#)
- typedef Handle< GeometryGroupObj > [optix::GeometryGroup](#)
- typedef Handle< GeometryInstanceObj > [optix::GeometryInstance](#)
- typedef Handle< GroupObj > [optix::Group](#)
- typedef Handle< MaterialObj > [optix::Material](#)
- typedef Handle< ProgramObj > [optix::Program](#)
- typedef Handle< SelectorObj > [optix::Selector](#)
- typedef Handle< TextureSamplerObj > [optix::TextureSampler](#)
- typedef Handle< TransformObj > [optix::Transform](#)
- typedef Handle< VariableObj > [optix::Variable](#)

3.1.1 Detailed Description

A C++ wrapper around the OptiX API.

Definition in file [optixpp_namespace.h](#).

3.2 optixpp_namespace.h

```

00001
00002 /*
00003  * Copyright (c) 2008 - 2009 NVIDIA Corporation. All rights reserved.
00004  *
00005  * NVIDIA Corporation and its licensors retain all intellectual property and
00006  * rights in and to this software, related documentation and any modifications
00007  * thereto.
00008  * Any use, reproduction, disclosure or distribution of this software and
00009  * related
00010  * documentation without an express license agreement from NVIDIA Corporation
00011  * is strictly
00012  * prohibited.
00013  *
00014  * TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THIS SOFTWARE IS PROVIDED
00015  * *AS IS*
00016  * AND NVIDIA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EITHER EXPRESS OR
00017  * IMPLIED,
00018  * INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND
00019  * FITNESS FOR A
00020  * PARTICULAR PURPOSE. IN NO EVENT SHALL NVIDIA OR ITS SUPPLIERS BE LIABLE FOR
00021  * ANY
00022  * SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER
00023  * (INCLUDING, WITHOUT
00024  * LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION,
00025  * LOSS OF
00026  * BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF
00027  * OR
00028  * INABILITY TO USE THIS SOFTWARE, EVEN IF NVIDIA HAS BEEN ADVISED OF THE
00029  * POSSIBILITY OF
00030  * SUCH DAMAGES
00031  */
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057

```

```

00058
00059 #ifndef __optixu_optixpp_namespace_h__
00060 #define __optixu_optixpp_namespace_h__
00061
00062 #include "../optix.h"
00063
00064 #ifdef _WIN32
00065 #   ifndef WIN32_LEAN_AND_MEAN
00066 #       define WIN32_LEAN_AND_MEAN
00067 #   endif
00068 #   include <windows.h>
00069 #   include "../optix_d3d9_interop.h"
00070 #   include "../optix_d3d10_interop.h"
00071 #   include "../optix_d3d11_interop.h"
00072 #endif
00073 #include "../optix_gl_interop.h"
00074 #include "../optix_cuda_interop.h"
00075
00076 #include <string>
00077 #include <vector>
00078 #include <iterator>
00079 #include "optixu_vector_types.h"
00080
00081 //-----
00082 //
00083 // Doxygen group specifications
00084 //
00085 //-----
00086
00087 //-----
00088 //
00089 // C++ API
00090 //
00091 //-----
00092
00093 namespace optix {
00094
00095     class AccelerationObj;
00096     class BufferObj;
00097     class ContextObj;
00098     class GeometryObj;
00099     class GeometryGroupObj;
00100     class GeometryInstanceObj;
00101     class GroupObj;
00102     class MaterialObj;
00103     class ProgramObj;
00104     class SelectorObj;
00105     class TextureSamplerObj;
00106     class TransformObj;
00107     class VariableObj;
00108
00109     class APIObj;
00110     class ScopedObj;
00111
00112     template<class T>
00123     class Handle {
00124     public:
00126         Handle() : ptr(0) {}
00127
00129         Handle(T* ptr) : ptr(ptr) { ref(); }
00130
00132         template<class U>
00133         Handle(U* ptr) : ptr(ptr) { ref(); }
00134
00136         Handle(const Handle<T>& copy) : ptr(copy.ptr) { ref(); }
00137
00139         template<class U>
00140         Handle(const Handle<U>& copy) : ptr(copy.ptr) { ref(); }
00141
00143         Handle<T>& operator=(const Handle<T>& copy)
00144         { if(ptr != copy.ptr) { unref(); ptr = copy.ptr; ref(); } return *this; }
00145
00147         template<class U>

```

```

00148     Handle<T>& operator=( const Handle<U>& copy)
00149     { if(ptr != copy.ptr) { unref(); ptr = copy.ptr; ref(); } return *this; }
00150
00152     ~Handle() { unref(); }
00153
00155     static Handle<T> take( typename T::api_t p ) { return p? new T(p) : 0; }
00158     static Handle<T> take( RObject p ) { return p? new T(static_cast<typename
T::api_t>(p)) : 0; }
00159
00161     T* operator->()           { return ptr; }
00162     const T* operator->() const { return ptr; }
00163
00165     T* get()                 { return ptr; }
00166     const T* get() const    { return ptr; }
00167
00169     operator bool() const   { return ptr != 0; }
00170
00174     Handle<VariableObj> operator[](const std::string& varname);
00175
00194     Handle<VariableObj> operator[](const char* varname);
00195
00197     static Handle<T> create() { return T::create(); }
00198
00200     static unsigned int getDeviceCount() { return T::getDeviceCount(); }
00201
00202 private:
00203     inline void ref() { if(ptr) ptr->addReference(); }
00204     inline void unref() { if(ptr && ptr->removeReference() == 0) delete ptr; }
00205     T* ptr;
00206 };
00207
00208
00209 //
-----
00210
00211 typedef Handle<AccelerationObj>   Acceleration;
00212 typedef Handle<BufferObj>         Buffer;
00213 typedef Handle<ContextObj>        Context;
00214 typedef Handle<GeometryObj>       Geometry;
00215 typedef Handle<GeometryGroupObj>  GeometryGroup;
00216 typedef Handle<GeometryInstanceObj> GeometryInstance;
00217 typedef Handle<GroupObj>          Group;
00218 typedef Handle<MaterialObj>       Material;
00219 typedef Handle<ProgramObj>        Program;
00220 typedef Handle<SelectorObj>       Selector;
00221 typedef Handle<TextureSamplerObj> TextureSampler;
00222 typedef Handle<TransformObj>      Transform;
00223 typedef Handle<VariableObj>       Variable;
00224
00225
00226 //
-----
00227
00228
00235 class Exception: public std::exception {
00236 public:
00238     Exception( const std::string& message, RResult error_code =
RT_ERROR_UNKNOWN )
00239     : m_message(message), m_error_code( error_code ) {}
00240
00243     virtual ~Exception() throw() {}
00244
00246     const std::string& getErrorString() const { return m_message; }
00247
00249     RResult getErrorCode() const { return m_error_code; }
00250
00253     static Exception makeException( RResult code, RTcontext context );
00254
00256     virtual const char* what() const throw() { return getErrorString().c_str(); }
}
00257 private:
00258     std::string m_message;
00259     RResult     m_error_code;
00260 };

```

```

00261
00262 inline Exception Exception::makeException( RResult code, RTcontext context )
00263 {
00264     const char* str;
00265     rtContextGetErrorString( context, code, &str);
00266     return Exception( std::string(str), code );
00267 }
00268
00269
00270 //
-----
00271
00272
00291 class APIObj {
00292 public:
00293     APIObj() : ref_count(0) {}
00294     virtual ~APIObj() {}
00295
00297     void addReference() { ++ref_count; }
00299     int removeReference() { return --ref_count; }
00300
00302     virtual Context getContext() const=0;
00303
00306     virtual void checkError(RResult code) const;
00307     virtual void checkError(RResult code, Context context) const;
00308
00309     void checkErrorNoGetContext(RResult code) const;
00310
00312     static Exception makeException( RResult code, RTcontext context );
00313 private:
00314     int ref_count;
00315 };
00316
00317 inline Exception APIObj::makeException( RResult code, RTcontext context )
00318 {
00319     return Exception::makeException( code, context );
00320 }
00321
00322
00323 //
-----
00324
00325
00341 class DestroyableObj : public APIObj {
00342 public:
00343     virtual ~DestroyableObj() {}
00344
00346     virtual void destroy() = 0;
00347
00349     virtual void validate() = 0;
00350 };
00351
00352
00353
00354 //
-----
00355
00356
00367 class ScopedObj : public DestroyableObj {
00368 public:
00369     virtual ~ScopedObj() {}
00370
00373     virtual Variable declareVariable(const std::string& name) = 0;
00376     virtual Variable queryVariable(const std::string& name) const = 0;
00378     virtual void removeVariable(Variable v) = 0;
00382     virtual unsigned int getVariableCount() const = 0;
00384     virtual Variable getVariable(unsigned int index) const = 0;
00385 };
00386
00387
00388
00389 //
-----
00390

```

```
00391
00406 class VariableObj : public APIObj {
00407 public:
00408
00409     Context getContext() const;
00410
00413
00414
00415     void setFloat(float f1);
00417     void setFloat(optix::float2 f);
00419     void setFloat(float f1, float f2);
00421     void setFloat(optix::float3 f);
00423     void setFloat(float f1, float f2, float f3);
00425     void setFloat(optix::float4 f);
00427     void setFloat(float f1, float f2, float f3, float f4);
00429     void set1fv(const float* f);
00431     void set2fv(const float* f);
00433     void set3fv(const float* f);
00435     void set4fv(const float* f);
00437
00440
00441     void setInt(int i1);
00442     void setInt(int i1, int i2);
00443     void setInt(optix::int2 i);
00444     void setInt(int i1, int i2, int i3);
00445     void setInt(optix::int3 i);
00446     void setInt(int i1, int i2, int i3, int i4);
00447     void setInt(optix::int4 i);
00448     void set1iv(const int* i);
00449     void set2iv(const int* i);
00450     void set3iv(const int* i);
00451     void set4iv(const int* i);
00453
00456
00457     void setUInt(unsigned int u1);
00458     void setUInt(unsigned int u1, unsigned int u2);
00459     void setUInt(unsigned int u1, unsigned int u2, unsigned int u3);
00460     void setUInt(unsigned int u1, unsigned int u2, unsigned int u3, unsigned
int u4);
00461     void setUInt(optix::uint2 u);
00462     void setUInt(optix::uint3 u);
00463     void setUInt(optix::uint4 u);
00464     void set1uiv(const unsigned int* u);
00465     void set2uiv(const unsigned int* u);
00466     void set3uiv(const unsigned int* u);
00467     void set4uiv(const unsigned int* u);
00469
00472
00473     void setMatrix2x2fv(bool transpose, const float* m);
00474     void setMatrix2x3fv(bool transpose, const float* m);
00475     void setMatrix2x4fv(bool transpose, const float* m);
00476     void setMatrix3x2fv(bool transpose, const float* m);
00477     void setMatrix3x3fv(bool transpose, const float* m);
00478     void setMatrix3x4fv(bool transpose, const float* m);
00479     void setMatrix4x2fv(bool transpose, const float* m);
00480     void setMatrix4x3fv(bool transpose, const float* m);
00481     void setMatrix4x4fv(bool transpose, const float* m);
00483
00486
00487     float getFloat() const;
00488     unsigned int getUInt() const;
00489     int getInt() const;
00491
00492 #if 0
00493     // Not implemented yet...
00494
00495     // The getFloat functions can be overloaded by parameter type.
00496     void getFloat(float* f);
00497     void getFloat(float* f1, float* f2);
00498     void getFloat(optix::float2* f);
00499     void getFloat(float* f1, float* f2, float* f3);
00500     void getFloat(optix::float3* f);
00501     void getFloat(float* f1, float* f2, float* f3, float* f4);
00502     void getFloat(optix::float4* f);
```

```

00503     // This one will need a different name to distinguish it from 'float
    getFloat()'.
00504     optix::float2 getFloat2();
00505     optix::float3 getFloat3();
00506     optix::float4 getFloat4();
00507
00508     void get1fv(float* f);
00509     void get2fv(float* f);
00510     void get3fv(float* f);
00511     void get4fv(float* f);
00512
00513     get1i (int* i1);
00514     get2i (int* i1, int* i2);
00515     get3i (int* i1, int* i2, int* i3);
00516     get4i (int* i1, int* i2, int* i3, int* i4);
00517     get1iv(int* i);
00518     get2iv(int* i);
00519     get3iv(int* i);
00520     get4iv(int* i);
00521
00522     get1ui (unsigned int* u1);
00523     get2ui (unsigned int* u1, unsigned int* u2);
00524     get3ui (unsigned int* u1, unsigned int* u2, unsigned int* u3);
00525     get4ui (unsigned int* u1, unsigned int* u2, unsigned int* u3, unsigned int*
00526     u4);
00527     get1uiv(unsigned int* u);
00528     get2uiv(unsigned int* u);
00529     get3uiv(unsigned int* u);
00530     get4uiv(unsigned int* u);
00531
00531     getMatrix2x2fv(bool transpose, float* m);
00532     getMatrix2x3fv(bool transpose, float* m);
00533     getMatrix2x4fv(bool transpose, float* m);
00534     getMatrix3x2fv(bool transpose, float* m);
00535     getMatrix3x3fv(bool transpose, float* m);
00536     getMatrix3x4fv(bool transpose, float* m);
00537     getMatrix4x2fv(bool transpose, float* m);
00538     getMatrix4x3fv(bool transpose, float* m);
00539     getMatrix4x4fv(bool transpose, float* m);
00540 #endif
00541
00542
00543
00544
00545
00546     void setBuffer(Buffer buffer);
00547     void set(Buffer buffer);
00548     void setTextureSampler(TextureSampler texturesample);
00549     void set(TextureSampler texturesample);
00550     void set(GeometryGroup group);
00551     void set(Group group);
00552     void set(Program program);
00553     void set(Selector selector);
00554     void set(Transform transform);
00555
00556
00557
00558
00559
00560     Buffer getBuffer() const;
00561     TextureSampler getTextureSampler() const;
00562     Program getProgram() const;
00563
00564
00565
00566
00567
00568     void setUserData(RTsize size, const void* ptr);
00569     void getUserData(RTsize size, void* ptr) const;
00570
00571
00572
00573
00574     std::string getName() const;
00575
00576
00577     std::string getAnnotation() const;
00578
00579
00580     RTobjecttype getType() const;
00581
00582
00583     RTvariable get();
00584
00585
00586     RTsize getSize() const;
00587
00588     private:

```

```

00589     typedef RTvariable api_t;
00590
00591     RTvariable m_variable;
00592     VariableObj(RTvariable variable) : m_variable(variable) {}
00593     friend class Handle<VariableObj>;
00594
00595 };
00596
00597 template<class T>
00598 Handle<VariableObj> Handle<T>::operator[](const std::string& varname)
00599 {
00600     Variable v = ptr->queryVariable( varname );
00601     if( v.operator->() == 0)
00602         v = ptr->declareVariable( varname );
00603     return v;
00604 }
00605
00606 template<class T>
00607 Handle<VariableObj> Handle<T>::operator[](const char* varname)
00608 {
00609     return (*this)[ std::string( varname ) ];
00610 }
00611
00612
00613 //
-----
00614
00615
00619 class ContextObj : public ScopedObj {
00620 public:
00621
00623     static unsigned int getDeviceCount();
00624
00626     static std::string getDeviceName(int ordinal);
00627
00629     static void getDeviceAttribute(int ordinal, RTdeviceattribute attrib,
RTsize size, void* p);
00630
00632     static Context create();
00633
00635     void destroy();
00636
00638     void validate();
00639
00642     Context getContext() const;
00643
00646     void checkError(RTresult code) const;
00647
00649     std::string getErrorString( RTresult code ) const;
00651
00654     Acceleration createAcceleration(const char* builder, const char* traverser)
;
00655
00657     Buffer createBuffer(unsigned int type);
00659     Buffer createBuffer(unsigned int type, RTformat format);
00662     Buffer createBuffer(unsigned int type, RTformat format, RTsize width);
00665     Buffer createBuffer(unsigned int type, RTformat format, RTsize width,
RTsize height);
00668     Buffer createBuffer(unsigned int type, RTformat format, RTsize width,
RTsize height, RTsize depth);
00669
00671     Buffer createBufferForCUDA(unsigned int type);
00673     Buffer createBufferForCUDA(unsigned int type, RTformat format);
00676     Buffer createBufferForCUDA(unsigned int type, RTformat format, RTsize width
);
00679     Buffer createBufferForCUDA(unsigned int type, RTformat format, RTsize width
, RTsize height);
00682     Buffer createBufferForCUDA(unsigned int type, RTformat format, RTsize width
, RTsize height, RTsize depth);
00683
00685     Buffer createBufferFromGLBO(unsigned int type, unsigned int vbo);
00686
00688     TextureSampler createTextureSamplerFromGLImage(unsigned int id, RTgltarget
target);

```

```

00689
00690 #ifdef _WIN32
00691
00692     Buffer createBufferFromD3D9Resource(unsigned int type, IDirect3DResource9 *
pResource);
00694     Buffer createBufferFromD3D10Resource(unsigned int type, ID3D10Resource *
pResource);
00696     Buffer createBufferFromD3D11Resource(unsigned int type, ID3D11Resource *
pResource);
00697
00699     TextureSampler createTextureSamplerFromD3D9Resource(IDirect3DResource9 *
pResource);
00701     TextureSampler createTextureSamplerFromD3D10Resource(ID3D10Resource *
pResource);
00703     TextureSampler createTextureSamplerFromD3D11Resource(ID3D11Resource *
pResource);
00704 #endif
00705
00707     Geometry createGeometry();
00709     GeometryInstance createGeometryInstance();
00712     template<class Iterator>
00713     GeometryInstance createGeometryInstance( Geometry geometry, Iterator
matlbegin, Iterator matlend );
00714
00716     Group createGroup();
00719     template<class Iterator>
00720     Group createGroup( Iterator childbegin, Iterator childend );
00721
00723     GeometryGroup createGeometryGroup();
00726     template<class Iterator>
00727     GeometryGroup createGeometryGroup( Iterator childbegin, Iterator childend )
;
00728
00730     Transform createTransform();
00731
00733     Material createMaterial();
00734
00736     Program createProgramFromPTXFile ( const std::string& ptx, const
std::string& program_name );
00738     Program createProgramFromPTXString( const std::string& ptx, const
std::string& program_name );
00739
00741     Selector createSelector();
00742
00744     TextureSampler createTextureSampler();
00746
00749     template<class Iterator>
00750     void setDevices(Iterator begin, Iterator end);
00751
00752 #ifdef _WIN32
00753
00754     void setD3D9Device(IDirect3DDevice9* device);
00756     void setD3D10Device(ID3D10Device* device);
00758     void setD3D11Device(ID3D11Device* device);
00759 #endif
00760
00762     std::vector<int> getEnabledDevices() const;
00763
00766     unsigned int getEnabledDeviceCount() const;
00768
00771     int getMaxTextureCount() const;
00772
00774     int getCPUNumThreads() const;
00775
00777     RTsize getUsedHostMemory() const;
00778
00780     int getGPUPagingActive() const;
00781
00783     int getGPUPagingForcedOff() const;
00784
00786     RTsize getAvailableDeviceMemory(int ordinal) const;
00788
00791     void setCPUNumThreads(int cpu_num_threads);
00792

```



```

00794     void setGPUPagingForcedOff(int gpu_paging_forced_off);
00796
00799     void setStackSize(RTsize stack_size_bytes);
00801     RTsize getStackSize() const;
00802
00805     void setTimeoutCallback(RTtimeoutcallback callback, double
min_polling_seconds);
00806
00808     void setEntryPointCount(unsigned int num_entry_points);
00810     unsigned int getEntryPointCount() const;
00811
00813     void setRayTypeCount(unsigned int num_ray_types);
00815     unsigned int getRayTypeCount() const;
00817
00820     void setRayGenerationProgram(unsigned int entry_point_index, Program
program);
00822     Program getRayGenerationProgram(unsigned int entry_point_index) const;
00823
00825     void setExceptionProgram(unsigned int entry_point_index, Program program);
00827     Program getExceptionProgram(unsigned int entry_point_index) const;
00828
00830     void setExceptionEnabled( RTexception exception, bool enabled );
00832     bool getExceptionEnabled( RTexception exception ) const;
00833
00835     void setMissProgram(unsigned int ray_type_index, Program program);
00837     Program getMissProgram(unsigned int ray_type_index) const;
00839
00841     void compile();
00842
00845     void launch(unsigned int entry_point_index, RTsize image_width);
00847     void launch(unsigned int entry_point_index, RTsize image_width, RTsize
image_height);
00849     void launch(unsigned int entry_point_index, RTsize image_width, RTsize
image_height, RTsize image_depth);
00851
00853     int getRunningState() const;
00854
00857     void setPrintEnabled(bool enabled);
00859     bool getPrintEnabled() const;
00861     void setPrintBufferSize(RTsize buffer_size_bytes);
00863     RTsize getPrintBufferSize() const;
00865     void setPrintLaunchIndex(int x, int y=-1, int z=-1);
00867     optix::int3 getPrintLaunchIndex() const;
00869
00871     Variable declareVariable (const std::string& name);
00872     Variable queryVariable (const std::string& name) const;
00873     void removeVariable (Variable v);
00874     unsigned int getVariableCount() const;
00875     Variable getVariable (unsigned int index) const;
00877
00879     RTcontext get();
00880 private:
00881     typedef RTcontext api_t;
00882
00883     virtual ~ContextObj() {}
00884     RTcontext m_context;
00885     ContextObj(RTcontext context) : m_context(context) {}
00886     friend class Handle<ContextObj>;
00887 };
00888
00889
00890 //
-----
00891
00892
00896 class ProgramObj : public ScopedObj {
00897 public:
00898     void destroy();
00899     void validate();
00900
00901     Context getContext() const;
00902
00903     Variable declareVariable (const std::string& name);
00904     Variable queryVariable (const std::string& name) const;

```

```

00905     void    removeVariable (Variable v);
00906     unsigned int getVariableCount() const;
00907     Variable getVariable (unsigned int index) const;
00908
00909     RTprogram get();
00910 private:
00911     typedef RTprogram api_t;
00912     virtual ~ProgramObj() {}
00913     RTprogram m_program;
00914     ProgramObj(RTprogram program) : m_program(program) {}
00915     friend class Handle<ProgramObj>;
00916 };
00917
00918
00919 //
-----
00920
00921
00925 class GroupObj : public DestroyableObj {
00926 public:
00927     void destroy();
00928     void validate();
00929
00930     Context getContext() const;
00931
00932     void setAcceleration(Acceleration acceleration);
00933     Acceleration getAcceleration() const;
00934
00935     void setChildCount(unsigned int count);
00936     unsigned int getChildCount() const;
00937
00938     template< typename T > void setChild(unsigned int index, T child);
00939     template< typename T > T getChild(unsigned int index) const;
00940
00941     RTgroup get();
00942 private:
00943     typedef RTgroup api_t;
00944     virtual ~GroupObj() {}
00945     RTgroup m_group;
00946     GroupObj(RTgroup group) : m_group(group) {}
00947     friend class Handle<GroupObj>;
00948 };
00949
00950 //
-----
00964
00965
00969 class GeometryGroupObj : public DestroyableObj {
00970 public:
00971     void destroy();
00972     void validate();
00973     Context getContext() const;
00974
00975     void setAcceleration(Acceleration acceleration);
00976     Acceleration getAcceleration() const;
00977
00978     void setChildCount(unsigned int count);
00979     unsigned int getChildCount() const;
00980
00981     void setChild(unsigned int index, GeometryInstance geometryinstance);
00982     GeometryInstance getChild(unsigned int index) const;
00983
00984     RTgeometrygroup get();
00985 private:
00986     typedef RTgeometrygroup api_t;
00987     virtual ~GeometryGroupObj() {}
00988     RTgeometrygroup m_geometrygroup;
00989     GeometryGroupObj(RTgeometrygroup geometrygroup) : m_geometrygroup(
00990     geometrygroup) {}
00991     friend class Handle<GeometryGroupObj>;
00992 };
00993

```

```

01004
01005
01006 //
-----
01007
01008
01012 class TransformObj : public DestroyableObj {
01013 public:
01014     void destroy();
01015     void validate();
01016     Context getContext() const;
01017
01020     template< typename T > void setChild(T child);
01022     template< typename T > T getChild() const;
01024
01027     void setMatrix(bool transpose, const float* matrix, const float*
inverse_matrix);
01029     void getMatrix(bool transpose, float* matrix, float* inverse_matrix) const;
01031
01033     RTtransform get();
01034
01035 private:
01036     typedef RTtransform api_t;
01037     virtual ~TransformObj() {}
01038     RTtransform m_transform;
01039     TransformObj(RTtransform transform) : m_transform(transform) {}
01040     friend class Handle<TransformObj>;
01041 };
01042
01043
01044 //
-----
01045
01046
01050 class SelectorObj : public DestroyableObj {
01051 public:
01052     void destroy();
01053     void validate();
01054     Context getContext() const;
01055
01058     void setVisitProgram(Program program);
01060     Program getVisitProgram() const;
01062
01065     void setChildCount(unsigned int count);
01067     unsigned int getChildCount() const;
01068
01070     template< typename T > void setChild(unsigned int index, T child);
01072     template< typename T > T getChild(unsigned int index) const;
01074
01076     Variable declareVariable (const std::string& name);
01077     Variable queryVariable (const std::string& name) const;
01078     void removeVariable (Variable v);
01079     unsigned int getVariableCount() const;
01080     Variable getVariable (unsigned int index) const;
01082
01084     RTselector get();
01085
01086 private:
01087     typedef RTselector api_t;
01088     virtual ~SelectorObj() {}
01089     RTselector m_selector;
01090     SelectorObj(RTselector selector) : m_selector(selector) {}
01091     friend class Handle<SelectorObj>;
01092 };
01093
01094
01095 //
-----
01096
01097
01101 class AccelerationObj : public DestroyableObj {
01102 public:
01103     void destroy();
01104     void validate();

```

```

01105     Context getContext() const;
01106
01109     void markDirty();
01111     bool isDirty() const;
01113
01117     void          setProperty( const std::string& name, const std::string& value
);
01120 );
01121     std::string getProperty( const std::string& name ) const;
01123
01123     void          setBuilder(const std::string& builder);
01125     std::string  getBuilder() const;
01127     void          setTraverser(const std::string& traverser);
01129     std::string  getTraverser() const;
01131
01134     RTsize getDataSize() const;
01136     void      getData( void* data ) const;
01138     void      setData( const void* data, RTsize size );
01140
01142     RTacceleration get();
01143
01144 private:
01145     typedef RTacceleration api_t;
01146     virtual ~AccelerationObj() {}
01147     RTacceleration m_acceleration;
01148     AccelerationObj(RTacceleration acceleration) : m_acceleration(acceleration)
{}
01149     friend class Handle<AccelerationObj>;
01150 };
01151
01152
01153 //
-----
01154
01155
01160 class GeometryInstanceObj : public ScopedObj {
01161 public:
01162     void destroy();
01163     void validate();
01164     Context getContext() const;
01165
01168     void setGeometry(Geometry geometry);
01170     Geometry getGeometry() const;
01171
01173     void setMaterialCount(unsigned int count);
01175     unsigned int getMaterialCount() const;
01176
01178     void setMaterial(unsigned int idx, Material material);
01180     Material getMaterial(unsigned int idx) const;
01181
01183     unsigned int addMaterial(Material material);
01185
01187     Variable declareVariable (const std::string& name);
01188     Variable queryVariable   (const std::string& name) const;
01189     void      removeVariable (Variable v);
01190     unsigned int getVariableCount() const;
01191     Variable getVariable   (unsigned int index) const;
01193
01195     RTgeometryinstance get();
01196
01197 private:
01198     typedef RTgeometryinstance api_t;
01199     virtual ~GeometryInstanceObj() {}
01200     RTgeometryinstance m_geometryinstance;
01201     GeometryInstanceObj(RTgeometryinstance geometryinstance) :
m_geometryinstance(geometryinstance) {}
01202     friend class Handle<GeometryInstanceObj>;
01203 };
01204
01205
01206 //
-----
01207
01208
01212 class GeometryObj : public ScopedObj {

```

```

01213 public:
01214     void destroy();
01215     void validate();
01216     Context getContext() const;
01217
01220     void markDirty();
01222     bool isDirty() const;
01224
01228     void setPrimitiveCount(unsigned int num_primitives);
01231     unsigned int getPrimitiveCount() const;
01233
01236     void setBoundingBoxProgram(Program program);
01238     Program getBoundingBoxProgram() const;
01239
01241     void setIntersectionProgram(Program program);
01243     Program getIntersectionProgram() const;
01245
01247     Variable declareVariable(const std::string& name);
01248     Variable queryVariable(const std::string& name) const;
01249     void removeVariable(Variable v);
01250     unsigned int getVariableCount() const;
01251     Variable getVariable(unsigned int index) const;
01253
01255     RTgeometry get();
01256
01257 private:
01258     typedef RTgeometry api_t;
01259     virtual ~GeometryObj() {}
01260     RTgeometry m_geometry;
01261     GeometryObj(RTgeometry geometry) : m_geometry(geometry) {}
01262     friend class Handle<GeometryObj>;
01263 };
01264
01265 //
01266 //-----
01267
01268
01272 class MaterialObj : public ScopedObj {
01273 public:
01274     void destroy();
01275     void validate();
01276     Context getContext() const;
01277
01280     void setClosestHitProgram(unsigned int ray_type_index, Program program);
01282     Program getClosestHitProgram(unsigned int ray_type_index) const;
01283
01285     void setAnyHitProgram(unsigned int ray_type_index, Program program);
01287     Program getAnyHitProgram(unsigned int ray_type_index) const;
01289
01291     Variable declareVariable(const std::string& name);
01292     Variable queryVariable(const std::string& name) const;
01293     void removeVariable(Variable v);
01294     unsigned int getVariableCount() const;
01295     Variable getVariable(unsigned int index) const;
01297
01299     RTmaterial get();
01300 private:
01301     typedef RTmaterial api_t;
01302     virtual ~MaterialObj() {}
01303     RTmaterial m_material;
01304     MaterialObj(RTmaterial material) : m_material(material) {}
01305     friend class Handle<MaterialObj>;
01306 };
01307
01308 //
01309 //-----
01310
01311
01315 class TextureSamplerObj : public DestroyableObj {
01316 public:
01317     void destroy();
01318     void validate();

```

```

01319     Context getContext() const;
01320
01323     void setMipLevelCount(unsigned int num_mip_levels);
01325     unsigned int getMipLevelCount() const;
01326
01328     void setArraySize(unsigned int num_textures_in_array);
01330     unsigned int getArraySize() const;
01331
01333     void setWrapMode(unsigned int dim, RTwrapmode wrapmode);
01335     RTwrapmode getWrapMode(unsigned int dim) const;
01336
01338     void setFilteringModes(RTfiltermode minification, RTfiltermode
magnification, RTfiltermode mipmapping);
01340     void getFilteringModes(RTfiltermode& minification, RTfiltermode&
magnification, RTfiltermode& mipmapping) const;
01341
01343     void setMaxAnisotropy(float value);
01345     float getMaxAnisotropy() const;
01346
01348     void setReadMode(RTtexturereadmode readmode);
01350     RTtexturereadmode getReadMode() const;
01351
01353     void setIndexingMode(RTtextureindexmode indexmode);
01355     RTtextureindexmode getIndexingMode() const;
01357
01360     int getId() const;
01362
01365     void setBuffer(unsigned int texture_array_idx, unsigned int mip_level,
Buffer buffer);
01367     Buffer getBuffer(unsigned int texture_array_idx, unsigned int mip_level)
const;
01369
01371     RTtexturesampler get();
01372
01375     void registerGLTexture();
01377     void unregisterGLTexture();
01379
01380 #ifdef _WIN32
01381
01384     void registerD3D9Texture();
01386     void registerD3D10Texture();
01388     void registerD3D11Texture();
01389
01391     void unregisterD3D9Texture();
01393     void unregisterD3D10Texture();
01395     void unregisterD3D11Texture();
01397
01398 #endif
01399
01400     private:
01401         typedef RTtexturesampler api_t;
01402         virtual ~TextureSamplerObj() {}
01403         RTtexturesampler m_texturesampler;
01404         TextureSamplerObj(RTtexturesampler texturesampler) : m_texturesampler(
texturesampler) {}
01405         friend class Handle<TextureSamplerObj>;
01406     };
01407
01408
01409     //
-----
01410
01411
01415     class BufferObj : public DestroyableObj {
01416     public:
01417         void destroy();
01418         void validate();
01419         Context getContext() const;
01420
01423         void setFormat(RTformat format);
01425         RTformat getFormat() const;
01426
01428         void setElementSize(RTsize size_of_element);
01430         RTsize getElementSize() const;

```

```

01431
01432     void getDevicePointer( unsigned int optix_device_number, CUdeviceptr *
device_pointer );
01434
01435     void setDevicePointer( unsigned int optix_device_number, CUdeviceptr
device_pointer );
01437
01438     void markDirty();
01439
01440
01441     void setSize(RTsize width);
01442     void getSize(RTsize& width) const;
01443     void setSize(RTsize width, RTsize height);
01444     void getSize(RTsize& width, RTsize& height) const;
01445     void setSize(RTsize width, RTsize height, RTsize depth);
01446     void getSize(RTsize& width, RTsize& height, RTsize& depth) const;
01447
01448     void setSize(unsigned int dimensionality, const RTsize* dims);
01449     void getSize(unsigned int dimensionality, RTsize* dims) const;
01450
01451     unsigned int getDimensionality() const;
01452
01453     unsigned int getGLBOId() const;
01454
01455     void registerGLBuffer();
01456     void unregisterGLBuffer();
01457
01458     #ifdef _WIN32
01459     void registerD3D9Buffer();
01460     void registerD3D10Buffer();
01461     void registerD3D11Buffer();
01462
01463     void unregisterD3D9Buffer();
01464     void unregisterD3D10Buffer();
01465     void unregisterD3D11Buffer();
01466
01467     IDirect3DResource9* getD3D9Resource();
01468     ID3D10Resource* getD3D10Resource();
01469     ID3D11Resource* getD3D11Resource();
01470
01471     #endif
01472
01473     void* map();
01474     void unmap();
01475
01476     RTbuffer get();
01477
01478     private:
01479     typedef RTbuffer api_t;
01480     virtual ~BufferObj() {}
01481     RTbuffer m_buffer;
01482     BufferObj(RTbuffer buffer) : m_buffer(buffer) {}
01483     friend class Handle<BufferObj>;
01484 };
01485
01486 //
01487
01488 -----
01489
01490
01491 inline void APIObj::checkError( RTresult code ) const
01492 {
01493     if( code != RT_SUCCESS) {
01494         RTcontext c = this->getContext()->get();
01495         throw Exception::makeException( code, c );
01496     }
01497 }
01498
01499 inline void APIObj::checkError( RTresult code, Context context ) const
01500 {
01501     if( code != RT_SUCCESS) {
01502         RTcontext c = context->get();
01503         throw Exception::makeException( code, c );
01504     }
01505 }

```

```

01537 }
01538
01539 inline void APIObj::checkErrorNoGetContext( RResult code ) const
01540 {
01541     if( code != RT_SUCCESS ) {
01542         throw Exception::makeException( code, 0u );
01543     }
01544 }
01545
01546 inline Context ContextObj::getContext() const
01547 {
01548     return Context::take( m_context );
01549 }
01550
01551 inline void ContextObj::checkError(RResult code) const
01552 {
01553     if( code != RT_SUCCESS && code != RT_TIMEOUT_CALLBACK )
01554         throw Exception::makeException( code, m_context );
01555 }
01556
01557 inline unsigned int ContextObj::getDeviceCount()
01558 {
01559     unsigned int count;
01560     if( RResult code = rtDeviceGetDeviceCount(&count) )
01561         throw Exception::makeException( code, 0 );
01562
01563     return count;
01564 }
01565
01566 inline std::string ContextObj::getDeviceName(int ordinal)
01567 {
01568     const RTsize max_string_size = 256;
01569     char name[max_string_size];
01570     if( RResult code = rtDeviceGetAttribute(ordinal, RT_DEVICE_ATTRIBUTE_NAME,
01571                                             max_string_size, name) )
01572         throw Exception::makeException( code, 0 );
01573     return std::string(name);
01574 }
01575
01576 inline void ContextObj::getDeviceAttribute(int ordinal, RTdeviceattribute
attrib, RTsize size, void* p)
01577 {
01578     if( RResult code = rtDeviceGetAttribute(ordinal, attrib, size, p) )
01579         throw Exception::makeException( code, 0 );
01580 }
01581
01582 inline Context ContextObj::create()
01583 {
01584     RTcontext c;
01585     if( RResult code = rtContextCreate(&c) )
01586         throw Exception::makeException( code, 0 );
01587
01588     return Context::take(c);
01589 }
01590
01591 inline void ContextObj::destroy()
01592 {
01593     checkErrorNoGetContext( rtContextDestroy( m_context ) );
01594     m_context = 0;
01595 }
01596
01597 inline void ContextObj::validate()
01598 {
01599     checkError( rtContextValidate( m_context ) );
01600 }
01601
01602 inline Acceleration ContextObj::createAcceleration(const char* builder, const
char* traverser)
01603 {
01604     RTacceleration acceleration;
01605     checkError( rtAccelerationCreate( m_context, &acceleration ) );
01606     checkError( rtAccelerationSetBuilder( acceleration, builder ) );
01607     checkError( rtAccelerationSetTraverser( acceleration, traverser ) );
01608     return Acceleration::take(acceleration);

```



```
01609 }
01610
01611
01612 inline Buffer ContextObj::createBuffer(unsigned int type)
01613 {
01614     RTbuffer buffer;
01615     checkError( rtBufferCreate( m_context, type, &buffer ) );
01616     return Buffer::take(buffer);
01617 }
01618
01619 inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format)
01620 {
01621     RTbuffer buffer;
01622     checkError( rtBufferCreate( m_context, type, &buffer ) );
01623     checkError( rtBufferSetFormat( buffer, format ) );
01624     return Buffer::take(buffer);
01625 }
01626
01627 inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format,
RTsize width)
01628 {
01629     RTbuffer buffer;
01630     checkError( rtBufferCreate( m_context, type, &buffer ) );
01631     checkError( rtBufferSetFormat( buffer, format ) );
01632     checkError( rtBufferSetSize1D( buffer, width ) );
01633     return Buffer::take(buffer);
01634 }
01635
01636 inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format,
RTsize width, RTsize height)
01637 {
01638     RTbuffer buffer;
01639     checkError( rtBufferCreate( m_context, type, &buffer ) );
01640     checkError( rtBufferSetFormat( buffer, format ) );
01641     checkError( rtBufferSetSize2D( buffer, width, height ) );
01642     return Buffer::take(buffer);
01643 }
01644
01645 inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format,
RTsize width, RTsize height, RTsize depth)
01646 {
01647     RTbuffer buffer;
01648     checkError( rtBufferCreate( m_context, type, &buffer ) );
01649     checkError( rtBufferSetFormat( buffer, format ) );
01650     checkError( rtBufferSetSize3D( buffer, width, height, depth ) );
01651     return Buffer::take(buffer);
01652 }
01653
01654 inline Buffer ContextObj::createBufferForCUDA(unsigned int type)
01655 {
01656     RTbuffer buffer;
01657     checkError( rtBufferCreateForCUDA( m_context, type, &buffer ) );
01658     return Buffer::take(buffer);
01659 }
01660
01661 inline Buffer ContextObj::createBufferForCUDA(unsigned int type, RTformat
format)
01662 {
01663     RTbuffer buffer;
01664     checkError( rtBufferCreateForCUDA( m_context, type, &buffer ) );
01665     checkError( rtBufferSetFormat( buffer, format ) );
01666     return Buffer::take(buffer);
01667 }
01668
01669 inline Buffer ContextObj::createBufferForCUDA(unsigned int type, RTformat
format, RTsize width)
01670 {
01671     RTbuffer buffer;
01672     checkError( rtBufferCreateForCUDA( m_context, type, &buffer ) );
01673     checkError( rtBufferSetFormat( buffer, format ) );
01674     checkError( rtBufferSetSize1D( buffer, width ) );
01675     return Buffer::take(buffer);
01676 }
01677
```

```
01678 inline Buffer ContextObj::createBufferForCUDA(unsigned int type, RTformat
format, RTsize width, RTsize height)
01679 {
01680     RTbuffer buffer;
01681     checkError( rtBufferCreateForCUDA( m_context, type, &buffer ) );
01682     checkError( rtBufferSetFormat( buffer, format ) );
01683     checkError( rtBufferSetSize2D( buffer, width, height ) );
01684     return Buffer::take(buffer);
01685 }
01686
01687 inline Buffer ContextObj::createBufferForCUDA(unsigned int type, RTformat
format, RTsize width, RTsize height, RTsize depth)
01688 {
01689     RTbuffer buffer;
01690     checkError( rtBufferCreateForCUDA( m_context, type, &buffer ) );
01691     checkError( rtBufferSetFormat( buffer, format ) );
01692     checkError( rtBufferSetSize3D( buffer, width, height, depth ) );
01693     return Buffer::take(buffer);
01694 }
01695
01696 inline Buffer ContextObj::createBufferFromGLBO(unsigned int type, unsigned
int vbo)
01697 {
01698     RTbuffer buffer;
01699     checkError( rtBufferCreateFromGLBO( m_context, type, vbo, &buffer ) );
01700     return Buffer::take(buffer);
01701 }
01702
01703 #ifdef _WIN32
01704
01705 inline Buffer ContextObj::createBufferFromD3D9Resource(unsigned int type,
IDirect3DResource9 *pResource)
01706 {
01707     RTbuffer buffer;
01708     checkError( rtBufferCreateFromD3D9Resource( m_context, type, pResource, &
buffer ) );
01709     return Buffer::take(buffer);
01710 }
01711
01712 inline Buffer ContextObj::createBufferFromD3D10Resource(unsigned int type,
ID3D10Resource *pResource)
01713 {
01714     RTbuffer buffer;
01715     checkError( rtBufferCreateFromD3D10Resource( m_context, type, pResource, &
buffer ) );
01716     return Buffer::take(buffer);
01717 }
01718
01719 inline Buffer ContextObj::createBufferFromD3D11Resource(unsigned int type,
ID3D11Resource *pResource)
01720 {
01721     RTbuffer buffer;
01722     checkError( rtBufferCreateFromD3D11Resource( m_context, type, pResource, &
buffer ) );
01723     return Buffer::take(buffer);
01724 }
01725
01726 inline TextureSampler ContextObj::createTextureSamplerFromD3D9Resource(
IDirect3DResource9 *pResource)
01727 {
01728     RTtexturesampler textureSampler;
01729     checkError( rtTextureSamplerCreateFromD3D9Resource( m_context, pResource, &
textureSampler ) );
01730     return TextureSampler::take(textureSampler);
01731 }
01732
01733 inline TextureSampler ContextObj::createTextureSamplerFromD3D10Resource(
ID3D10Resource *pResource)
01734 {
01735     RTtexturesampler textureSampler;
01736     checkError( rtTextureSamplerCreateFromD3D10Resource( m_context, pResource, &
textureSampler ) );
01737     return TextureSampler::take(textureSampler);
01738 }
```

```

01739
01740 inline TextureSampler ContextObj::createTextureSamplerFromD3D11Resource(
    ID3D11Resource *pResource)
01741 {
01742     RTtexturesampler textureSampler;
01743     checkError( rtTextureSamplerCreateFromD3D11Resource(m_context, pResource, &
    textureSampler));
01744     return TextureSampler::take(textureSampler);
01745 }
01746
01747 inline void ContextObj::setD3D9Device(IDirect3DDevice9* device)
01748 {
01749     checkError( rtContextSetD3D9Device( m_context, device ) );
01750 }
01751
01752 inline void ContextObj::setD3D10Device(ID3D10Device* device)
01753 {
01754     checkError( rtContextSetD3D10Device( m_context, device ) );
01755 }
01756
01757 inline void ContextObj::setD3D11Device(ID3D11Device* device)
01758 {
01759     checkError( rtContextSetD3D11Device( m_context, device ) );
01760 }
01761
01762 #endif
01763
01764 inline TextureSampler ContextObj::createTextureSamplerFromGLImage(unsigned
    int id, RTgltarget target)
01765 {
01766     RTtexturesampler textureSampler;
01767     checkError( rtTextureSamplerCreateFromGLImage(m_context, id, target, &
    textureSampler));
01768     return TextureSampler::take(textureSampler);
01769 }
01770
01771 inline Geometry ContextObj::createGeometry()
01772 {
01773     RTgeometry geometry;
01774     checkError( rtGeometryCreate( m_context, &geometry ) );
01775     return Geometry::take(geometry);
01776 }
01777
01778 inline GeometryInstance ContextObj::createGeometryInstance()
01779 {
01780     RTgeometryinstance geometryinstance;
01781     checkError( rtGeometryInstanceCreate( m_context, &geometryinstance ) );
01782     return GeometryInstance::take(geometryinstance);
01783 }
01784
01785 template<class Iterator>
01786 GeometryInstance ContextObj::createGeometryInstance( Geometry geometry,
    Iterator matlbegin, Iterator matlend)
01787 {
01788     GeometryInstance result = createGeometryInstance();
01789     result->setGeometry( geometry );
01790     unsigned int count = 0;
01791     for( Iterator iter = matlbegin; iter != matlend; ++iter )
01792         ++count;
01793     result->setMaterialCount( count );
01794     unsigned int index = 0;
01795     for(Iterator iter = matlbegin; iter != matlend; ++iter, ++index )
01796         result->setMaterial( index, *iter );
01797     return result;
01798 }
01799
01800 inline Group ContextObj::createGroup()
01801 {
01802     RTgroup group;
01803     checkError( rtGroupCreate( m_context, &group ) );
01804     return Group::take(group);
01805 }
01806
01807 template<class Iterator>

```

```

01808     inline Group ContextObj::createGroup( Iterator childbegin, Iterator
childend )
01809     {
01810         Group result = createGroup();
01811         unsigned int count = 0;
01812         for(Iterator iter = childbegin; iter != childend; ++iter )
01813             ++count;
01814         result->setChildCount( count );
01815         unsigned int index = 0;
01816         for(Iterator iter = childbegin; iter != childend; ++iter, ++index )
01817             result->setChild( index, *iter );
01818         return result;
01819     }
01820
01821     inline GeometryGroup ContextObj::createGeometryGroup()
01822     {
01823         RTgeometrygroup gg;
01824         checkError( rtGeometryGroupCreate( m_context, &gg ) );
01825         return GeometryGroup::take( gg );
01826     }
01827
01828     template<class Iterator>
01829     inline GeometryGroup ContextObj::createGeometryGroup( Iterator childbegin,
Iterator childend )
01830     {
01831         GeometryGroup result = createGeometryGroup();
01832         unsigned int count = 0;
01833         for(Iterator iter = childbegin; iter != childend; ++iter )
01834             ++count;
01835         result->setChildCount( count );
01836         unsigned int index = 0;
01837         for(Iterator iter = childbegin; iter != childend; ++iter, ++index )
01838             result->setChild( index, *iter );
01839         return result;
01840     }
01841
01842     inline Transform ContextObj::createTransform()
01843     {
01844         RTtransform t;
01845         checkError( rtTransformCreate( m_context, &t ) );
01846         return Transform::take( t );
01847     }
01848
01849     inline Material ContextObj::createMaterial()
01850     {
01851         RTmaterial material;
01852         checkError( rtMaterialCreate( m_context, &material ) );
01853         return Material::take(material);
01854     }
01855
01856     inline Program ContextObj::createProgramFromPTXFile( const std::string&
filename, const std::string& program_name )
01857     {
01858         RTprogram program;
01859         checkError( rtProgramCreateFromPTXFile( m_context, filename.c_str(),
program_name.c_str(), &program ) );
01860         return Program::take(program);
01861     }
01862
01863     inline Program ContextObj::createProgramFromPTXString( const std::string& ptx
, const std::string& program_name )
01864     {
01865         RTprogram program;
01866         checkError( rtProgramCreateFromPTXString( m_context, ptx.c_str(),
program_name.c_str(), &program ) );
01867         return Program::take(program);
01868     }
01869
01870     inline Selector ContextObj::createSelector()
01871     {
01872         RTselector selector;
01873         checkError( rtSelectorCreate( m_context, &selector ) );
01874         return Selector::take(selector);
01875     }

```

```

01876
01877 inline TextureSampler ContextObj::createTextureSampler()
01878 {
01879     RTtexturesampler texturesampler;
01880     checkError( rtTextureSamplerCreate( m_context, &texturesampler ) );
01881     return TextureSampler::take(texturesampler);
01882 }
01883
01884 inline std::string ContextObj::getErrorString( RTresult code ) const
01885 {
01886     const char* str;
01887     rtContextGetErrorString( m_context, code, &str);
01888     return std::string(str);
01889 }
01890
01891 template<class Iterator> inline
01892 void ContextObj::setDevices(Iterator begin, Iterator end)
01893 {
01894     std::vector<int> devices;
01895     std::copy( begin, end, std::insert_iterator<std::vector<int> >( devices,
devices.begin() ) );
01896     checkError( rtContextSetDevices( m_context, static_cast<unsigned int>(
devices.size()), &devices[0]) );
01897 }
01898
01899 inline std::vector<int> ContextObj::getEnabledDevices() const
01900 {
01901     // Initialize with the number of enabled devices
01902     std::vector<int> devices(getEnabledDeviceCount());
01903     checkError( rtContextGetDevices( m_context, &devices[0] ) );
01904     return devices;
01905 }
01906
01907 inline unsigned int ContextObj::getEnabledDeviceCount() const
01908 {
01909     unsigned int num;
01910     checkError( rtContextGetDeviceCount( m_context, &num ) );
01911     return num;
01912 }
01913
01914 inline int ContextObj::getMaxTextureCount() const
01915 {
01916     int tex_count;
01917     checkError( rtContextGetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_MAX_TEXTURE_COUNT, sizeof(tex_count), &tex_count) );
01918     return tex_count;
01919 }
01920
01921 inline int ContextObj::getCPUNumThreads() const
01922 {
01923     int cpu_num_threads;
01924     checkError( rtContextGetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS, sizeof(cpu_num_threads), &cpu_num_threads) );
01925     return cpu_num_threads;
01926 }
01927
01928 inline RTsize ContextObj::getUsedHostMemory() const
01929 {
01930     RTsize used_mem;
01931     checkError( rtContextGetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_USED_HOST_MEMORY, sizeof(used_mem), &used_mem) );
01932     return used_mem;
01933 }
01934
01935 inline int ContextObj::getGPUPagingActive() const
01936 {
01937     int gpu_paging_active;
01938     checkError( rtContextGetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_GPU_PAGING_ACTIVE, sizeof(gpu_paging_active), &gpu_paging_active) );
01939     return gpu_paging_active;
01940 }
01941
01942 inline int ContextObj::getGPUPagingForcedOff() const
01943 {

```

```

01944     int gpu_paging_forced_off;
01945     checkError( rtContextGetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF, sizeof(gpu_paging_forced_off), &gpu_paging_forced_off) );
01946     return gpu_paging_forced_off;
01947 }
01948
01949 inline RTsize ContextObj::getAvailableDeviceMemory(int ordinal) const
01950 {
01951     RTsize free_mem;
01952     checkError( rtContextGetAttribute( m_context,
01953         static_cast<RTcontextattribute>(
RT_CONTEXT_ATTRIBUTE_AVAILABLE_DEVICE_MEMORY + ordinal),
01954         sizeof(free_mem), &free_mem) );
01955     return free_mem;
01956 }
01957
01958 inline void ContextObj::setCPUNumThreads(int cpu_num_threads)
01959 {
01960     checkError( rtContextSetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS, sizeof(cpu_num_threads), &cpu_num_threads) );
01961 }
01962
01963 inline void ContextObj::setGPUPagingForcedOff(int gpu_paging_forced_off)
01964 {
01965     checkError( rtContextSetAttribute( m_context,
RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF, sizeof(gpu_paging_forced_off), &gpu_paging_forced_off) );
01966 }
01967
01968 inline void ContextObj::setStackSize(RTsize stack_size_bytes)
01969 {
01970     checkError( rtContextSetStackSize( m_context, stack_size_bytes) );
01971 }
01972
01973 inline RTsize ContextObj::getStackSize() const
01974 {
01975     RTsize result;
01976     checkError( rtContextGetStackSize( m_context, &result ) );
01977     return result;
01978 }
01979
01980 inline void ContextObj::setTimeoutCallback(RTtimeoutcallback callback, double
min_polling_seconds)
01981 {
01982     checkError( rtContextSetTimeoutCallback( m_context, callback,
min_polling_seconds ) );
01983 }
01984
01985 inline void ContextObj::setEntryPointCount(unsigned int num_entry_points)
01986 {
01987     checkError( rtContextSetEntryPointCount( m_context, num_entry_points ) );
01988 }
01989
01990 inline unsigned int ContextObj::getEntryPointCount() const
01991 {
01992     unsigned int result;
01993     checkError( rtContextGetEntryPointCount( m_context, &result ) );
01994     return result;
01995 }
01996
01997
01998 inline void ContextObj::setRayGenerationProgram(unsigned int
entry_point_index, Program program)
01999 {
02000     checkError( rtContextSetRayGenerationProgram( m_context, entry_point_index,
program->get() ) );
02001 }
02002
02003 inline Program ContextObj::getRayGenerationProgram(unsigned int
entry_point_index) const
02004 {
02005     RTprogram result;
02006     checkError( rtContextGetRayGenerationProgram( m_context, entry_point_index,
&result ) );
02007     return Program::take( result );

```

```

02008 }
02009
02010
02011 inline void ContextObj::setExceptionProgram(unsigned int entry_point_index,
Program program)
02012 {
02013     checkError( rtContextSetExceptionProgram( m_context, entry_point_index,
program->get() ) );
02014 }
02015
02016 inline Program ContextObj::getExceptionProgram(unsigned int entry_point_index
) const
02017 {
02018     RTprogram result;
02019     checkError( rtContextGetExceptionProgram( m_context, entry_point_index, &
result ) );
02020     return Program::take( result );
02021 }
02022
02023
02024 inline void ContextObj::setExceptionEnabled( RTextception exception, bool
enabled )
02025 {
02026     checkError( rtContextSetExceptionEnabled( m_context, exception, enabled ) )
;
02027 }
02028
02029 inline bool ContextObj::getExceptionEnabled( RTextception exception ) const
02030 {
02031     int enabled;
02032     checkError( rtContextGetExceptionEnabled( m_context, exception, &enabled )
);
02033     return enabled != 0;
02034 }
02035
02036
02037 inline void ContextObj::setRayTypeCount(unsigned int num_ray_types)
02038 {
02039     checkError( rtContextSetRayTypeCount( m_context, num_ray_types ) );
02040 }
02041
02042 inline unsigned int ContextObj::getRayTypeCount() const
02043 {
02044     unsigned int result;
02045     checkError( rtContextGetRayTypeCount( m_context, &result ) );
02046     return result;
02047 }
02048
02049 inline void ContextObj::setMissProgram(unsigned int ray_type_index, Program
program)
02050 {
02051     checkError( rtContextSetMissProgram( m_context, ray_type_index, program->get
() ) );
02052 }
02053
02054 inline Program ContextObj::getMissProgram(unsigned int ray_type_index) const
02055 {
02056     RTprogram result;
02057     checkError( rtContextGetMissProgram( m_context, ray_type_index, &result ) )
;
02058     return Program::take( result );
02059 }
02060
02061 inline void ContextObj::compile()
02062 {
02063     checkError( rtContextCompile( m_context ) );
02064 }
02065
02066 inline void ContextObj::launch(unsigned int entry_point_index, RTsize
image_width)
02067 {
02068     checkError( rtContextLaunch1D( m_context, entry_point_index, image_width )
);
02069 }

```

```

02070
02071 inline void ContextObj::launch(unsigned int entry_point_index, RTsize
image_width, RTsize image_height)
02072 {
02073     checkError( rtContextLaunch2D( m_context, entry_point_index, image_width,
image_height ) );
02074 }
02075
02076 inline void ContextObj::launch(unsigned int entry_point_index, RTsize
image_width, RTsize image_height, RTsize image_depth)
02077 {
02078     checkError( rtContextLaunch3D( m_context, entry_point_index, image_width,
image_height, image_depth ) );
02079 }
02080
02081
02082 inline int ContextObj::getRunningState() const
02083 {
02084     int result;
02085     checkError( rtContextGetRunningState( m_context, &result ) );
02086     return result;
02087 }
02088
02089 inline void ContextObj::setPrintEnabled(bool enabled)
02090 {
02091     checkError( rtContextSetPrintEnabled( m_context, enabled ) );
02092 }
02093
02094 inline bool ContextObj::getPrintEnabled() const
02095 {
02096     int enabled;
02097     checkError( rtContextGetPrintEnabled( m_context, &enabled ) );
02098     return enabled != 0;
02099 }
02100
02101 inline void ContextObj::setPrintBufferSize(RTsize buffer_size_bytes)
02102 {
02103     checkError( rtContextSetPrintBufferSize( m_context, buffer_size_bytes ) );
02104 }
02105
02106 inline RTsize ContextObj::getPrintBufferSize() const
02107 {
02108     RTsize result;
02109     checkError( rtContextGetPrintBufferSize( m_context, &result ) );
02110     return result;
02111 }
02112
02113 inline void ContextObj::setPrintLaunchIndex(int x, int y, int z)
02114 {
02115     checkError( rtContextSetPrintLaunchIndex( m_context, x, y, z ) );
02116 }
02117
02118 inline optix::int3 ContextObj::getPrintLaunchIndex() const
02119 {
02120     optix::int3 result;
02121     checkError( rtContextGetPrintLaunchIndex( m_context, &result.x, &result.y,
&result.z ) );
02122     return result;
02123 }
02124
02125 inline Variable ContextObj::declareVariable(const std::string& name)
02126 {
02127     RTvariable v;
02128     checkError( rtContextDeclareVariable( m_context, name.c_str(), &v ) );
02129     return Variable::take( v );
02130 }
02131
02132 inline Variable ContextObj::queryVariable(const std::string& name) const
02133 {
02134     RTvariable v;
02135     checkError( rtContextQueryVariable( m_context, name.c_str(), &v ) );
02136     return Variable::take( v );
02137 }
02138

```



```
02139 inline void ContextObj::removeVariable(Variable v)
02140 {
02141     checkError( rtContextRemoveVariable( m_context, v->get() ) );
02142 }
02143
02144 inline unsigned int ContextObj::getVariableCount() const
02145 {
02146     unsigned int result;
02147     checkError( rtContextGetVariableCount( m_context, &result ) );
02148     return result;
02149 }
02150
02151 inline Variable ContextObj::getVariable(unsigned int index) const
02152 {
02153     RTvariable v;
02154     checkError( rtContextGetVariable( m_context, index, &v ) );
02155     return Variable::take( v );
02156 }
02157
02158
02159 inline RTcontext ContextObj::get()
02160 {
02161     return m_context;
02162 }
02163
02164 inline void ProgramObj::destroy()
02165 {
02166     Context context = getContext();
02167     checkError( rtProgramDestroy( m_program ), context );
02168     m_program = 0;
02169 }
02170
02171 inline void ProgramObj::validate()
02172 {
02173     checkError( rtProgramValidate( m_program ) );
02174 }
02175
02176 inline Context ProgramObj::getContext() const
02177 {
02178     RTcontext c;
02179     checkErrorNoGetContext( rtProgramGetContext( m_program, &c ) );
02180     return Context::take( c );
02181 }
02182
02183 inline Variable ProgramObj::declareVariable(const std::string& name)
02184 {
02185     RTvariable v;
02186     checkError( rtProgramDeclareVariable( m_program, name.c_str(), &v ) );
02187     return Variable::take( v );
02188 }
02189
02190 inline Variable ProgramObj::queryVariable(const std::string& name) const
02191 {
02192     RTvariable v;
02193     checkError( rtProgramQueryVariable( m_program, name.c_str(), &v ) );
02194     return Variable::take( v );
02195 }
02196
02197 inline void ProgramObj::removeVariable(Variable v)
02198 {
02199     checkError( rtProgramRemoveVariable( m_program, v->get() ) );
02200 }
02201
02202 inline unsigned int ProgramObj::getVariableCount() const
02203 {
02204     unsigned int result;
02205     checkError( rtProgramGetVariableCount( m_program, &result ) );
02206     return result;
02207 }
02208
02209 inline Variable ProgramObj::getVariable(unsigned int index) const
02210 {
02211     RTvariable v;
02212     checkError( rtProgramGetVariable( m_program, index, &v ) );
```

```

02213     return Variable::take(v);
02214 }
02215
02216 inline RTprogram ProgramObj::get()
02217 {
02218     return m_program;
02219 }
02220
02221 inline void GroupObj::destroy()
02222 {
02223     Context context = getContext();
02224     checkError( rtGroupDestroy( m_group ), context );
02225     m_group = 0;
02226 }
02227
02228 inline void GroupObj::validate()
02229 {
02230     checkError( rtGroupValidate( m_group ) );
02231 }
02232
02233 inline Context GroupObj::getContext() const
02234 {
02235     RTcontext c;
02236     checkErrorNoGetContext( rtGroupGetContext( m_group, &c ) );
02237     return Context::take(c);
02238 }
02239
02240 inline void SelectorObj::destroy()
02241 {
02242     Context context = getContext();
02243     checkError( rtSelectorDestroy( m_selector ), context );
02244     m_selector = 0;
02245 }
02246
02247 inline void SelectorObj::validate()
02248 {
02249     checkError( rtSelectorValidate( m_selector ) );
02250 }
02251
02252 inline Context SelectorObj::getContext() const
02253 {
02254     RTcontext c;
02255     checkErrorNoGetContext( rtSelectorGetContext( m_selector, &c ) );
02256     return Context::take( c );
02257 }
02258
02259 inline void SelectorObj::setVisitProgram(Program program)
02260 {
02261     checkError( rtSelectorSetVisitProgram( m_selector, program->get() ) );
02262 }
02263
02264 inline Program SelectorObj::getVisitProgram() const
02265 {
02266     RTprogram result;
02267     checkError( rtSelectorGetVisitProgram( m_selector, &result ) );
02268     return Program::take( result );
02269 }
02270
02271 inline void SelectorObj::setChildCount(unsigned int count)
02272 {
02273     checkError( rtSelectorSetChildCount( m_selector, count ) );
02274 }
02275
02276 inline unsigned int SelectorObj::getChildCount() const
02277 {
02278     unsigned int result;
02279     checkError( rtSelectorGetChildCount( m_selector, &result ) );
02280     return result;
02281 }
02282
02283 template< typename T >
02284 inline void SelectorObj::setChild(unsigned int index, T child)
02285 {
02286     checkError( rtSelectorSetChild( m_selector, index, child->get() ) );

```

```

02287 }
02288
02289 template< typename T >
02290 inline T SelectorObj::getChild(unsigned int index) const
02291 {
02292     RObject result;
02293     checkError( rtSelectorGetChild( m_selector, index, &result ) );
02294     return T::take( result );
02295 }
02296
02297 inline Variable SelectorObj::declareVariable(const std::string& name)
02298 {
02299     RTvariable v;
02300     checkError( rtSelectorDeclareVariable( m_selector, name.c_str(), &v ) );
02301     return Variable::take( v );
02302 }
02303
02304 inline Variable SelectorObj::queryVariable(const std::string& name) const
02305 {
02306     RTvariable v;
02307     checkError( rtSelectorQueryVariable( m_selector, name.c_str(), &v ) );
02308     return Variable::take( v );
02309 }
02310
02311 inline void SelectorObj::removeVariable(Variable v)
02312 {
02313     checkError( rtSelectorRemoveVariable( m_selector, v->get() ) );
02314 }
02315
02316 inline unsigned int SelectorObj::getVariableCount() const
02317 {
02318     unsigned int result;
02319     checkError( rtSelectorGetVariableCount( m_selector, &result ) );
02320     return result;
02321 }
02322
02323 inline Variable SelectorObj::getVariable(unsigned int index) const
02324 {
02325     RTvariable v;
02326     checkError( rtSelectorGetVariable( m_selector, index, &v ) );
02327     return Variable::take( v );
02328 }
02329
02330 inline RTselector SelectorObj::get()
02331 {
02332     return m_selector;
02333 }
02334
02335 inline void GroupObj::setAcceleration(Acceleration acceleration)
02336 {
02337     checkError( rtGroupSetAcceleration( m_group, acceleration->get() ) );
02338 }
02339
02340 inline Acceleration GroupObj::getAcceleration() const
02341 {
02342     RTacceleration result;
02343     checkError( rtGroupGetAcceleration( m_group, &result ) );
02344     return Acceleration::take( result );
02345 }
02346
02347 inline void GroupObj::setChildCount(unsigned int count)
02348 {
02349     checkError( rtGroupSetChildCount( m_group, count ) );
02350 }
02351
02352 inline unsigned int GroupObj::getChildCount() const
02353 {
02354     unsigned int result;
02355     checkError( rtGroupGetChildCount( m_group, &result ) );
02356     return result;
02357 }
02358
02359 template< typename T >
02360 inline void GroupObj::setChild(unsigned int index, T child)

```

```

02361 {
02362     checkError( rtGroupSetChild( m_group, index, child->get() ) );
02363 }
02364
02365 template< typename T >
02366 inline T GroupObj::getChild( unsigned int index ) const
02367 {
02368     RObject result;
02369     checkError( rtGroupGetChild( m_group, index, &result ) );
02370     return T::take( result );
02371 }
02372
02373 inline RTgroup GroupObj::get()
02374 {
02375     return m_group;
02376 }
02377
02378 inline void GeometryGroupObj::destroy()
02379 {
02380     Context context = getContext();
02381     checkError( rtGeometryGroupDestroy( m_geometrygroup ), context );
02382     m_geometrygroup = 0;
02383 }
02384
02385 inline void GeometryGroupObj::validate()
02386 {
02387     checkError( rtGeometryGroupValidate( m_geometrygroup ) );
02388 }
02389
02390 inline Context GeometryGroupObj::getContext() const
02391 {
02392     RTcontext c;
02393     checkErrorNoGetContext( rtGeometryGroupGetContext( m_geometrygroup, &c ) );
02394     return Context::take( c );
02395 }
02396
02397 inline void GeometryGroupObj::setAcceleration( Acceleration acceleration )
02398 {
02399     checkError( rtGeometryGroupSetAcceleration( m_geometrygroup, acceleration->
02400 get() ) );
02401 }
02402
02403 inline Acceleration GeometryGroupObj::getAcceleration() const
02404 {
02405     RTacceleration result;
02406     checkError( rtGeometryGroupGetAcceleration( m_geometrygroup, &result ) );
02407     return Acceleration::take( result );
02408 }
02409
02410 inline void GeometryGroupObj::setChildCount( unsigned int count )
02411 {
02412     checkError( rtGeometryGroupSetChildCount( m_geometrygroup, count ) );
02413 }
02414
02415 inline unsigned int GeometryGroupObj::getChildCount() const
02416 {
02417     unsigned int result;
02418     checkError( rtGeometryGroupGetChildCount( m_geometrygroup, &result ) );
02419     return result;
02420 }
02421
02422 inline void GeometryGroupObj::setChild( unsigned int index, GeometryInstance
02423 child )
02424 {
02425     checkError( rtGeometryGroupSetChild( m_geometrygroup, index, child->get() ) );
02426 }
02427
02428 inline GeometryInstance GeometryGroupObj::getChild( unsigned int index ) const
02429 {
02430     RTgeometryinstance result;
02431     checkError( rtGeometryGroupGetChild( m_geometrygroup, index, &result ) );
02432     return GeometryInstance::take( result );
02433 }

```

```

02432
02433 inline RTgeometrygroup GeometryGroupObj::get()
02434 {
02435     return m_geometrygroup;
02436 }
02437
02438 inline void TransformObj::destroy()
02439 {
02440     Context context = getContext();
02441     checkError( rtTransformDestroy( m_transform ), context );
02442     m_transform = 0;
02443 }
02444
02445 inline void TransformObj::validate()
02446 {
02447     checkError( rtTransformValidate( m_transform ) );
02448 }
02449
02450 inline Context TransformObj::getContext() const
02451 {
02452     RTcontext c;
02453     checkErrorNoGetContext( rtTransformGetContext( m_transform, &c ) );
02454     return Context::take(c);
02455 }
02456
02457 template< typename T >
02458 inline void TransformObj::setChild(T child)
02459 {
02460     checkError( rtTransformSetChild( m_transform, child->get() ) );
02461 }
02462
02463 template< typename T >
02464 inline T TransformObj::getChild() const
02465 {
02466     RObject result;
02467     checkError( rtTransformGetChild( m_transform, &result ) );
02468     return T::take( result );
02469 }
02470
02471 inline void TransformObj::setMatrix(bool transpose, const float* matrix,
const float* inverse_matrix)
02472 {
02473     rtTransformSetMatrix( m_transform, transpose, matrix, inverse_matrix );
02474 }
02475
02476 inline void TransformObj::getMatrix(bool transpose, float* matrix, float*
inverse_matrix) const
02477 {
02478     rtTransformGetMatrix( m_transform, transpose, matrix, inverse_matrix );
02479 }
02480
02481 inline RTtransform TransformObj::get()
02482 {
02483     return m_transform;
02484 }
02485
02486 inline void AccelerationObj::destroy()
02487 {
02488     Context context = getContext();
02489     checkError( rtAccelerationDestroy(m_acceleration), context );
02490     m_acceleration = 0;
02491 }
02492
02493 inline void AccelerationObj::validate()
02494 {
02495     checkError( rtAccelerationValidate(m_acceleration) );
02496 }
02497
02498 inline Context AccelerationObj::getContext() const
02499 {
02500     RTcontext c;
02501     checkErrorNoGetContext( rtAccelerationGetContext(m_acceleration, &c ) );
02502     return Context::take( c );
02503 }

```

```
02504
02505 inline void AccelerationObj::markDirty()
02506 {
02507     checkError( rtAccelerationMarkDirty(m_acceleration) );
02508 }
02509
02510 inline bool AccelerationObj::isDirty() const
02511 {
02512     int dirty;
02513     checkError( rtAccelerationIsDirty(m_acceleration,&dirty) );
02514     return dirty != 0;
02515 }
02516
02517 inline void AccelerationObj::setProperty( const std::string& name, const
std::string& value )
02518 {
02519     checkError( rtAccelerationSetProperty(m_acceleration, name.c_str(), value.
c_str() ) );
02520 }
02521
02522 inline std::string AccelerationObj::getProperty( const std::string& name )
const
02523 {
02524     const char* s;
02525     checkError( rtAccelerationGetProperty(m_acceleration, name.c_str(), &s ) );
02526     return std::string( s );
02527 }
02528
02529 inline void AccelerationObj::setBuilder(const std::string& builder)
02530 {
02531     checkError( rtAccelerationSetBuilder(m_acceleration, builder.c_str() ) );
02532 }
02533
02534 inline std::string AccelerationObj::getBuilder() const
02535 {
02536     const char* s;
02537     checkError( rtAccelerationGetBuilder(m_acceleration, &s ) );
02538     return std::string( s );
02539 }
02540
02541 inline void AccelerationObj::setTraverser(const std::string& traverser)
02542 {
02543     checkError( rtAccelerationSetTraverser(m_acceleration, traverser.c_str() )
);
02544 }
02545
02546 inline std::string AccelerationObj::getTraverser() const
02547 {
02548     const char* s;
02549     checkError( rtAccelerationGetTraverser(m_acceleration, &s ) );
02550     return std::string( s );
02551 }
02552
02553 inline RTsize AccelerationObj::getDataSize() const
02554 {
02555     RTsize sz;
02556     checkError( rtAccelerationGetDataSize(m_acceleration, &sz) );
02557     return sz;
02558 }
02559
02560 inline void AccelerationObj::getData( void* data ) const
02561 {
02562     checkError( rtAccelerationGetData(m_acceleration,data) );
02563 }
02564
02565 inline void AccelerationObj::setData( const void* data, RTsize size )
02566 {
02567     checkError( rtAccelerationSetData(m_acceleration,data,size) );
02568 }
02569
02570 inline RTacceleration AccelerationObj::get()
02571 {
02572     return m_acceleration;
02573 }
```

```

02574
02575 inline void GeometryInstanceObj::destroy()
02576 {
02577     Context context = getContext();
02578     checkError( rtGeometryInstanceDestroy( m_geometryinstance ), context );
02579     m_geometryinstance = 0;
02580 }
02581
02582 inline void GeometryInstanceObj::validate()
02583 {
02584     checkError( rtGeometryInstanceValidate( m_geometryinstance ) );
02585 }
02586
02587 inline Context GeometryInstanceObj::getContext() const
02588 {
02589     RTcontext c;
02590     checkErrorNoGetContext( rtGeometryInstanceGetContext( m_geometryinstance, &
02591 c ) );
02592     return Context::take( c );
02593 }
02594
02595 inline void GeometryInstanceObj::setGeometry(Geometry geometry)
02596 {
02597     checkError( rtGeometryInstanceSetGeometry( m_geometryinstance, geometry->get
02598 ( ) ) );
02599 }
02600
02601 inline Geometry GeometryInstanceObj::getGeometry() const
02602 {
02603     RTgeometry result;
02604     checkError( rtGeometryInstanceGetGeometry( m_geometryinstance, &result ) );
02605     return Geometry::take( result );
02606 }
02607
02608 inline void GeometryInstanceObj::setMaterialCount(unsigned int count)
02609 {
02610     checkError( rtGeometryInstanceSetMaterialCount( m_geometryinstance, count )
02611 );
02612 }
02613
02614 inline unsigned int GeometryInstanceObj::getMaterialCount() const
02615 {
02616     unsigned int result;
02617     checkError( rtGeometryInstanceGetMaterialCount( m_geometryinstance, &result
02618 ) );
02619     return result;
02620 }
02621
02622 inline void GeometryInstanceObj::setMaterial(unsigned int idx, Material
02623 material)
02624 {
02625     checkError( rtGeometryInstanceSetMaterial( m_geometryinstance, idx,
02626 material->get() ) );
02627 }
02628
02629 inline Material GeometryInstanceObj::getMaterial(unsigned int idx) const
02630 {
02631     RTmaterial result;
02632     checkError( rtGeometryInstanceGetMaterial( m_geometryinstance, idx, &result
02633 ) );
02634     return Material::take( result );
02635 }
02636
02637 // Adds the material and returns the index to the added material.
02638 inline unsigned int GeometryInstanceObj::addMaterial(Material material)
02639 {
02640     unsigned int old_count = getMaterialCount();
02641     setMaterialCount( old_count+1 );
02642     setMaterial( old_count, material );
02643     return old_count;
02644 }
02645
02646 inline Variable GeometryInstanceObj::declareVariable(const std::string& name)
02647 {

```

```

02641     RTvariable v;
02642     checkError( rtGeometryInstanceDeclareVariable( m_geometryinstance, name.
c_str(), &v ) );
02643     return Variable::take( v );
02644 }
02645
02646 inline Variable GeometryInstanceObj::queryVariable(const std::string& name)
const
02647 {
02648     RTvariable v;
02649     checkError( rtGeometryInstanceQueryVariable( m_geometryinstance, name.c_str
(), &v ) );
02650     return Variable::take( v );
02651 }
02652
02653 inline void GeometryInstanceObj::removeVariable(Variable v)
02654 {
02655     checkError( rtGeometryInstanceRemoveVariable( m_geometryinstance, v->get()
) );
02656 }
02657
02658 inline unsigned int GeometryInstanceObj::getVariableCount() const
02659 {
02660     unsigned int result;
02661     checkError( rtGeometryInstanceGetVariableCount( m_geometryinstance, &result
) );
02662     return result;
02663 }
02664
02665 inline Variable GeometryInstanceObj::getVariable(unsigned int index) const
02666 {
02667     RTvariable v;
02668     checkError( rtGeometryInstanceGetVariable( m_geometryinstance, index, &v )
);
02669     return Variable::take( v );
02670 }
02671
02672 inline RTgeometryinstance GeometryInstanceObj::get()
02673 {
02674     return m_geometryinstance;
02675 }
02676
02677 inline void GeometryObj::destroy()
02678 {
02679     Context context = getContext();
02680     checkError( rtGeometryDestroy( m_geometry ), context );
02681     m_geometry = 0;
02682 }
02683
02684 inline void GeometryObj::validate()
02685 {
02686     checkError( rtGeometryValidate( m_geometry ) );
02687 }
02688
02689 inline Context GeometryObj::getContext() const
02690 {
02691     RTcontext c;
02692     checkErrorNoGetContext( rtGeometryGetContext( m_geometry, &c ) );
02693     return Context::take( c );
02694 }
02695
02696 inline void GeometryObj::setPrimitiveCount(unsigned int num_primitives)
02697 {
02698     checkError( rtGeometrySetPrimitiveCount( m_geometry, num_primitives ) );
02699 }
02700
02701 inline unsigned int GeometryObj::getPrimitiveCount() const
02702 {
02703     unsigned int result;
02704     checkError( rtGeometryGetPrimitiveCount( m_geometry, &result ) );
02705     return result;
02706 }
02707
02708 inline void GeometryObj::setBoundingBoxProgram(Program program)

```



```

02709 {
02710     checkError( rtGeometrySetBoundingBoxProgram( m_geometry, program->get() ) )
;
02711 }
02712
02713 inline Program GeometryObj::getBoundingBoxProgram() const
02714 {
02715     RTprogram result;
02716     checkError( rtGeometryGetBoundingBoxProgram( m_geometry, &result ) );
02717     return Program::take( result );
02718 }
02719
02720 inline void GeometryObj::setIntersectionProgram(Program program)
02721 {
02722     checkError( rtGeometrySetIntersectionProgram( m_geometry, program->get() )
);
02723 }
02724
02725 inline Program GeometryObj::getIntersectionProgram() const
02726 {
02727     RTprogram result;
02728     checkError( rtGeometryGetIntersectionProgram( m_geometry, &result ) );
02729     return Program::take( result );
02730 }
02731
02732 inline Variable GeometryObj::declareVariable(const std::string& name)
02733 {
02734     RTvariable v;
02735     checkError( rtGeometryDeclareVariable( m_geometry, name.c_str(), &v ) );
02736     return Variable::take( v );
02737 }
02738
02739 inline Variable GeometryObj::queryVariable(const std::string& name) const
02740 {
02741     RTvariable v;
02742     checkError( rtGeometryQueryVariable( m_geometry, name.c_str(), &v ) );
02743     return Variable::take( v );
02744 }
02745
02746 inline void GeometryObj::removeVariable(Variable v)
02747 {
02748     checkError( rtGeometryRemoveVariable( m_geometry, v->get() ) );
02749 }
02750
02751 inline unsigned int GeometryObj::getVariableCount() const
02752 {
02753     unsigned int result;
02754     checkError( rtGeometryGetVariableCount( m_geometry, &result ) );
02755     return result;
02756 }
02757
02758 inline Variable GeometryObj::getVariable(unsigned int index) const
02759 {
02760     RTvariable v;
02761     checkError( rtGeometryGetVariable( m_geometry, index, &v ) );
02762     return Variable::take( v );
02763 }
02764
02765 inline void GeometryObj::markDirty()
02766 {
02767     checkError( rtGeometryMarkDirty(m_geometry) );
02768 }
02769
02770 inline bool GeometryObj::isDirty() const
02771 {
02772     int dirty;
02773     checkError( rtGeometryIsDirty(m_geometry,&dirty) );
02774     return dirty != 0;
02775 }
02776
02777 inline RTgeometry GeometryObj::get()
02778 {
02779     return m_geometry;
02780 }

```

```

02781
02782 inline void MaterialObj::destroy()
02783 {
02784     Context context = getContext();
02785     checkError( rtMaterialDestroy( m_material ), context );
02786     m_material = 0;
02787 }
02788
02789 inline void MaterialObj::validate()
02790 {
02791     checkError( rtMaterialValidate( m_material ) );
02792 }
02793
02794 inline Context MaterialObj::getContext() const
02795 {
02796     RTcontext c;
02797     checkErrorNoGetContext( rtMaterialGetContext( m_material, &c ) );
02798     return Context::take( c );
02799 }
02800
02801 inline void MaterialObj::setClosestHitProgram(unsigned int ray_type_index,
Program program)
02802 {
02803     checkError( rtMaterialSetClosestHitProgram( m_material, ray_type_index,
program->get() ) );
02804 }
02805
02806 inline Program MaterialObj::getClosestHitProgram(unsigned int ray_type_index)
const
02807 {
02808     RTprogram result;
02809     checkError( rtMaterialGetClosestHitProgram( m_material, ray_type_index, &
result ) );
02810     return Program::take( result );
02811 }
02812
02813 inline void MaterialObj::setAnyHitProgram(unsigned int ray_type_index, Program
program)
02814 {
02815     checkError( rtMaterialSetAnyHitProgram( m_material, ray_type_index, program
->get() ) );
02816 }
02817
02818 inline Program MaterialObj::getAnyHitProgram(unsigned int ray_type_index)
const
02819 {
02820     RTprogram result;
02821     checkError( rtMaterialGetAnyHitProgram( m_material, ray_type_index, &result
) );
02822     return Program::take( result );
02823 }
02824
02825 inline Variable MaterialObj::declareVariable(const std::string& name)
02826 {
02827     RTvariable v;
02828     checkError( rtMaterialDeclareVariable( m_material, name.c_str(), &v ) );
02829     return Variable::take( v );
02830 }
02831
02832 inline Variable MaterialObj::queryVariable(const std::string& name) const
02833 {
02834     RTvariable v;
02835     checkError( rtMaterialQueryVariable( m_material, name.c_str(), &v ) );
02836     return Variable::take( v );
02837 }
02838
02839 inline void MaterialObj::removeVariable(Variable v)
02840 {
02841     checkError( rtMaterialRemoveVariable( m_material, v->get() ) );
02842 }
02843
02844 inline unsigned int MaterialObj::getVariableCount() const
02845 {
02846     unsigned int result;

```

```

02847     checkError( rtMaterialGetVariableCount( m_material, &result ) );
02848     return result;
02849 }
02850
02851 inline Variable MaterialObj::getVariable(unsigned int index) const
02852 {
02853     RTvariable v;
02854     checkError( rtMaterialGetVariable( m_material, index, &v ) );
02855     return Variable::take( v );
02856 }
02857
02858 inline RTmaterial MaterialObj::get()
02859 {
02860     return m_material;
02861 }
02862
02863 inline void TextureSamplerObj::destroy()
02864 {
02865     Context context = getContext();
02866     checkError( rtTextureSamplerDestroy( m_texturesampler ), context );
02867     m_texturesampler = 0;
02868 }
02869
02870 inline void TextureSamplerObj::validate()
02871 {
02872     checkError( rtTextureSamplerValidate( m_texturesampler ) );
02873 }
02874
02875 inline Context TextureSamplerObj::getContext() const
02876 {
02877     RTcontext c;
02878     checkErrorNoGetContext( rtTextureSamplerGetContext( m_texturesampler, &c )
02879 );
02880     return Context::take( c );
02881 }
02882 inline void TextureSamplerObj::setMipLevelCount(unsigned int num_mip_levels)
02883 {
02884     checkError( rtTextureSamplerSetMipLevelCount( m_texturesampler,
02885 num_mip_levels ) );
02886 }
02887 inline unsigned int TextureSamplerObj::getMipLevelCount() const
02888 {
02889     unsigned int result;
02890     checkError( rtTextureSamplerGetMipLevelCount( m_texturesampler, &result ) )
02891 ;
02892     return result;
02893 }
02894 inline void TextureSamplerObj::setArraySize(unsigned int
02895 num_textures_in_array)
02896 {
02897     checkError( rtTextureSamplerSetArraySize( m_texturesampler,
02898 num_textures_in_array ) );
02899 }
02898
02899 inline unsigned int TextureSamplerObj::getArraySize() const
02900 {
02901     unsigned int result;
02902     checkError( rtTextureSamplerGetArraySize( m_texturesampler, &result ) );
02903     return result;
02904 }
02905
02906 inline void TextureSamplerObj::setWrapMode(unsigned int dim, RTwrapmode
02907 wrapmode)
02908 {
02909     checkError( rtTextureSamplerSetWrapMode( m_texturesampler, dim, wrapmode )
02910 );
02911 }
02910
02911 inline RTwrapmode TextureSamplerObj::getWrapMode(unsigned int dim) const
02912 {
02913     RTwrapmode wrapmode;

```

```

02914     checkError( rtTextureSamplerGetWrapMode( m_texturesampler, dim, &wrapmode )
02915 );
02916     return wrapmode;
02917 }
02918 inline void TextureSamplerObj::setFilteringModes(RTfiltermode minification,
02919 RTfiltermode magnification, RTfiltermode mipmapping)
02920 {
02921     checkError( rtTextureSamplerSetFilteringModes( m_texturesampler,
02922 minification, magnification, mipmapping ) );
02923 }
02924 inline void TextureSamplerObj::getFilteringModes(RTfiltermode& minification,
02925 RTfiltermode& magnification, RTfiltermode& mipmapping) const
02926 {
02927     checkError( rtTextureSamplerGetFilteringModes( m_texturesampler, &
02928 minification, &magnification, &mipmapping ) );
02929 }
02930 inline void TextureSamplerObj::setMaxAnisotropy(float value)
02931 {
02932     checkError( rtTextureSamplerSetMaxAnisotropy(m_texturesampler, value ) );
02933 }
02934 inline float TextureSamplerObj::getMaxAnisotropy() const
02935 {
02936     float result;
02937     checkError( rtTextureSamplerGetMaxAnisotropy( m_texturesampler, &result ) );
02938     return result;
02939 }
02940 inline int TextureSamplerObj::getId() const
02941 {
02942     int result;
02943     checkError( rtTextureSamplerGetId( m_texturesampler, &result ) );
02944     return result;
02945 }
02946 inline void TextureSamplerObj::setReadMode(RTtexturereadmode readmode)
02947 {
02948     checkError( rtTextureSamplerSetReadMode( m_texturesampler, readmode ) );
02949 }
02950 inline RTtexturereadmode TextureSamplerObj::getReadMode() const
02951 {
02952     RTtexturereadmode result;
02953     checkError( rtTextureSamplerGetReadMode( m_texturesampler, &result ) );
02954     return result;
02955 }
02956 inline void TextureSamplerObj::setIndexingMode(RTtextureindexmode indexmode)
02957 {
02958     checkError( rtTextureSamplerSetIndexingMode( m_texturesampler, indexmode )
02959 );
02960 }
02961 inline RTtextureindexmode TextureSamplerObj::getIndexingMode() const
02962 {
02963     RTtextureindexmode result;
02964     checkError( rtTextureSamplerGetIndexingMode( m_texturesampler, &result ) );
02965     return result;
02966 }
02967 inline void TextureSamplerObj::setBuffer(unsigned int texture_array_idx,
02968 unsigned int mip_level, Buffer buffer)
02969 {
02970     checkError( rtTextureSamplerSetBuffer( m_texturesampler, texture_array_idx,
02971 mip_level, buffer->get() ) );
02972 }
02973 inline Buffer TextureSamplerObj::getBuffer(unsigned int texture_array_idx,
02974 unsigned int mip_level) const
02975 {
02976     RTbuffer result;

```

```

02979     checkError( rtTextureSamplerGetBuffer(m_texturesampler, texture_array_idx,
mip_level, &result ) );
02980     return Buffer::take(result);
02981 }
02982
02983 inline RTtexturesampler TextureSamplerObj::get()
02984 {
02985     return m_texturesampler;
02986 }
02987
02988 inline void TextureSamplerObj::registerGLTexture()
02989 {
02990     checkError( rtTextureSamplerGLRegister( m_texturesampler ) );
02991 }
02992
02993 inline void TextureSamplerObj::unregisterGLTexture()
02994 {
02995     checkError( rtTextureSamplerGLUnregister( m_texturesampler ) );
02996 }
02997
02998 #ifdef _WIN32
02999
03000 inline void TextureSamplerObj::registerD3D9Texture()
03001 {
03002     checkError( rtTextureSamplerD3D9Register( m_texturesampler ) );
03003 }
03004
03005 inline void TextureSamplerObj::registerD3D10Texture()
03006 {
03007     checkError( rtTextureSamplerD3D10Register( m_texturesampler ) );
03008 }
03009
03010 inline void TextureSamplerObj::registerD3D11Texture()
03011 {
03012     checkError( rtTextureSamplerD3D11Register( m_texturesampler ) );
03013 }
03014
03015 inline void TextureSamplerObj::unregisterD3D9Texture()
03016 {
03017     checkError( rtTextureSamplerD3D9Unregister( m_texturesampler ) );
03018 }
03019
03020 inline void TextureSamplerObj::unregisterD3D10Texture()
03021 {
03022     checkError( rtTextureSamplerD3D10Unregister( m_texturesampler ) );
03023 }
03024
03025 inline void TextureSamplerObj::unregisterD3D11Texture()
03026 {
03027     checkError( rtTextureSamplerD3D11Unregister( m_texturesampler ) );
03028 }
03029
03030 #endif
03031
03032 inline void BufferObj::destroy()
03033 {
03034     Context context = getContext();
03035     checkError( rtBufferDestroy( m_buffer ), context );
03036     m_buffer = 0;
03037 }
03038
03039 inline void BufferObj::validate()
03040 {
03041     checkError( rtBufferValidate( m_buffer ) );
03042 }
03043
03044 inline Context BufferObj::getContext() const
03045 {
03046     RTcontext c;
03047     checkErrorNoGetContext( rtBufferGetContext( m_buffer, &c ) );
03048     return Context::take( c );
03049 }
03050
03051 inline void BufferObj::setFormat( RTformat format)

```

```
03052 {
03053     checkError( rtBufferSetFormat( m_buffer, format ) );
03054 }
03055
03056 inline RTformat BufferObj::getFormat() const
03057 {
03058     RTformat result;
03059     checkError( rtBufferGetFormat( m_buffer, &result ) );
03060     return result;
03061 }
03062
03063 inline void BufferObj::setElementSize(RTsize size_of_element)
03064 {
03065     checkError( rtBufferSetElementSize ( m_buffer, size_of_element ) );
03066 }
03067
03068 inline RTsize BufferObj::getElementSize() const
03069 {
03070     RTsize result;
03071     checkError( rtBufferGetElementSize ( m_buffer, &result ) );
03072     return result;
03073 }
03074
03075 inline void BufferObj::getDevicePointer(unsigned int optix_device_number,
CUdeviceptr *device_pointer)
03076 {
03077     checkError( rtBufferGetDevicePointer( m_buffer, optix_device_number, (void*
*)device_pointer ) );
03078 }
03079
03080 inline void BufferObj::setDevicePointer(unsigned int optix_device_number,
CUdeviceptr device_pointer)
03081 {
03082     checkError( rtBufferSetDevicePointer( m_buffer, optix_device_number,
device_pointer ) );
03083 }
03084
03085 inline void BufferObj::markDirty()
03086 {
03087     checkError( rtBufferMarkDirty( m_buffer ) );
03088 }
03089
03090 inline void BufferObj::setSize(RTsize width)
03091 {
03092     checkError( rtBufferSetSize1D( m_buffer, width ) );
03093 }
03094
03095 inline void BufferObj::getSize(RTsize& width) const
03096 {
03097     checkError( rtBufferGetSize1D( m_buffer, &width ) );
03098 }
03099
03100 inline void BufferObj::setSize(RTsize width, RTsize height)
03101 {
03102     checkError( rtBufferSetSize2D( m_buffer, width, height ) );
03103 }
03104
03105 inline void BufferObj::getSize(RTsize& width, RTsize& height) const
03106 {
03107     checkError( rtBufferGetSize2D( m_buffer, &width, &height ) );
03108 }
03109
03110 inline void BufferObj::setSize(RTsize width, RTsize height, RTsize depth)
03111 {
03112     checkError( rtBufferSetSize3D( m_buffer, width, height, depth ) );
03113 }
03114
03115 inline void BufferObj::getSize(RTsize& width, RTsize& height, RTsize& depth)
const
03116 {
03117     checkError( rtBufferGetSize3D( m_buffer, &width, &height, &depth ) );
03118 }
03119
03120 inline void BufferObj::setSize(unsigned int dimensionality, const RTsize*
```

```
    dims)
03121 {
03122     checkError( rtBufferSetSizev( m_buffer, dimensionality, dims ) );
03123 }
03124
03125 inline void BufferObj::getSize(unsigned int dimensionality, RTsize* dims)
    const
03126 {
03127     checkError( rtBufferGetSizev( m_buffer, dimensionality, dims ) );
03128 }
03129
03130 inline unsigned int BufferObj::getDimensionality() const
03131 {
03132     unsigned int result;
03133     checkError( rtBufferGetDimensionality( m_buffer, &result ) );
03134     return result;
03135 }
03136
03137 inline unsigned int BufferObj::getGLBOId() const
03138 {
03139     unsigned int result;
03140     checkError( rtBufferGetGLBOId( m_buffer, &result ) );
03141     return result;
03142 }
03143
03144 inline void BufferObj::registerGLBuffer()
03145 {
03146     checkError( rtBufferGLRegister( m_buffer ) );
03147 }
03148
03149 inline void BufferObj::unregisterGLBuffer()
03150 {
03151     checkError( rtBufferGLUnregister( m_buffer ) );
03152 }
03153
03154 #ifndef _WIN32
03155
03156 inline void BufferObj::registerD3D9Buffer()
03157 {
03158     checkError( rtBufferD3D9Register( m_buffer ) );
03159 }
03160
03161 inline void BufferObj::registerD3D10Buffer()
03162 {
03163     checkError( rtBufferD3D10Register( m_buffer ) );
03164 }
03165
03166 inline void BufferObj::registerD3D11Buffer()
03167 {
03168     checkError( rtBufferD3D11Register( m_buffer ) );
03169 }
03170
03171 inline void BufferObj::unregisterD3D9Buffer()
03172 {
03173     checkError( rtBufferD3D9Unregister( m_buffer ) );
03174 }
03175
03176 inline void BufferObj::unregisterD3D10Buffer()
03177 {
03178     checkError( rtBufferD3D10Unregister( m_buffer ) );
03179 }
03180
03181 inline void BufferObj::unregisterD3D11Buffer()
03182 {
03183     checkError( rtBufferD3D11Unregister( m_buffer ) );
03184 }
03185
03186 inline IDirect3DResource9* BufferObj::getD3D9Resource()
03187 {
03188     IDirect3DResource9* result = NULL;
03189     checkError( rtBufferGetD3D9Resource( m_buffer, &result ) );
03190     return result;
03191 }
03192
```

```

03193 inline ID3D10Resource* BufferObj::getD3D10Resource()
03194 {
03195     ID3D10Resource* result = NULL;
03196     checkError( rtBufferGetD3D10Resource( m_buffer, &result ) );
03197     return result;
03198 }
03199
03200 inline ID3D11Resource* BufferObj::getD3D11Resource()
03201 {
03202     ID3D11Resource* result = NULL;
03203     checkError( rtBufferGetD3D11Resource( m_buffer, &result ) );
03204     return result;
03205 }
03206
03207 #endif
03208
03209 inline void* BufferObj::map()
03210 {
03211     void* result;
03212     checkError( rtBufferMap( m_buffer, &result ) );
03213     return result;
03214 }
03215
03216 inline void BufferObj::unmap()
03217 {
03218     checkError( rtBufferUnmap( m_buffer ) );
03219 }
03220
03221
03222 inline RTbuffer BufferObj::get()
03223 {
03224     return m_buffer;
03225 }
03226
03227 inline Context VariableObj::getContext() const
03228 {
03229     RTcontext c;
03230     checkErrorNoGetContext( rtVariableGetContext( m_variable, &c ) );
03231     return Context::take( c );
03232 }
03233
03234 inline void VariableObj::setUint(unsigned int u1)
03235 {
03236     checkError( rtVariableSet1ui( m_variable, u1 ) );
03237 }
03238
03239 inline void VariableObj::setUint(unsigned int u1, unsigned int u2)
03240 {
03241     checkError( rtVariableSet2ui( m_variable, u1, u2 ) );
03242 }
03243
03244 inline void VariableObj::setUint(unsigned int u1, unsigned int u2, unsigned
int u3)
03245 {
03246     checkError( rtVariableSet3ui( m_variable, u1, u2, u3 ) );
03247 }
03248
03249 inline void VariableObj::setUint(unsigned int u1, unsigned int u2, unsigned
int u3, unsigned int u4)
03250 {
03251     checkError( rtVariableSet4ui( m_variable, u1, u2, u3, u4 ) );
03252 }
03253
03254 inline void VariableObj::setUint(optix::uint2 u)
03255 {
03256     checkError( rtVariableSet2uiv( m_variable, &u.x ) );
03257 }
03258
03259 inline void VariableObj::setUint(optix::uint3 u)
03260 {
03261     checkError( rtVariableSet3uiv( m_variable, &u.x ) );
03262 }
03263
03264 inline void VariableObj::setUint(optix::uint4 u)

```



```
03265 {
03266     checkError( rtVariableSet4uiv( m_variable, &u.x ) );
03267 }
03268
03269 inline void VariableObj::set1uiv(const unsigned int* u)
03270 {
03271     checkError( rtVariableSet1uiv( m_variable, u ) );
03272 }
03273
03274 inline void VariableObj::set2uiv(const unsigned int* u)
03275 {
03276     checkError( rtVariableSet2uiv( m_variable, u ) );
03277 }
03278
03279 inline void VariableObj::set3uiv(const unsigned int* u)
03280 {
03281     checkError( rtVariableSet3uiv( m_variable, u ) );
03282 }
03283
03284 inline void VariableObj::set4uiv(const unsigned int* u)
03285 {
03286     checkError( rtVariableSet4uiv( m_variable, u ) );
03287 }
03288
03289 inline void VariableObj::setMatrix2x2fv(bool transpose, const float* m)
03290 {
03291     checkError( rtVariableSetMatrix2x2fv( m_variable, (int)transpose, m ) );
03292 }
03293
03294 inline void VariableObj::setMatrix2x3fv(bool transpose, const float* m)
03295 {
03296     checkError( rtVariableSetMatrix2x3fv( m_variable, (int)transpose, m ) );
03297 }
03298
03299 inline void VariableObj::setMatrix2x4fv(bool transpose, const float* m)
03300 {
03301     checkError( rtVariableSetMatrix2x4fv( m_variable, (int)transpose, m ) );
03302 }
03303
03304 inline void VariableObj::setMatrix3x2fv(bool transpose, const float* m)
03305 {
03306     checkError( rtVariableSetMatrix3x2fv( m_variable, (int)transpose, m ) );
03307 }
03308
03309 inline void VariableObj::setMatrix3x3fv(bool transpose, const float* m)
03310 {
03311     checkError( rtVariableSetMatrix3x3fv( m_variable, (int)transpose, m ) );
03312 }
03313
03314 inline void VariableObj::setMatrix3x4fv(bool transpose, const float* m)
03315 {
03316     checkError( rtVariableSetMatrix3x4fv( m_variable, (int)transpose, m ) );
03317 }
03318
03319 inline void VariableObj::setMatrix4x2fv(bool transpose, const float* m)
03320 {
03321     checkError( rtVariableSetMatrix4x2fv( m_variable, (int)transpose, m ) );
03322 }
03323
03324 inline void VariableObj::setMatrix4x3fv(bool transpose, const float* m)
03325 {
03326     checkError( rtVariableSetMatrix4x3fv( m_variable, (int)transpose, m ) );
03327 }
03328
03329 inline void VariableObj::setMatrix4x4fv(bool transpose, const float* m)
03330 {
03331     checkError( rtVariableSetMatrix4x4fv( m_variable, (int)transpose, m ) );
03332 }
03333
03334 inline void VariableObj::setFloat(float f1)
03335 {
03336     checkError( rtVariableSet1f( m_variable, f1 ) );
03337 }
03338
```

```
03339 inline void VariableObj::setFloat(optix::float2 f)
03340 {
03341     checkError( rtVariableSet2fv( m_variable, &f.x ) );
03342 }
03343
03344 inline void VariableObj::setFloat(float f1, float f2)
03345 {
03346     checkError( rtVariableSet2f( m_variable, f1, f2 ) );
03347 }
03348
03349 inline void VariableObj::setFloat(optix::float3 f)
03350 {
03351     checkError( rtVariableSet3fv( m_variable, &f.x ) );
03352 }
03353
03354 inline void VariableObj::setFloat(float f1, float f2, float f3)
03355 {
03356     checkError( rtVariableSet3f( m_variable, f1, f2, f3 ) );
03357 }
03358
03359 inline void VariableObj::setFloat(optix::float4 f)
03360 {
03361     checkError( rtVariableSet4fv( m_variable, &f.x ) );
03362 }
03363
03364 inline void VariableObj::setFloat(float f1, float f2, float f3, float f4)
03365 {
03366     checkError( rtVariableSet4f( m_variable, f1, f2, f3, f4 ) );
03367 }
03368
03369 inline void VariableObj::set1fv(const float* f)
03370 {
03371     checkError( rtVariableSet1fv( m_variable, f ) );
03372 }
03373
03374 inline void VariableObj::set2fv(const float* f)
03375 {
03376     checkError( rtVariableSet2fv( m_variable, f ) );
03377 }
03378
03379 inline void VariableObj::set3fv(const float* f)
03380 {
03381     checkError( rtVariableSet3fv( m_variable, f ) );
03382 }
03383
03384 inline void VariableObj::set4fv(const float* f)
03385 {
03386     checkError( rtVariableSet4fv( m_variable, f ) );
03387 }
03388
03389 inline void VariableObj::setInt(int i1)
03390 {
03391     checkError( rtVariableSet1i( m_variable, i1 ) );
03392 }
03393
03394 inline void VariableObj::setInt(optix::int2 i)
03395 {
03396     checkError( rtVariableSet2iv( m_variable, &i.x ) );
03397 }
03398
03399 inline void VariableObj::setInt(int i1, int i2)
03400 {
03401     checkError( rtVariableSet2i( m_variable, i1, i2 ) );
03402 }
03403
03404 inline void VariableObj::setInt(optix::int3 i)
03405 {
03406     checkError( rtVariableSet3iv( m_variable, &i.x ) );
03407 }
03408
03409 inline void VariableObj::setInt(int i1, int i2, int i3)
03410 {
03411     checkError( rtVariableSet3i( m_variable, i1, i2, i3 ) );
03412 }
03413 }
```

```
03414
03415 inline void VariableObj::setInt(optix::int4 i)
03416 {
03417     checkError( rtVariableSet4iv( m_variable, &i.x ) );
03418 }
03419
03420 inline void VariableObj::setInt(int i1, int i2, int i3, int i4)
03421 {
03422     checkError( rtVariableSet4i( m_variable, i1, i2, i3, i4 ) );
03423 }
03424
03425 inline void VariableObj::setliv( const int* i )
03426 {
03427     checkError( rtVariableSetliv( m_variable, i ) );
03428 }
03429
03430 inline void VariableObj::set2iv( const int* i )
03431 {
03432     checkError( rtVariableSet2iv( m_variable, i ) );
03433 }
03434
03435 inline void VariableObj::set3iv( const int* i )
03436 {
03437     checkError( rtVariableSet3iv( m_variable, i ) );
03438 }
03439
03440 inline void VariableObj::set4iv( const int* i )
03441 {
03442     checkError( rtVariableSet4iv( m_variable, i ) );
03443 }
03444
03445 inline float VariableObj::getFloat() const
03446 {
03447     float f;
03448     checkError( rtVariableGet1f( m_variable, &f ) );
03449     return f;
03450 }
03451
03452 inline unsigned int VariableObj::getUInt() const
03453 {
03454     unsigned int i;
03455     checkError( rtVariableGet1ui( m_variable, &i ) );
03456     return i;
03457 }
03458
03459 inline int VariableObj::getInt() const
03460 {
03461     int i;
03462     checkError( rtVariableGet1i( m_variable, &i ) );
03463     return i;
03464 }
03465
03466 inline void VariableObj::setBuffer(Buffer buffer)
03467 {
03468     checkError( rtVariableSetObject( m_variable, buffer->get() ) );
03469 }
03470
03471 inline void VariableObj::set(Buffer buffer)
03472 {
03473     checkError( rtVariableSetObject( m_variable, buffer->get() ) );
03474 }
03475
03476 inline void VariableObj::setUserData(RTsize size, const void* ptr)
03477 {
03478     checkError( rtVariableSetUserData( m_variable, size, ptr ) );
03479 }
03480
03481 inline void VariableObj::getUserData(RTsize size, void* ptr) const
03482 {
03483     checkError( rtVariableGetUserData( m_variable, size, ptr ) );
03484 }
03485
03486 inline void VariableObj::setTextureSampler(TextureSampler texturesampler)
03487 {
```

```
03488     checkError( rtVariableSetObject( m_variable, texturesampler->get() ) );
03489 }
03490
03491 inline void VariableObj::set(TextureSampler texturesampler)
03492 {
03493     checkError( rtVariableSetObject( m_variable, texturesampler->get() ) );
03494 }
03495
03496 inline void VariableObj::set(GeometryGroup group)
03497 {
03498     checkError( rtVariableSetObject( m_variable, group->get() ) );
03499 }
03500
03501 inline void VariableObj::set(Group group)
03502 {
03503     checkError( rtVariableSetObject( m_variable, group->get() ) );
03504 }
03505
03506 inline void VariableObj::set(Program program)
03507 {
03508     checkError( rtVariableSetObject( m_variable, program->get() ) );
03509 }
03510
03511 inline void VariableObj::set(Selector sel)
03512 {
03513     checkError( rtVariableSetObject( m_variable, sel->get() ) );
03514 }
03515
03516 inline void VariableObj::set(Transform tran)
03517 {
03518     checkError( rtVariableSetObject( m_variable, tran->get() ) );
03519 }
03520
03521 inline Buffer VariableObj::getBuffer() const
03522 {
03523     RObject temp;
03524     checkError( rtVariableGetObject( m_variable, &temp ) );
03525     RTbuffer buffer = reinterpret_cast<RTbuffer>(temp);
03526     return Buffer::take(buffer);
03527 }
03528
03529 inline std::string VariableObj::getName() const
03530 {
03531     const char* name;
03532     checkError( rtVariableGetName( m_variable, &name ) );
03533     return std::string(name);
03534 }
03535
03536 inline std::string VariableObj::getAnnotation() const
03537 {
03538     const char* annotation;
03539     checkError( rtVariableGetAnnotation( m_variable, &annotation ) );
03540     return std::string(annotation);
03541 }
03542
03543 inline RObjecttype VariableObj::getType() const
03544 {
03545     RObjecttype type;
03546     checkError( rtVariableGetType( m_variable, &type ) );
03547     return type;
03548 }
03549
03550 inline RTvariable VariableObj::get()
03551 {
03552     return m_variable;
03553 }
03554
03555 inline RTsize VariableObj::getSize() const
03556 {
03557     RTsize size;
03558     checkError( rtVariableGetSize( m_variable, &size ) );
03559     return size;
03560 }
03561
```

```

03562 inline optix::TextureSampler VariableObj::getTextureSampler() const
03563 {
03564     RObject temp;
03565     checkError( rtVariableGetObject( m_variable, &temp ) );
03566     RTtexturesampler sampler = reinterpret_cast<RTtexturesampler>(temp);
03567     return TextureSampler::take(sampler);
03568 }
03569
03570 inline optix::Program VariableObj::getProgram() const
03571 {
03572     RObject temp;
03573     checkError( rtVariableGetObject( m_variable, &temp ) );
03574     RTprogram program = reinterpret_cast<RTprogram>(temp);
03575     return Program::take(program);
03576 }
03577
03579 }
03580
03581 #endif /* __optixu_optixpp_namespace_h__ */
03582
03584

```

3.3 optixu.h File Reference

```
#include <stddef.h> #include "../optix.h"
```

Defines

- #define [RTU_INLINE](#) inline
- #define [RTU_CHECK_ERROR](#)(func)
- #define [RTU_GROUP_ADD_CHILD](#)(_parent, _child, _index)
- #define [RTU_SELECTOR_ADD_CHILD](#)(_parent, _child, _index)

Functions

- [RTResult RTAPI rtuNameForType](#) (RObjecttype type, char *buffer, RTsize bufferSize)
- [RTResult RTAPI rtuGetSizeForRTformat](#) (RTformat format, size_t *size)
- [RTResult RTAPI rtuCUDACompileString](#) (const char *source, const char **preprocessorArguments, unsigned int numPreprocessorArguments, RTsize *resultSize, RTsize *errorSize)
- [RTResult RTAPI rtuCUDACompileFile](#) (const char *filename, const char **preprocessorArguments, unsigned int numPreprocessorArguments, RTsize *resultSize, RTsize *errorSize)
- [RTResult RTAPI rtuCUDAGetCompileResult](#) (char *result, char *error)
- [RTResult rtuGroupAddChild](#) (RTgroup group, RObject child, unsigned int *index)
- [RTResult rtuSelectorAddChild](#) (RTselector selector, RObject child, unsigned int *index)
- [RTResult rtuGeometryGroupAddChild](#) (RTgeometrygroup geometrygroup, RTgeometryinstance child, unsigned int *index)
- [RTResult rtuTransformSetChild](#) (RTtransform transform, RObject child)
- [RTResult rtuGroupRemoveChild](#) (RTgroup group, RObject child)
- [RTResult rtuSelectorRemoveChild](#) (RTselector selector, RObject child)

- RTresult [rtuGeometryGroupRemoveChild](#) (RTgeometrygroup geometrygroup, RTgeometryinstance child)
- [RTU_INLINE](#) RTresult [rtuGroupRemoveChildByIndex](#) (RTgroup group, unsigned int index)
- [RTU_INLINE](#) RTresult [rtuSelectorRemoveChildByIndex](#) (RTselector selector, unsigned int index)
- [RTU_INLINE](#) RTresult [rtuGeometryGroupRemoveChildByIndex](#) (RTgeometrygroup geometrygroup, unsigned int index)
- [RTU_INLINE](#) RTresult [rtuGroupGetChildIndex](#) (RTgroup group, RObject child, unsigned int *index)
- [RTU_INLINE](#) RTresult [rtuSelectorGetChildIndex](#) (RTselector selector, RObject child, unsigned int *index)
- [RTU_INLINE](#) RTresult [rtuGeometryGroupGetChildIndex](#) (RTgeometrygroup geometrygroup, RTgeometryinstance child, unsigned int *index)
- RTresult RTAPI [rtuCreateClusteredMesh](#) (RTcontext context, unsigned int usePTX32InHost64, RTgeometry *mesh, unsigned int num_verts, const float *verts, unsigned int num_tris, const unsigned *indices, const unsigned *mat_indices)
- RTresult RTAPI [rtuCreateClusteredMeshExt](#) (RTcontext context, unsigned int usePTX32InHost64, RTgeometry *mesh, unsigned int num_verts, const float *verts, unsigned int num_tris, const unsigned *indices, const unsigned *mat_indices, RTbuffer norms, const unsigned *norm_indices, RTbuffer tex_coords, const unsigned *tex_indices)

3.3.1 Define Documentation

3.3.1.1 #define RTU_CHECK_ERROR(func)

Value:

```
do {
    RTresult code = func;
    if( code != RT_SUCCESS )
        return code;
} while (0)
```

Definition at line 154 of file [optixu.h](#).

3.3.1.2 #define RTU_GROUP_ADD_CHILD(_parent, _child, _index)

Value:

```
unsigned int _count;
RTU_CHECK_ERROR( rtGroupGetChildCount( (_parent), &_count ) );
RTU_CHECK_ERROR( rtGroupSetChildCount( (_parent), _count+1 ) );
RTU_CHECK_ERROR( rtGroupSetChild( (_parent), _count, (_child) ) );
if( _index ) *(_index) = _count;
return RT_SUCCESS
```

Definition at line 161 of file [optixu.h](#).

3.3.1.3 #define RTU_INLINE inline

Definition at line 34 of file [optixu.h](#).

3.3.1.4 #define RTU_SELECTOR_ADD_CHILD(*_parent*, *_child*, *_index*)

Value:

```

unsigned int _count;
RTU_CHECK_ERROR( rtSelectorGetChildCount( (_parent), &_count ) ); \
RTU_CHECK_ERROR( rtSelectorSetChildCount( (_parent), _count+1 ) ); \
RTU_CHECK_ERROR( rtSelectorSetChild( (_parent), _count, (_child) ) ); \
if( _index ) *(_index) = _count; \
return RT_SUCCESS

```

Definition at line 169 of file [optixu.h](#).

3.3.2 Function Documentation

3.3.2.1 RTresult RTAPI rtuCreateClusteredMesh (RTcontext *context*, unsigned int *usePTX32InHost64*, RTgeometry * *mesh*, unsigned int *num_verts*, const float * *verts*, unsigned int *num_tris*, const unsigned * *indices*, const unsigned * *mat_indices*)

Create clustered triangle mesh for good memory coherence with paging on. Vertex, index and material buffers are created and attached to the mesh. Cluster's bounding box and intersection programs are attached to the mesh. The intersection program has the following attributes: `rtDeclareVariable(float3, texcoord, attribute texcoord,);` It is always zero `rtDeclareVariable(float3, geometric_normal, attribute geometric_normal,);` `rtDeclareVariable(float3, shading_normal, attribute shading_normal,);` It is equal to `geometric_normal`

Created RTgeometry mesh expects there to be placed into a RTgeometryinstance where the `mat_indices` specified map into materials attached to the RTgeometryinstance

In the event of an error, please query the error string from the RTcontext.

Parameters

<i>context</i>	Context
<i>usePTX32InHost64</i>	Use 32bit PTX bounding box and intersection programs in 64bit application. Takes effect only with 64bit host.
<i>mesh</i>	Output geometry
<i>num_verts</i>	Vertex count
<i>verts</i>	Vertices (<code>num_verts*float*3</code>) [<code>v1_x, v1_y, v1_z, v2.x, ...</code>]
<i>num_tris</i>	Triangle count
<i>indices</i>	Vertex indices (<code>num_tris*unsigned*3</code>) [<code>tri1_index1, tri1_index2, ...</code>]
<i>mat_indices</i>	Indices of materials (<code>num_tris*unsigned</code>) [<code>tri1_mat_index, tri2_mat_index, ...</code>]

3.3.2.2 `RTresult RTAPI rtuCreateClusteredMeshExt (RTcontext context, unsigned int usePTX32InHost64, RTgeometry * mesh, unsigned int num_verts, const float * verts, unsigned int num_tris, const unsigned * indices, const unsigned * mat_indices, RTbuffer norms, const unsigned * norm_indices, RTbuffer tex_coords, const unsigned * tex_indices)`

Create clustered triangle mesh for good memory coherence with paging on. Buffers for vertices, indices, normals, indices of normals, texture coordinates, indices of texture coordinates and materials are created and attached to the mesh. Cluster's bounding box and intersection programs are attached to the mesh. The intersection program has the following attributes: `rtDeclareVariable(float3, texcoord, attribute texcoord,);` `rtDeclareVariable(float3, geometric_normal, attribute geometric_normal,);` `rtDeclareVariable(float3, shading_normal, attribute shading_normal,);`

Created `RTgeometry` mesh expects there to be placed into a `RTgeometryinstance` where the `mat_indices` specified map into materials attached to the `RTgeometryinstance`

Vertex, normal and texture coordinate buffers can be shared between many geometry objects

In the event of an error, please query the error string from the `RTcontext`.

Parameters

<i>context</i>	Context
<i>usePTX32-InHost64</i>	Use 32bit PTX bounding box and intersection programs in 64bit application. Takes effect only with 64bit host.
<i>mesh</i>	Output geometry
<i>num_verts</i>	Vertex count
<i>verts</i>	Vertices ($\text{num_verts} * \text{float} * 3$) [<i>v1_x</i> , <i>v1_y</i> , <i>v1_z</i> , <i>v2.x</i> , ...]
<i>num_tris</i>	Triangle count
<i>indices</i>	Vertex indices ($\text{num_tris} * \text{unsigned} * 3$) [<i>tri1_index1</i> , <i>tri1_index2</i> , ...]
<i>mat_indices</i>	Indices of materials ($\text{num_tris} * \text{unsigned}$) [<i>tri1_mat_index</i> , <i>tri2_mat_index</i> , ...]
<i>norms</i>	Normals ($\text{num_norms} * \text{float} * 3$) [<i>v1_x</i> , <i>v1_y</i> , <i>v1_z</i> , <i>v2.x</i> , ...]
<i>norm_indices</i>	Indices of vertex normals ($\text{num_tris} * \text{unsigned} * 3$) [<i>tri1_norm_index1</i> , <i>tri1_norm_index2</i> ...]
<i>tex_coords</i>	Texture uv coords ($\text{num_tex_coords} * \text{float} * 2$) [<i>t1_u</i> , <i>t1_v</i> , <i>t2_u</i> ...]
<i>tex_indices</i>	Indices of texture uv ($\text{num_tris} * \text{unsigned} * 3$) [<i>tri1_tex_index1</i> , <i>tri1_tex_index2</i> ...]

3.3.2.3 `RTresult RTAPI rtuCUDACompileFile (const char * filename, const char ** preprocessorArguments, unsigned int numPreprocessorArguments, RTsize * resultSize, RTsize * errorSize)`

3.3.2.4 `RTresult RTAPI rtuCUDACompileString (const char * source, const char ** preprocessorArguments, unsigned int numPreprocessorArguments, RTsize * resultSize, RTsize * errorSize)`

3.3.2.5 `RTresult RTAPI rtuCUDAGetCompileResult (char * result, char * error)`

3.3.2.6 **RTU_INLINE** RTresult rtuGeometryGroupAddChild (RTgeometrygroup *geometrygroup*, RTgeometryinstance *child*, unsigned int * *index*)

Definition at line 273 of file [optixu.h](#).

3.3.2.7 **RTU_INLINE** RTresult rtuGeometryGroupGetChildIndex (RTgeometrygroup *geometrygroup*, RTgeometryinstance *child*, unsigned int * *index*)

Definition at line 366 of file [optixu.h](#).

3.3.2.8 **RTU_INLINE** RTresult rtuGeometryGroupRemoveChild (RTgeometrygroup *geometrygroup*, RTgeometryinstance *child*)

Definition at line 299 of file [optixu.h](#).

3.3.2.9 **RTU_INLINE** RTresult rtuGeometryGroupRemoveChildByIndex (RTgeometrygroup *geometrygroup*, unsigned int *index*)

Definition at line 329 of file [optixu.h](#).

3.3.2.10 RTresult RTAPI rtuGetSizeForRTformat (RTformat *format*, size_t * *size*)

3.3.2.11 **RTU_INLINE** RTresult rtuGroupAddChild (RTgroup *group*, RTOBJECT *child*, unsigned int * *index*)

Definition at line 180 of file [optixu.h](#).

3.3.2.12 **RTU_INLINE** RTresult rtuGroupGetChildIndex (RTgroup *group*, RTOBJECT *child*, unsigned int * *index*)

Definition at line 340 of file [optixu.h](#).

3.3.2.13 **RTU_INLINE** RTresult rtuGroupRemoveChild (RTgroup *group*, RTOBJECT *child*)

Definition at line 283 of file [optixu.h](#).

3.3.2.14 **RTU_INLINE** RTresult rtuGroupRemoveChildByIndex (RTgroup *group*, unsigned int *index*)

Definition at line 307 of file [optixu.h](#).

3.3.2.15 RTresult RTAPI rtuNameForType (RTOBJECTTYPE *type*, char * *buffer*, RTsize *bufferSize*)

3.3.2.16 **RTU_INLINE** RTresult rtuSelectorAddChild (RTselector *selector*, RTOBJECT *child*, unsigned int * *index*)

Definition at line 185 of file [optixu.h](#).

3.3.2.17 `RTU_INLINE RTresult rtuSelectorGetChildIndex (RTselector selector, RObject child, unsigned int * index)`

Definition at line 353 of file [optixu.h](#).

3.3.2.18 `RTU_INLINE RTresult rtuSelectorRemoveChild (RTselector selector, RObject child)`

Definition at line 291 of file [optixu.h](#).

3.3.2.19 `RTU_INLINE RTresult rtuSelectorRemoveChildByIndex (RTselector selector, unsigned int index)`

Definition at line 318 of file [optixu.h](#).

3.3.2.20 `RTU_INLINE RTresult rtuTransformSetChild (RTtransform transform, RObject child)`

Definition at line 239 of file [optixu.h](#).

3.4 optixu.h

```

00001
00002 /*
00003  * Copyright (c) 2008 - 2009 NVIDIA Corporation. All rights reserved.
00004  *
00005  * NVIDIA Corporation and its licensors retain all intellectual property and
00006  * proprietary
00007  * rights in and to this software, related documentation and any modifications
00008  * thereto.
00009  * Any use, reproduction, disclosure or distribution of this software and
00010  * related
00011  * documentation without an express license agreement from NVIDIA Corporation
00012  * is strictly
00013  * prohibited.
00014  *
00015  * TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THIS SOFTWARE IS PROVIDED
00016  * *AS IS*
00017  * AND NVIDIA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EITHER EXPRESS OR
00018  * IMPLIED,
00019  * INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND
00020  * FITNESS FOR A
00021  * PARTICULAR PURPOSE. IN NO EVENT SHALL NVIDIA OR ITS SUPPLIERS BE LIABLE FOR
00022  * ANY
00023  * SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER
00024  * (INCLUDING, WITHOUT
00025  * LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION,
00026  * LOSS OF
00027  * BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF
00028  * OR
00029  * INABILITY TO USE THIS SOFTWARE, EVEN IF NVIDIA HAS BEEN ADVISED OF THE
00030  * POSSIBILITY OF
00031  * SUCH DAMAGES
00032  */
00033
00034 #ifndef __optix_optixu_h__
00035 #define __optix_optixu_h__
00036
00037 #include <stddef.h>
00038 #include "../optix.h"
00039
00040 #ifdef __cplusplus
00041 # define RTU_INLINE inline
00042 #else

```

```

00031 #   ifdef _MSC_VER
00032 #       define RTU_INLINE __inline
00033 #   else
00034 #       define RTU_INLINE inline
00035 #   endif
00036 #endif
00037
00038 #ifdef __cplusplus
00039 extern "C" {
00040 #endif
00041
00042 /*
00043  * Get the name string of a given type.
00044  */
00045 RTresult RTAPI rtuNameForType( RObjectType type, char* buffer, RTsize
bufferSize );
00046
00047 /*
00048  * Return the size of a given RTformat.  RT_FORMAT_USER and RT_FORMAT_UNKNOWN
return 0.
00049  * Returns RT_ERROR_INVALID_VALUE if the format isn't recognized, RT_SUCCESS
otherwise.
00050  */
00051 RTresult RTAPI rtuGetSizeForRTformat( RTformat format, size_t* size);
00052
00053 /*
00054  * Compile a cuda source string.
00055  * ARGS:
00056  *
00057  * source                source code string
00058  * preprocessorArguments  list of preprocessor arguments
00059  * numPreprocessorArguments  number of preprocessor arguments
00060  * resultSize            [out] size required to hold compiled result
string
00061  * errorSize            [out] size required to hold error string
00062  */
00063 RTresult RTAPI rtuCUDACompileString( const char* source, const char**
preprocessorArguments, unsigned int numPreprocessorArguments, RTsize* resultSize,
RTsize* errorSize );
00064
00065 /*
00066  * Compile a cuda source file.
00067  * ARGS:
00068  *
00069  * filename                source code file name
00070  * preprocessorArguments  list of preprocessor arguments
00071  * numPreprocessorArguments  number of preprocessor arguments
00072  * resultSize            [out] size required to hold compiled result
string
00073  * errorSize            [out] size required to hold error string
00074  */
00075 RTresult RTAPI rtuCUDACompileFile( const char* filename, const char**
preprocessorArguments, unsigned int numPreprocessorArguments, RTsize* resultSize,
RTsize* errorSize );
00076
00077 /*
00078  * Get the result of the most recent call to one of the above compile
functions.
00079  * The 'result' and 'error' parameters must point to memory large enough to
hold
00080  * the respective strings, as returned by the compile function.
00081  * ARGS:
00082  *
00083  * result                compiled result string
00084  * error                error string
00085  */
00086 RTresult RTAPI rtuCUDAGetCompileResult( char* result, char* error );
00087
00088 #ifdef __cplusplus
00089 } /* extern "C" */
00090 #endif
00091
00092 /*
00093  * Add an entry to the end of the child array.

```

```

00094  * Fills 'index' with the index of the added child, if the pointer is
        non-NULL.
00095  */
00096  #ifndef __cplusplus
00097  RTresult rtuGroupAddChild      ( RTgroup group, RTOBJECT child, unsigned int
        * index );
00098  RTresult rtuSelectorAddChild  ( RTselector selector, RTOBJECT child,
        unsigned int* index );
00099  #else
00100  RTresult rtuGroupAddChild      ( RTgroup group, RTgroup      child,
        unsigned int* index );
00101  RTresult rtuGroupAddChild      ( RTgroup group, RTselector   child,
        unsigned int* index );
00102  RTresult rtuGroupAddChild      ( RTgroup group, RTtransform   child,
        unsigned int* index );
00103  RTresult rtuGroupAddChild      ( RTgroup group, RTgeometrygroup child,
        unsigned int* index );
00104  RTresult rtuSelectorAddChild  ( RTselector selector, RTgroup      child,
        unsigned int* index );
00105  RTresult rtuSelectorAddChild  ( RTselector selector, RTselector   child,
        unsigned int* index );
00106  RTresult rtuSelectorAddChild  ( RTselector selector, RTtransform   child,
        unsigned int* index );
00107  RTresult rtuSelectorAddChild  ( RTselector selector, RTgeometrygroup child,
        unsigned int* index );
00108  #endif
00109  RTresult rtuGeometryGroupAddChild( RTgeometrygroup geometrygroup,
        RTgeometryinstance child, unsigned int* index );
00110
00111  /*
00112  * Wrap rtTransformSetChild in order to provide a type-safe version for C++.
00113  */
00114  #ifndef __cplusplus
00115  RTresult rtuTransformSetChild  ( RTtransform transform, RTOBJECT
        child );
00116  #else
00117  RTresult rtuTransformSetChild  ( RTtransform transform, RTgroup
        child );
00118  RTresult rtuTransformSetChild  ( RTtransform transform, RTselector
        child );
00119  RTresult rtuTransformSetChild  ( RTtransform transform, RTtransform
        child );
00120  RTresult rtuTransformSetChild  ( RTtransform transform, RTgeometrygroup
        child );
00121  #endif
00122
00123  /*
00124  * Find the given child using a linear search in the child array and remove
00125  * it. If it's not the last entry in the child array, the last entry in the
00126  * array will replace the deleted entry, in order to shrink the array size by
        one.
00127  */
00128  RTresult rtuGroupRemoveChild   ( RTgroup group, RTOBJECT child );
00129  RTresult rtuSelectorRemoveChild ( RTselector selector, RTOBJECT child );
00130  RTresult rtuGeometryGroupRemoveChild( RTgeometrygroup geometrygroup,
        RTgeometryinstance child );
00131
00132  /*
00133  * Remove the child at the given index in the child array. If it's not the
        last
00134  * entry in the child array, the last entry in the array will replace the
        deleted
00135  * entry, in order to shrink the array size by one.
00136  */
00137  RTU_INLINE RTresult rtuGroupRemoveChildByIndex      ( RTgroup group,
        unsigned int index );
00138  RTU_INLINE RTresult rtuSelectorRemoveChildByIndex  ( RTselector selector,
        unsigned int index );
00139  RTU_INLINE RTresult rtuGeometryGroupRemoveChildByIndex( RTgeometrygroup
        geometrygroup, unsigned int index );
00140
00141  /*
00142  * Use a linear search to find the child in the child array, and return its
        index.

```

```

00143  * Returns RT_SUCCESS if the child was found, RT_INVALID_VALUE otherwise.
00144  */
00145  RTU_INLINE RTresult rtuGroupGetChildIndex      ( RTgroup group, RTOBJECT
child, unsigned int* index );
00146  RTU_INLINE RTresult rtuSelectorGetChildIndex  ( RTselector selector,
RTOBJECT child, unsigned int* index );
00147  RTU_INLINE RTresult rtuGeometryGroupGetChildIndex( RTgeometrygroup
geometrygroup, RTgeometryinstance child, unsigned int* index );
00148
00149
00150  /*
00151  * The following implements the child management helpers declared above.
00152  */
00153
00154  #define RTU_CHECK_ERROR( func )                \
00155  do {                                          \
00156      RTresult code = func;                    \
00157      if( code != RT_SUCCESS )                 \
00158          return code;                         \
00159  } while(0)
00160
00161  #define RTU_GROUP_ADD_CHILD( _parent, _child, _index ) \
00162  unsigned int _count;                                \
00163  RTU_CHECK_ERROR( rtGroupGetChildCount( (_parent), &_count ) ); \
00164  RTU_CHECK_ERROR( rtGroupSetChildCount( (_parent), _count+1 ) ); \
00165  RTU_CHECK_ERROR( rtGroupSetChild( (_parent), _count, (_child) ) ); \
00166  if( (_index) *(_index) = _count;                \
00167  return RT_SUCCESS
00168
00169  #define RTU_SELECTOR_ADD_CHILD( _parent, _child, _index ) \
00170  unsigned int _count;                                \
00171  RTU_CHECK_ERROR( rtSelectorGetChildCount( (_parent), &_count ) ); \
00172  RTU_CHECK_ERROR( rtSelectorSetChildCount( (_parent), _count+1 ) ); \
00173  RTU_CHECK_ERROR( rtSelectorSetChild( (_parent), _count, (_child) ) ); \
00174  if( (_index) *(_index) = _count;                \
00175  return RT_SUCCESS
00176
00177
00178  #ifndef __cplusplus
00179
00180  RTU_INLINE RTresult rtuGroupAddChild( RTgroup group, RTOBJECT child, unsigned
int* index )
00181  {
00182      RTU_GROUP_ADD_CHILD( group, child, index );
00183  }
00184
00185  RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTOBJECT child,
unsigned int* index )
00186  {
00187      RTU_SELECTOR_ADD_CHILD( selector, child, index );
00188  }
00189
00190  #else /* __cplusplus */
00191
00192  RTU_INLINE RTresult rtuGroupAddChild( RTgroup group, RTgroup child, unsigned
int* index )
00193  {
00194      RTU_GROUP_ADD_CHILD( group, child, index );
00195  }
00196
00197  RTU_INLINE RTresult rtuGroupAddChild( RTgroup group, RTselector child,
unsigned int* index )
00198  {
00199      RTU_GROUP_ADD_CHILD( group, child, index );
00200  }
00201
00202  RTU_INLINE RTresult rtuGroupAddChild( RTgroup group, RTtransform child,
unsigned int* index )
00203  {
00204      RTU_GROUP_ADD_CHILD( group, child, index );
00205  }
00206
00207  RTU_INLINE RTresult rtuGroupAddChild( RTgroup group, RTgeometrygroup child,
unsigned int* index )

```

```
00208 {
00209     RTU_GROUP_ADD_CHILD( group, child, index );
00210 }
00211
00212 RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTgroup child,
    unsigned int* index )
00213 {
00214     RTU_SELECTOR_ADD_CHILD( selector, child, index );
00215 }
00216
00217 RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTselector child
    , unsigned int* index )
00218 {
00219     RTU_SELECTOR_ADD_CHILD( selector, child, index );
00220 }
00221
00222 RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTtransform
    child, unsigned int* index )
00223 {
00224     RTU_SELECTOR_ADD_CHILD( selector, child, index );
00225 }
00226
00227 RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTgeometrygroup
    child, unsigned int* index )
00228 {
00229     RTU_SELECTOR_ADD_CHILD( selector, child, index );
00230 }
00231
00232 #endif /* __cplusplus */
00233
00234 #undef RTU_GROUP_ADD_CHILD
00235 #undef RTU_SELECTOR_ADD_CHILD
00236
00237 #ifndef __cplusplus
00238
00239 RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform, RTobject
    child )
00240 {
00241     RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00242     return RT_SUCCESS;
00243 }
00244
00245 #else /* __cplusplus */
00246
00247 RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform, RTgroup child
    )
00248 {
00249     RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00250     return RT_SUCCESS;
00251 }
00252
00253 RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform, RTselector
    child )
00254 {
00255     RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00256     return RT_SUCCESS;
00257 }
00258
00259 RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform, RTtransform
    child )
00260 {
00261     RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00262     return RT_SUCCESS;
00263 }
00264
00265 RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform,
    RTgeometrygroup child )
00266 {
00267     RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00268     return RT_SUCCESS;
00269 }
00270
00271 #endif /* __cplusplus */
00272
```

```

00273 RTU_INLINE RTresult rtuGeometryGroupAddChild( RTgeometrygroup geometrygroup,
RTgeometryinstance child, unsigned int* index )
00274 {
00275     unsigned int count;
00276     RTU_CHECK_ERROR( rtGeometryGroupGetChildCount( geometrygroup, &count ) );
00277     RTU_CHECK_ERROR( rtGeometryGroupSetChildCount( geometrygroup, count+1 ) );
00278     RTU_CHECK_ERROR( rtGeometryGroupSetChild( geometrygroup, count, child ) );
00279     if( index ) *index = count;
00280     return RT_SUCCESS;
00281 }
00282
00283 RTU_INLINE RTresult rtuGroupRemoveChild( RTgroup group, RTobject child )
00284 {
00285     unsigned int index;
00286     RTU_CHECK_ERROR( rtuGroupGetChildIndex( group, child, &index ) );
00287     RTU_CHECK_ERROR( rtuGroupRemoveChildByIndex( group, index ) );
00288     return RT_SUCCESS;
00289 }
00290
00291 RTU_INLINE RTresult rtuSelectorRemoveChild( RTselector selector, RTobject
child )
00292 {
00293     unsigned int index;
00294     RTU_CHECK_ERROR( rtuSelectorGetChildIndex( selector, child, &index ) );
00295     RTU_CHECK_ERROR( rtuSelectorRemoveChildByIndex( selector, index ) );
00296     return RT_SUCCESS;
00297 }
00298
00299 RTU_INLINE RTresult rtuGeometryGroupRemoveChild( RTgeometrygroup geometrygroup
, RTgeometryinstance child )
00300 {
00301     unsigned int index;
00302     RTU_CHECK_ERROR( rtuGeometryGroupGetChildIndex( geometrygroup, child, &index
) );
00303     RTU_CHECK_ERROR( rtuGeometryGroupRemoveChildByIndex( geometrygroup, index )
);
00304     return RT_SUCCESS;
00305 }
00306
00307 RTU_INLINE RTresult rtuGroupRemoveChildByIndex( RTgroup group, unsigned int
index )
00308 {
00309     unsigned int count;
00310     RTobject temp;
00311     RTU_CHECK_ERROR( rtGroupGetChildCount( group, &count ) );
00312     RTU_CHECK_ERROR( rtGroupGetChild( group, count-1, &temp ) );
00313     RTU_CHECK_ERROR( rtGroupSetChild( group, index, temp ) );
00314     RTU_CHECK_ERROR( rtGroupSetChildCount( group, count-1 ) );
00315     return RT_SUCCESS;
00316 }
00317
00318 RTU_INLINE RTresult rtuSelectorRemoveChildByIndex( RTselector selector,
unsigned int index )
00319 {
00320     unsigned int count;
00321     RTobject temp;
00322     RTU_CHECK_ERROR( rtSelectorGetChildCount( selector, &count ) );
00323     RTU_CHECK_ERROR( rtSelectorGetChild( selector, count-1, &temp ) );
00324     RTU_CHECK_ERROR( rtSelectorSetChild( selector, index, temp ) );
00325     RTU_CHECK_ERROR( rtSelectorSetChildCount( selector, count-1 ) );
00326     return RT_SUCCESS;
00327 }
00328
00329 RTU_INLINE RTresult rtuGeometryGroupRemoveChildByIndex( RTgeometrygroup
geometrygroup, unsigned int index )
00330 {
00331     unsigned int count;
00332     RTgeometryinstance temp;
00333     RTU_CHECK_ERROR( rtGeometryGroupGetChildCount( geometrygroup, &count ) );
00334     RTU_CHECK_ERROR( rtGeometryGroupGetChild( geometrygroup, count-1, &temp ) );
00335     RTU_CHECK_ERROR( rtGeometryGroupSetChild( geometrygroup, index, temp ) );
00336     RTU_CHECK_ERROR( rtGeometryGroupSetChildCount( geometrygroup, count-1 ) );
00337     return RT_SUCCESS;
00338 }

```

```

00339
00340 RTU_INLINE RTresult rtuGroupGetChildIndex(RTgroup group, RObject child,
      unsigned int* index)
00341 {
00342     unsigned int count;
00343     RObject temp;
00344     RTU_CHECK_ERROR( rtGroupGetChildCount( group, &count ) );
00345     for( *index=0; *index<count; (*index)++ ) {
00346         RTU_CHECK_ERROR( rtGroupGetChild( group, *index, &temp ) );
00347         if( child==temp ) return RT_SUCCESS;
00348     }
00349     *index = ~0u;
00350     return RT_ERROR_INVALID_VALUE;
00351 }
00352
00353 RTU_INLINE RTresult rtuSelectorGetChildIndex( RTselector selector, RObject
      child, unsigned int* index )
00354 {
00355     unsigned int count;
00356     RObject temp;
00357     RTU_CHECK_ERROR( rtSelectorGetChildCount( selector, &count ) );
00358     for( *index=0; *index<count; (*index)++ ) {
00359         RTU_CHECK_ERROR( rtSelectorGetChild( selector, *index, &temp ) );
00360         if( child==temp ) return RT_SUCCESS;
00361     }
00362     *index = ~0u;
00363     return RT_ERROR_INVALID_VALUE;
00364 }
00365
00366 RTU_INLINE RTresult rtuGeometryGroupGetChildIndex( RTgeometrygroup
      geometrygroup, RTgeometryinstance child, unsigned int* index )
00367 {
00368     unsigned int count;
00369     RTgeometryinstance temp;
00370     RTU_CHECK_ERROR( rtGeometryGroupGetChildCount( geometrygroup, &count ) );
00371     for( *index=0; *index<count; (*index)++ ) {
00372         RTU_CHECK_ERROR( rtGeometryGroupGetChild( geometrygroup, *index, &temp ) )
      ;
00373         if( child==temp ) return RT_SUCCESS;
00374     }
00375     *index = ~0u;
00376     return RT_ERROR_INVALID_VALUE;
00377 }
00378
00379
00380 #ifdef __cplusplus
00381 extern "C" {
00382 #endif
00383
00407 RTresult RTAPI rtuCreateClusteredMesh( RTcontext          context,
00408                                         unsigned int      usePTX32InHost64,
00409                                         RTgeometry*      mesh,
00410                                         unsigned int      num_verts,
00411                                         const float*      verts,
00412                                         unsigned int      num_tris,
00413                                         const unsigned*   indices,
00414                                         const unsigned*   mat_indices);
00415
00416
00417
00448 RTresult RTAPI rtuCreateClusteredMeshExt( RTcontext          context,
00449                                         unsigned int      usePTX32InHost64,
00450                                         RTgeometry*      mesh,
00451                                         unsigned int      num_verts,
00452                                         const float*      verts,
00453                                         unsigned int      num_tris,
00454                                         const unsigned*   indices,
00455                                         const unsigned*   mat_indices,
00456                                         RTbuffer         norms,
00457                                         const unsigned*   norm_indices,
00458                                         RTbuffer         tex_coords,
00459                                         const unsigned*   tex_indices );
00460
00461 #ifdef __cplusplus

```



```

00462 } /* extern "C" */
00463 #endif
00464
00465
00466 #undef RTU_CHECK_ERROR
00467 #undef RTU_INLINE
00468
00469 #endif /* __optix_optixu_h__ */

```

3.5 optixu_traversal.h File Reference

```
#include "../optix.h"
```

Classes

- struct [RTUtraversalresult](#)
Structure encapsulating the result of a single ray query.

Typedefs

- typedef struct RTUtraversal_api * [RTUtraversal](#)

Enumerations

- enum [RTUquerytype](#) { [RTU_QUERY_TYPE_ANY_HIT](#) = 0, [RTU_QUERY_TYPE_CLOSEST_HIT](#), [RTU_QUERY_TYPE_COUNT](#) }
- enum [RTUrayformat](#) { [RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED](#) = 0, [RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED](#), [RTU_RAYFORMAT_COUNT](#) }
- enum [RTUtriformat](#) { [RTU_TRIFORMAT_MESH](#) = 0, [RTU_TRIFORMAT_TRIANGLE_SOUP](#), [RTU_TRIFORMAT_COUNT](#) }
- enum [RTUinitoptions](#) { [RTU_INITOPTION_NONE](#) = 0, [RTU_INITOPTION_GPU_ONLY](#) = 1 << 0, [RTU_INITOPTION_CPU_ONLY](#) = 1 << 1, [RTU_INITOPTION_CULL_BACKFACE](#) = 1 << 2 }
- enum [RTUoutput](#) { [RTU_OUTPUT_NONE](#) = 0, [RTU_OUTPUT_NORMAL](#) = 1 << 0, [RTU_OUTPUT_BARYCENTRIC](#) = 1 << 1, [RTU_OUTPUT_BACKFACING](#) = 1 << 2 }
- enum [RTUoption](#) { [RTU_OPTION_INT_NUM_THREADS](#) = 0 }

Functions

- RTResult RTAPI [rtuTraversalCreate](#) ([RTUtraversal](#) *traversal, [RTUquerytype](#) query_type, [RTUrayformat](#) ray_format, [RTUtriformat](#) tri_format, unsigned int outputs, unsigned int options, RTcontext context)
- RTResult RTAPI [rtuTraversalGetErrorString](#) ([RTUtraversal](#) traversal, RTresult code, const char **return_string)
- RTResult RTAPI [rtuTraversalSetOption](#) ([RTUtraversal](#) traversal, [RTUoption](#) option, void *value)

- `RTresult RTAPI rtuTraversalSetMesh` (`RTUtraversal` traversal, unsigned int num_verts, const float *verts, unsigned int num_tris, const unsigned *indices)
- `RTresult RTAPI rtuTraversalSetTriangles` (`RTUtraversal` traversal, unsigned int num_tris, const float *tris)
- `RTresult RTAPI rtuTraversalSetAccelData` (`RTUtraversal` traversal, const void *data, `RTsize` data_size)
- `RTresult RTAPI rtuTraversalGetAccelDataSize` (`RTUtraversal` traversal, `RTsize` *data_size)
- `RTresult RTAPI rtuTraversalGetAccelData` (`RTUtraversal` traversal, void *data)
- `RTresult RTAPI rtuTraversalMapRays` (`RTUtraversal` traversal, unsigned int num_rays, float **rays)
- `RTresult RTAPI rtuTraversalUnmapRays` (`RTUtraversal` traversal)
- `RTresult RTAPI rtuTraversalPreprocess` (`RTUtraversal` traversal)
- `RTresult RTAPI rtuTraversalTraverse` (`RTUtraversal` traversal)
- `RTresult RTAPI rtuTraversalMapResults` (`RTUtraversal` traversal, `RTUtraversalresult` **results)
- `RTresult RTAPI rtuTraversalUnmapResults` (`RTUtraversal` traversal)
- `RTresult RTAPI rtuTraversalMapOutput` (`RTUtraversal` traversal, `RTUoutput` which, void **output)
- `RTresult RTAPI rtuTraversalUnmapOutput` (`RTUtraversal` traversal, `RTUoutput` which)
- `RTresult RTAPI rtuTraversalDestroy` (`RTUtraversal` traversal)

3.5.1 Detailed Description

A simple API for performing raytracing queries using OptiX or the CPU.

Definition in file [optixu_traversal.h](#).

3.5.2 Typedef Documentation

3.5.2.1 typedef struct `RTUtraversal_api*` `RTUtraversal`

Opaque type. Note that the *_api types should never be used directly. Only the typedef target names will be guaranteed to remain unchanged.

Definition at line 116 of file [optixu_traversal.h](#).

3.5.3 Enumeration Type Documentation

3.5.3.1 enum `RTUinitoptions`

Initialization options (static across life of traversal object).

The `rtuTraverse` API supports both running on the CPU and GPU. When `RTU_INIT_OPTION_NONE` is specified GPU context creation is attempted. If that fails (such as when there isn't an NVIDIA GPU part present, the CPU code path is automatically

chosen. Specifying `RTU_INITOPTION_GPU_ONLY` or `RTU_INITOPTION_CPU_ONLY` will only use the GPU or CPU modes without automatic transitions from one to the other.

`RTU_INITOPTION_CULL_BACKFACE` will enable back face culling during intersection.

Enumerator:

`RTU_INITOPTION_NONE`
`RTU_INITOPTION_GPU_ONLY`
`RTU_INITOPTION_CPU_ONLY`
`RTU_INITOPTION_CULL_BACKFACE`

Definition at line 89 of file [optixu_traversal.h](#).

3.5.3.2 enum `RTUoption`

Runtime options (can be set multiple times for a given traversal object).

Enumerator:

`RTU_OPTION_INT_NUM_THREADS`

Definition at line 107 of file [optixu_traversal.h](#).

3.5.3.3 enum `RTUoutput`

Enumerator:

`RTU_OUTPUT_NONE`
`RTU_OUTPUT_NORMAL`
`RTU_OUTPUT_BARYCENTRIC`
`RTU_OUTPUT_BACKFACING`

Definition at line 96 of file [optixu_traversal.h](#).

3.5.3.4 enum `RTUquerytype`

The type of ray query to be performed.

See OptiX Programming Guide for explanation of any vs. closest hit queries. Note that in the case of `RTU_QUERY_TYPE_ANY_HIT`, the `prim_id` and `t` intersection values in [RTUtraversalresult](#) will correspond to the first successful intersection. These values may not be indicative of the closest intersection, only that there was at least one.

Enumerator:

`RTU_QUERY_TYPE_ANY_HIT` Perform any hit calculation
`RTU_QUERY_TYPE_CLOSEST_HIT` Perform closest hit calculation
`RTU_QUERY_TYPE_COUNT`

Definition at line 49 of file [optixu_traversal.h](#).

3.5.3.5 enum RTUrayformat

The input format of the ray vector.

Enumerator:

RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED
RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED
RTU_RAYFORMAT_COUNT

Definition at line 58 of file [optixu_traversal.h](#).

3.5.3.6 enum RTUtriformat

The input format of the triangles.

TRIANGLE_SOUP implies future use of `rtuTraversalSetTriangles` while MESH implies use of `rtuTraversalSetMesh`.

Enumerator:

RTU_TRIFORMAT_MESH
RTU_TRIFORMAT_TRIANGLE_SOUP
RTU_TRIFORMAT_COUNT

Definition at line 70 of file [optixu_traversal.h](#).

3.5.4 Function Documentation

3.5.4.1 RTresult RTAPI `rtuTraversalCreate (RTUtraversal * traversal, RTUquerytype query_type, RTUrayformat ray_format, RTUtriformat tri_format, unsigned int outputs, unsigned int options, RTcontext context)`

Create a traversal state and associate a context with it. If context is a null pointer a new context will be created internally. The context should also not be used for any other launch commands from the OptiX host API, nor attached to multiple RTUtraversal objects at one time.

Parameters

out	<i>traversal</i>	Return pointer for traverse state handle
	<i>query_type</i>	Ray query type
	<i>ray_format</i>	Ray format
	<i>tri_format</i>	Triangle format
	<i>outputs</i>	OR'ed mask of requested RTUoutputs
	<i>options</i>	Bit vector of or'ed RTUinitoptions.
	<i>context</i>	RTcontext used for internal object creation

3.5.4.2 RTresult RTAPI rtuTraversalDestroy (RTUtraversal *traversal*)

Clean up any internal memory associated with rtuTraversal operations. Includes destruction of result buffers returned via rtuTraversalGetResults. Invalidates traversal object.

Parameters

<i>traversal</i>	Traversal state handle
------------------	------------------------

3.5.4.3 RTresult RTAPI rtuTraversalGetAccelData (RTUtraversal *traversal*, void * *data*)

Retrieve acceleration data for current geometry. Will force acceleration build if necessary. The data parameter should be preallocated and its length should match return value of rtuTraversalGetAccelDataSize.

Parameters

	<i>traversal</i>	Traversal state handle
out	<i>data</i>	Acceleration data

3.5.4.4 RTresult RTAPI rtuTraversalGetAccelDataSize (RTUtraversal *traversal*, RTsize * *data_size*)

Retrieve acceleration data size for current geometry. Will force acceleration build if necessary.

Parameters

	<i>traversal</i>	Traversal state handle
out	<i>data_size</i>	Size of acceleration data

3.5.4.5 RTresult RTAPI rtuTraversalGetErrorString (RTUtraversal *traversal*, RTresult *code*, const char ** *return_string*)

Returns the string associated with the error code and any additional information from the last error. If traversal is non-NULL return_string only remains valid while traversal is live.

Parameters

	<i>traversal</i>	Traversal state handle. Can be NULL.
	<i>code</i>	Error code from last error
out	<i>return_string</i>	Pointer to string with error message in it.

3.5.4.6 RTresult RTAPI rtuTraversalMapOutput (RTUtraversal *traversal*, RTUoutput *which*, void ** *output*)

Retrieve user-specified output from last rtuTraversal call. Output can be copied from the pointer returned by rtuTraversalMapOutput and will have length 'num_rays' from as

prescribed from the previous call to `rtuTraversalSetRays`. For each `RTUoutput`, a single `rtuTraversalMapOutput` pointers can be outstanding. `rtuTraversalUnmapOutput` should be called when finished reading the output.

If requested output type was not turned on with a previous call to `rtuTraverseSetOutputs` an error will be returned. See `RTUoutput` enum for description of output data formats for various outputs.

Parameters

	<i>traversal</i>	Traversal state handle
	<i>which</i>	Output type to be specified
out	<i>output</i>	Pointer to output from last traverse

3.5.4.7 RTResult RTAPI `rtuTraversalMapRays` (`RTUtraversal traversal`, unsigned int `num_rays`, float ** `rays`)

Specify set of rays to be cast upon next call to `rtuTraversalTraverse`. `rtuTraversalMapRays` obtains a pointer which can be used to copy the ray data into. Rays should be packed in the format described in `rtuTraversalCreate` call. When copying is completed `rtuTraversalUnmapRays` should be called. Note that this call invalidates any existing results buffers until `rtuTraversalTraverse` is called again.

Parameters

	<i>traversal</i>	Traversal state handle
	<i>num_rays</i>	Number of rays to be traced
	<i>rays</i>	Pointer to ray data

3.5.4.8 RTResult RTAPI `rtuTraversalMapResults` (`RTUtraversal traversal`, `RTUtraversalresult ** results`)

Retrieve results of last `rtuTraversal` call. Results can be copied from the pointer returned by `rtuTraversalMapResults` and will have length '`num_rays`' as prescribed from the previous call to `rtuTraversalMapRays`. `rtuTraversalUnmapResults` should be called when finished reading the results. Returned primitive ID of -1 indicates a ray miss.

Parameters

	<i>traversal</i>	Traversal state handle
out	<i>results</i>	Pointer to results of last traverse

3.5.4.9 RTResult RTAPI `rtuTraversalPreprocess` (`RTUtraversal traversal`)

Perform any necessary preprocessing (eg, acceleration structure building, optix context compilation). It is not necessary to call this function as `rtuTraversalTraverse` will call this internally as necessary.

Parameters

<i>traversal</i>	Traversal state handle
------------------	------------------------

3.5.4.10 RTresult RTAPI rtuTraversalSetAccelData (RTUtraversal *traversal*, const void * *data*, RTsize *data_size*)

Specify acceleration data for current geometry. Input acceleration data should be result of rtuTraversalGetAccelData or rtAccelerationGetData call.

Parameters

<i>traversal</i>	Traversal state handle
<i>data</i>	Acceleration data
<i>data_size</i>	Size of acceleration data

3.5.4.11 RTresult RTAPI rtuTraversalSetMesh (RTUtraversal *traversal*, unsigned int *num_verts*, const float * *verts*, unsigned int *num_tris*, const unsigned * *indices*)

Specify triangle mesh to be intersected by the next call to rtuTraversalLaunch. Only one geometry set may be active at a time. Subsequent calls to rtuTraversalSetTriangles or rtuTraversalSetMesh will override any previously specified geometry. No internal copies of the mesh data are made. The user should ensure that the mesh data remains valid until after rtuTraversalTraverse has been called. Counter-clockwise winding is assumed for normal and backfacing computations.

Parameters

<i>traversal</i>	Traversal state handle
<i>num_verts</i>	Vertex count
<i>verts</i>	Vertices [v1_x, v1_y, v1_z, v2.x, ...]
<i>num_tris</i>	Triangle count
<i>indices</i>	Indices [tri1_index1, tri_index2, ...]

3.5.4.12 RTresult RTAPI rtuTraversalSetOption (RTUtraversal *traversal*, RTUoption *option*, void * *value*)

Set a runtime option. Unlike initialization options, these options may be set more than once for a given RTUtraversal instance.

Parameters

<i>traversal</i>	Traversal state handle
<i>option</i>	The option to be set
<i>value</i>	Value of the option

3.5.4.13 RTresult RTAPI rtuTraversalSetTriangles (RTUtraversal *traversal*, unsigned int *num_tris*, const float * *tris*)

Specify triangle soup to be intersected by the next call to `rtuTraversalLaunch`. Only one geometry set may be active at a time. Subsequent calls to `rtuTraversalSetTriangles` or `rtuTraversalSetMesh` will override any previously specified geometry. No internal copies of the triangle data are made. The user should ensure that the triangle data remains valid until after `rtuTraversalTraverse` has been called. Counter-clockwise winding is assumed for normal and backfacing computations.

Parameters

<i>traversal</i>	Traversal state handle
<i>num_tris</i>	Triangle count
<i>tris</i>	Triangles [<i>tri1_v1.x</i> , <i>tri1_v1.y</i> , <i>tri1_v1.z</i> , <i>tri1_v2.x</i> , ...]

3.5.4.14 RTresult RTAPI rtuTraversalTraverse (RTUtraversal *traversal*)

Perform any necessary preprocessing (eg, acceleration structure building and kernel compilation) and cast current rays against current geometry.

Parameters

<i>traversal</i>	Traversal state handle
------------------	------------------------

3.5.4.15 RTresult RTAPI rtuTraversalUnmapOutput (RTUtraversal *traversal*, RTUOutput *which*)

See `rtuTraversalMapOutput`

3.5.4.16 RTresult RTAPI rtuTraversalUnmapRays (RTUtraversal *traversal*)

See `rtuTraversalMapRays`.

3.5.4.17 RTresult RTAPI rtuTraversalUnmapResults (RTUtraversal *traversal*)

See `rtuTraversalMapResults`

3.6 optixu_traversal.h

```

00001
00002
00003 /*****\
00004 *
00005 * Traversal API
00006 *
00007 \*****/
00008
00023 #ifndef __optixu_optux_traversal_h_
00024 #define __optixu_optux_traversal_h_
00025
00026 #include "../optix.h"
00027
00028 #ifdef __cplusplus

```



```

00029 extern "C" {
00030 #endif
00031
00035 typedef struct {
00036     int    prim_id;
00037     float  t;
00038 } RTUtraversalresult;
00039
00040
00049 typedef enum {
00050     RTU_QUERY_TYPE_ANY_HIT = 0,
00051     RTU_QUERY_TYPE_CLOSEST_HIT,
00052     RTU_QUERY_TYPE_COUNT
00053 } RTUquerytype;
00054
00058 typedef enum {
00059     RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED = 0,
00060     RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED,
00061     RTU_RAYFORMAT_COUNT
00062 } RTUrayformat;
00063
00070 typedef enum {
00071     RTU_TRIFORMAT_MESH= 0,
00072     RTU_TRIFORMAT_TRIANGLE_SOUP,
00073     RTU_TRIFORMAT_COUNT
00074 } RTUtriformat;
00075
00089 typedef enum {
00090     RTU_INITOPTION_NONE           = 0,
00091     RTU_INITOPTION_GPU_ONLY      = 1 << 0,
00092     RTU_INITOPTION_CPU_ONLY      = 1 << 1,
00093     RTU_INITOPTION_CULL_BACKFACE = 1 << 2
00094 } RTUinitoptions;
00095
00096 typedef enum {
00097     RTU_OUTPUT_NONE           = 0,
00098     RTU_OUTPUT_NORMAL        = 1 << 0, /*< float3 [x, y, z]
00099 */
00099     RTU_OUTPUT_BARYCENTRIC   = 1 << 1, /*< float2 [alpha, beta] (gamma implicit)
00100 */
00100     RTU_OUTPUT_BACKFACING    = 1 << 2 /*< char  [1 | 0]
00101 */
00101 } RTUoutput;
00102
00107 typedef enum {
00108     RTU_OPTION_INT_NUM_THREADS=0
00109 } RTUoption;
00110
00111
00116 typedef struct RTUtraversal_api* RTUtraversal;
00117
00118
00133 RTresult RTAPI rtuTraversalCreate( RTUtraversal* traversal,
00134                                   RTUquerytype  query_type,
00135                                   RTUrayformat   ray_format,
00136                                   RTUtriformat   tri_format,
00137                                   unsigned int   outputs,
00138                                   unsigned int   options,
00139                                   RTcontext      context );
00140
00150 RTresult RTAPI rtuTraversalGetErrorString( RTUtraversal traversal,
00151                                           RTresult code,
00152                                           const char** return_string);
00161 RTresult RTAPI rtuTraversalSetOption( RTUtraversal traversal,
00162                                       RTUoption   option,
00163                                       void*       value );
00164
00180 RTresult RTAPI rtuTraversalSetMesh( RTUtraversal traversal,
00181                                     unsigned int num_verts,
00182                                     const float*  verts,
00183                                     unsigned int  num_tris,
00184                                     const unsigned* indices );
00185
00200 RTresult RTAPI rtuTraversalSetTriangles( RTUtraversal traversal,

```

```
00201             unsigned int num_tris,
00202             const float* tris );
00203
00212 RTresult RTAPI rtuTraversalSetAccelData( RTUtraversal traversal,
00213             const void* data,
00214             RTsize      data_size );
00215
00223 RTresult RTAPI rtuTraversalGetAccelDataSize( RTUtraversal traversal,
00224             RTsize*      data_size );
00225
00234 RTresult RTAPI rtuTraversalGetAccelData( RTUtraversal traversal,
00235             void*      data );
00236
00249 RTresult RTAPI rtuTraversalMapRays( RTUtraversal traversal,
00250             unsigned int num_rays,
00251             float** rays );
00252
00256 RTresult RTAPI rtuTraversalUnmapRays( RTUtraversal traversal );
00257
00265 RTresult RTAPI rtuTraversalPreprocess( RTUtraversal traversal );
00266
00273 RTresult RTAPI rtuTraversalTraverse( RTUtraversal traversal );
00274
00285 RTresult RTAPI rtuTraversalMapResults( RTUtraversal traversal,
00286             RTUtraversalresult** results );
00287
00291 RTresult RTAPI rtuTraversalUnmapResults( RTUtraversal traversal );
00292
00309 RTresult RTAPI rtuTraversalMapOutput( RTUtraversal traversal,
00310             RTUoutput which,
00311             void** output );
00315 RTresult RTAPI rtuTraversalUnmapOutput( RTUtraversal traversal,
00316             RTUoutput which );
00324 RTresult RTAPI rtuTraversalDestroy( RTUtraversal traversal );
00325
00326 #ifndef __cplusplus
00327 } /* extern "C" */
00328 #endif
00329
00330 #endif /* _optixu_optux_traversal.h */
00331
```