OptiX Utility Library

2.5.0

Generated by Doxygen 1.6.1

Tue Dec 6 15:09:21 2011

Contents

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 113 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_INITOPTION_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 86 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 104 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 93 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.

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 46 of file optixu_traversal.h.

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 55 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 67 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:

→ ***traversal*** Return pointer for traverse state handle

query_type Ray query type

ray_format Ray format

tri_format Triangle format

outputs OR'ed mask of requested RTUoutputs

options Bit vector of or'ed RTUinitoptions.

context RTcontext used for internal object creation

1.1.4.2 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:

traversal Traversal state handle

1.1.4.3 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:

traversal Traversal state handle

→ ***data*** Acceleration data

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

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

Parameters:

traversal Traversal state handle

→ ***data_size*** Size of acceleration data

1.1.4.5 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:

traversal Traversal state handle. Can be NULL.

code Error code from last error

→ ***return_string*** Pointer to string with error message in it.

1.1.4.6 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:

traversal Traversal state handle

which Output type to be specified

→ ***output*** 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:

traversal Traversal state handle

num_rays Number of rays to be traced

rays 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:

traversal Traversal state handle

→ ***results*** 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.

Parameters:

traversal Traversal state handle

1.1.4.10 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:

traversal Traversal state handle

data Acceleration data

data_size Size of acceleration data

1.1.4.11 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:

traversal Traversal state handle

num_verts Vertex count

verts Vertices [v1_x, v1_y, v1_z, v2.x, ...]

num_tris Triangle count

indices Indices [tri1_index1, tr1_index2, ...]

1.1.4.12 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:

traversal Traversal state handle

option The option to be set

value Value of the option

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:

traversal Traversal state handle

num_tris Triangle count

tris Triangles [tri1_v1.x, tri1_v1.y, tr1_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:

traversal 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::operator[ ] (const std::string &varname)
- Handle< VariableObj > optix::Handle::operator[ ] (const char *varname)
- virtual void optix::APIObj::checkError (RTresult code)
- void optix::APIObj::checkErrorNoGetContext (RTresult code)
- Context optix::ContextObj::getContext ()
- 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 ()
- RTcontext optix::ContextObj::get ()
- void optix::ProgramObj::destroy ()
- void optix::ProgramObj::validate ()
- Context optix::ProgramObj::getContext ()
- Variable optix::ProgramObj::declareVariable (const std::string &name)
- Variable optix::ProgramObj::queryVariable (const std::string &name)
- void optix::ProgramObj::removeVariable (Variable v)
- unsigned int optix::ProgramObj::getVariableCount ()
- Variable optix::ProgramObj::getVariable (unsigned int index)
- RTprogram optix::ProgramObj::get ()
- void optix::GroupObj::destroy ()
- void optix::GroupObj::validate ()
- Context optix::GroupObj::getContext ()
- void optix::SelectorObj::destroy ()
- void optix::SelectorObj::validate ()
- Context optix::SelectorObj::getContext ()
- RTselector optix::SelectorObj::get ()
- RTgroup optix::GroupObj::get ()
- void optix::GeometryGroupObj::destroy ()
- void optix::GeometryGroupObj::validate ()
- Context optix::GeometryGroupObj::getContext ()
- RTgeometrygroup optix::GeometryGroupObj::get ()
- void optix::TransformObj::destroy ()
- void optix::TransformObj::validate ()
- Context optix::TransformObj::getContext ()
- RTtransform optix::TransformObj::get ()
- void optix::AccelerationObj::destroy ()
- void optix::AccelerationObj::validate ()
- Context optix::AccelerationObj::getContext ()
- RTacceleration optix::AccelerationObj::get ()
- void optix::GeometryInstanceObj::destroy ()
- void optix::GeometryInstanceObj::validate ()
- Context optix::GeometryInstanceObj::getContext ()
- RTgeometryinstance optix::GeometryInstanceObj::get ()
- void optix::GeometryObj::destroy ()
- void optix::GeometryObj::validate ()
- Context optix::GeometryObj::getContext ()
- RTgeometry optix::GeometryObj::get ()
- void optix::MaterialObj::destroy ()
- void optix::MaterialObj::validate ()
- Context optix::MaterialObj::getContext ()
- RTmaterial optix::MaterialObj::get ()
- void optix::TextureSamplerObj::destroy ()

- void optix::TextureSamplerObj::validate ()
- Context optix::TextureSamplerObj::getContext ()
- RTtexturesampler optix::TextureSamplerObj::get ()
- void optix::BufferObj::destroy ()
- void optix::BufferObj::validate ()
- Context optix::BufferObj::getContext ()
- RTbuffer optix::BufferObj::get ()
- Context optix::VariableObj::getContext ()
- std::string optix::VariableObj::getName ()
- std::string optix::VariableObj::getAnnotation ()
- RTobjecttype optix::VariableObj::getType ()
- RTvariable optix::VariableObj::get ()
- RTsize optix::VariableObj::getSize ()

- void optix::ContextObj::checkError (RTresult code)
- 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::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 ()
- std::string optix::ContextObj::getErrorString (RTresult code)
- template<class Iterator >
  void optix::ContextObj::setDevices (Iterator begin, Iterator end)
- std::vector< int > optix::ContextObj::getEnabledDevices ()
- unsigned int optix::ContextObj::getEnabledDeviceCount ()

- int optix::ContextObj::getMaxTextureCount ()
- int optix::ContextObj::getCPUNumThreads ()
- RTsize optix::ContextObj::getUsedHostMemory ()
- int optix::ContextObj::getGPUPagingActive ()
- int optix::ContextObj::getGPUPagingForcedOff ()
- RTsize optix::ContextObj::getAvailableDeviceMemory (int ordinal)
- 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 ()
- void optix::ContextObj::setTimeoutCallback (RTtimeoutcallback callback, double min_polling_seconds)
- void optix::ContextObj::setEntryPointCount (unsigned int num_entry_points)
- unsigned int optix::ContextObj::getEntryPointCount ()
- void optix::ContextObj::setRayGenerationProgram (unsigned int entry_point_index, Program program)
- Program optix::ContextObj::getRayGenerationProgram (unsigned int entry_point_index)
- void optix::ContextObj::setExceptionProgram (unsigned int entry_point_index, Program program)
- Program optix::ContextObj::getExceptionProgram (unsigned int entry_point_index)
- void optix::ContextObj::setExceptionEnabled (RTexception exception, bool enabled)
- bool optix::ContextObj::getExceptionEnabled (RTexception exception)
- void optix::ContextObj::setRayTypeCount (unsigned int num_ray_types)
- unsigned int optix::ContextObj::getRayTypeCount ()
- void optix::ContextObj::setMissProgram (unsigned int ray_type_index, Program program)
- Program optix::ContextObj::getMissProgram (unsigned int ray_type_index)
- 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 ()
- void optix::ContextObj::setPrintBufferSize (RTsize buffer_size_bytes)
- RTsize optix::ContextObj::getPrintBufferSize ()
- void optix::ContextObj::setPrintLaunchIndex (int x, int y=-1, int z=-1)
- optix::int3 optix::ContextObj::getPrintLaunchIndex ()
- Variable optix::ContextObj::declareVariable (const std::string &name)
- Variable optix::ContextObj::queryVariable (const std::string &name)
- void optix::ContextObj::removeVariable (Variable v)
- unsigned int optix::ContextObj::getVariableCount ()
- Variable optix::ContextObj::getVariable (unsigned int index)

- void optix::SelectorObj::setVisitProgram (Program program)
- Program optix::SelectorObj::getVisitProgram ()
- void optix::SelectorObj::setChildCount (unsigned int count)
- unsigned int optix::SelectorObj::getChildCount ()
- template<typename T >
  void optix::SelectorObj::setChild (unsigned int index, T child)
- template<typename T >
  T optix::SelectorObj::getChild (unsigned int index)

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

- void optix::GroupObj::setAcceleration (Acceleration acceleration)
- Acceleration optix::GroupObj::getAcceleration ()
- void optix::GroupObj::setChildCount (unsigned int count)
- unsigned int optix::GroupObj::getChildCount ()
- template<typename T >
  void optix::GroupObj::setChild (unsigned int index, T child)
- template<typename T >
  T optix::GroupObj::getChild (unsigned int index)

- void optix::GeometryGroupObj::setAcceleration (Acceleration acceleration)
- Acceleration optix::GeometryGroupObj::getAcceleration ()
- void optix::GeometryGroupObj::setChildCount (unsigned int count)
- unsigned int optix::GeometryGroupObj::getChildCount ()
- void optix::GeometryGroupObj::setChild (unsigned int index, GeometryInstance geometryinstance)
- GeometryInstance optix::GeometryGroupObj::getChild (unsigned int index)

- template<typename T >
  void optix::TransformObj::setChild (T child)
- template<typename T >
  T optix::TransformObj::getChild ()
- void optix::TransformObj::setMatrix (bool transpose, const float *matrix, const float *inverse_matrix)
- void optix::TransformObj::getMatrix (bool transpose, float *matrix, float *inverse_matrix)

- void optix::AccelerationObj::markDirty ()
- bool optix::AccelerationObj::isDirty ()
- void optix::AccelerationObj::setProperty (const std::string &name, const std::string &value)
- std::string optix::AccelerationObj::getProperty (const std::string &name)
- void optix::AccelerationObj::setBuilder (const std::string &builder)
- std::string optix::AccelerationObj::getBuilder ()
- void optix::AccelerationObj::setTraverser (const std::string &traverser)
- std::string optix::AccelerationObj::getTraverser ()
- RTsize optix::AccelerationObj::getDataSize ()
- void optix::AccelerationObj::getData (void *data)
- void optix::AccelerationObj::setData (const void *data, RTsize size)

- void optix::GeometryInstanceObj::setGeometry (Geometry geometry)
- Geometry optix::GeometryInstanceObj::getGeometry ()
- void optix::GeometryInstanceObj::setMaterialCount (unsigned int count)
- unsigned int optix::GeometryInstanceObj::getMaterialCount ()

- void optix::GeometryInstanceObj::setMaterial (unsigned int idx, Material material)
- Material optix::GeometryInstanceObj::getMaterial (unsigned int idx)
- unsigned int optix::GeometryInstanceObj::addMaterial (Material material)
- Variable optix::GeometryInstanceObj::declareVariable (const std::string &name)
- Variable optix::GeometryInstanceObj::queryVariable (const std::string &name)
- void optix::GeometryInstanceObj::removeVariable (Variable v)
- unsigned int optix::GeometryInstanceObj::getVariableCount ()
- Variable optix::GeometryInstanceObj::getVariable (unsigned int index)

- void optix::GeometryObj::setPrimitiveCount (unsigned int num_primitives)
- unsigned int optix::GeometryObj::getPrimitiveCount ()
- void optix::GeometryObj::setBoundingBoxProgram (Program program)
- Program optix::GeometryObj::getBoundingBoxProgram ()
- void optix::GeometryObj::setIntersectionProgram (Program program)
- Program optix::GeometryObj::getIntersectionProgram ()
- Variable optix::GeometryObj::declareVariable (const std::string &name)
- Variable optix::GeometryObj::queryVariable (const std::string &name)
- void optix::GeometryObj::removeVariable (Variable v)
- unsigned int optix::GeometryObj::getVariableCount ()
- Variable optix::GeometryObj::getVariable (unsigned int index)
- void optix::GeometryObj::markDirty ()
- bool optix::GeometryObj::isDirty ()

- void optix::MaterialObj::setClosestHitProgram (unsigned int ray_type_index, Program program)
- Program optix::MaterialObj::getClosestHitProgram (unsigned int ray_type_index)
- void optix::MaterialObj::setAnyHitProgram (unsigned int ray_type_index, Program program)
- Program optix::MaterialObj::getAnyHitProgram (unsigned int ray_type_index)
- Variable optix::MaterialObj::declareVariable (const std::string &name)
- Variable optix::MaterialObj::queryVariable (const std::string &name)
- void optix::MaterialObj::removeVariable (Variable v)
- unsigned int optix::MaterialObj::getVariableCount ()
- Variable optix::MaterialObj::getVariable (unsigned int index)

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

- RTtextureindexmode optix::TextureSamplerObj::getIndexingMode ()
- 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)
- void optix::TextureSamplerObj::registerGLTexture ()
- void optix::TextureSamplerObj::unregisterGLTexture ()

- void optix::BufferObj::setFormat (RTformat format)
- RTformat optix::BufferObj::getFormat ()
- void optix::BufferObj::setElementSize (RTsize size_of_element)
- RTsize optix::BufferObj::getElementSize ()
- void optix::BufferObj::setSize (RTsize width)
- void optix::BufferObj::getSize (RTsize &width)
- void optix::BufferObj::setSize (RTsize width, RTsize height)
- void optix::BufferObj::getSize (RTsize &width, RTsize &height)
- void optix::BufferObj::setSize (RTsize width, RTsize height, RTsize depth)
- void optix::BufferObj::getSize (RTsize &width, RTsize &height, RTsize &depth)
- void optix::BufferObj::setSize (unsigned int dimensionality, const RTsize *dims)
- void optix::BufferObj::getSize (unsigned int dimensionality, RTsize *dims)
- unsigned int optix::BufferObj::getDimensionality ()
- unsigned int optix::BufferObj::getGLBOId ()
- void optix::BufferObj::registerGLBuffer ()
- void optix::BufferObj::unregisterGLBuffer ()
- void * optix::BufferObj::map ()
- void optix::BufferObj::unmap ()

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::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)

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)

Numeric value getters

Query value of a variable with scalar numeric value

- float optix::VariableObj::getFloat ()
- unsigned int optix::VariableObj::getUint ()
- int optix::VariableObj::getInt ()

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)

User data variable accessors

- void optix::VariableObj::setUserData (RTsize size, const void *ptr)
- void optix::VariableObj::getUserData (RTsize size, void *ptr)

OptiX API object getters

Reitrieve OptiX API object value from a variable

- Buffer optix::VariableObj::getBuffer ()
- TextureSampler optix::VariableObj::getTextureSampler ()

1.2.2 Typedef Documentation

1.2.2.1 typedef Handle<AccelerationObj> optix::Acceleration

Use this to manipulate RTacceleration objects.

Definition at line 208 of file optixpp_namespace.h.

1.2.2.2 typedef Handle<BufferObj> optix::Buffer

Use this to manipulate RTbuffer objects.

Definition at line 209 of file optixpp_namespace.h.

1.2.2.3 typedef Handle<ContextObj> optix::Context

Use this to manipulate RTcontext objects.

Definition at line 210 of file optixpp_namespace.h.

1.2.2.4 typedef Handle<GeometryObj> optix::Geometry

Use this to manipulate RTgeometry objects.

Definition at line 211 of file optixpp_namespace.h.

1.2.2.5 typedef Handle<GeometryGroupObj> optix::GeometryGroup

Use this to manipulate RTgeometrygroup objects.

Definition at line 212 of file optixpp_namespace.h.

1.2.2.6 typedef Handle<GeometryInstanceObj> optix::GeometryInstance

Use this to manipulate RTgeometryinstance objects.

Definition at line 213 of file optixpp_namespace.h.

1.2.2.7 typedef Handle<GroupObj> optix::Group

Use this to manipulate RTgroup objects.

Definition at line 214 of file optixpp_namespace.h.

1.2.2.8 typedef Handle<MaterialObj> optix::Material

Use this to manipulate RTmaterial objects.

Definition at line 215 of file optixpp_namespace.h.

1.2.2.9 typedef Handle<ProgramObj> optix::Program

Use this to manipulate RTprogram objects.

Definition at line 216 of file optixpp_namespace.h.

1.2.2.10 typedef Handle<SelectorObj> optix::Selector

Use this to manipulate RTselector objects.

Definition at line 217 of file optixpp_namespace.h.

1.2.2.11 typedef Handle<TextureSamplerObj> optix::TextureSampler

Use this to manipulate RTtexturesampler objects.

Definition at line 218 of file optixpp_namespace.h.

1.2.2.12 typedef Handle<TransformObj> optix::Transform

Use this to manipulate RTtransform objects.

Definition at line 219 of file optixpp_namespace.h.

1.2.2.13 typedef Handle<VariableObj> optix::Variable

Use this to manipulate RTvariable objects.

Definition at line 220 of file optixpp_namespace.h.

1.2.3 Function Documentation

1.2.3.1 unsigned int optix::GeometryInstanceObj::addMaterial (Material *material*) [inline, inherited]

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

Definition at line 2537 of file optixpp_namespace.h.

1.2.3.2 void optix::ContextObj::checkError (RTresult *code*) [inline, virtual, inherited]

See APIObj::checkError

Reimplemented from optix::APIObj.

Definition at line 1506 of file optixpp_namespace.h.

1.2.3.3 void optix::APIObj::checkError (RTresult *code*) [inline, virtual, inherited]

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 1486 of file optixpp_namespace.h.

1.2.3.4 void optix::APIObj::checkErrorNoGetContext (RTresult *code*) [inline, inherited]

Definition at line 1494 of file optixpp_namespace.h.

1.2.3.5 void optix::ContextObj::compile () [inline, inherited]

See rtContextCompile.

Definition at line 1974 of file optixpp_namespace.h.

1.2.3.6 Context optix::ContextObj::create () [inline, static, inherited]

Creates a Context object. See rtContextCreate.

Definition at line 1537 of file optixpp_namespace.h.

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

See rtAccelerationCreate

Definition at line 1557 of file optixpp_namespace.h.

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

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

Definition at line 1600 of file optixpp_namespace.h.

1.2.3.9 Buffer optix::ContextObj::createBuffer (unsigned int *type*, RTformat *format*, RTsize *width*, RTsize *height*) [inline, inherited]

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

Definition at line 1591 of file optixpp_namespace.h.

1.2.3.10 Buffer optix::ContextObj::createBuffer (unsigned int *type*, RTformat *format*, RTsize *width*) [inline, inherited]

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

Definition at line 1582 of file optixpp_namespace.h.

1.2.3.11 Buffer optix::ContextObj::createBuffer (unsigned int *type*, RTformat *format*) [inline, inherited]

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

Definition at line 1574 of file optixpp_namespace.h.

1.2.3.12 Buffer optix::ContextObj::createBuffer (unsigned int *type*) [inline, inherited]

Create a buffer with given RTbuffertype. See rtBufferCreate.

Definition at line 1567 of file optixpp_namespace.h.

1.2.3.13 Buffer optix::ContextObj::createBufferFromGLBO (unsigned int *type*, unsigned int *vbo*) [inline, inherited]

Create buffer from GL buffer object. See rtBufferCreateFromGLBO.

Definition at line 1609 of file optixpp_namespace.h.

1.2.3.14 Geometry optix::ContextObj::createGeometry () [inline, inherited]

See rtGeometryCreate.

Definition at line 1684 of file optixpp_namespace.h.

1.2.3.15 template<class Iterator > GeometryGroup optix::ContextObj::createGeometryGroup (Iterator *childbegin*, Iterator *childend*) [inline, inherited]

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

Definition at line 1742 of file optixpp_namespace.h.

1.2.3.16 GeometryGroup optix::ContextObj::createGeometryGroup () [inline, inherited]

See rtGeometryGroupCreate.

Definition at line 1734 of file optixpp_namespace.h.

1.2.3.17 template<class Iterator > GeometryInstance optix::ContextObj::createGeometryInstance (Geometry *geometry*, Iterator *matlbegin*, Iterator *matlend*) [inline, inherited]

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

Definition at line 1699 of file optixpp_namespace.h.

1.2.3.18 GeometryInstance optix::ContextObj::createGeometryInstance () [inline, inherited]

See rtGeometryInstanceCreate.

Definition at line 1691 of file optixpp_namespace.h.

1.2.3.19 template<class Iterator > Group optix::ContextObj::createGroup (Iterator *childbegin*, Iterator *childend*) [inline, inherited]

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

Definition at line 1721 of file optixpp_namespace.h.

1.2.3.20 Group optix::ContextObj::createGroup () [inline, inherited]

See rtGroupCreate.

Definition at line 1713 of file optixpp_namespace.h.

1.2.3.21 Material optix::ContextObj::createMaterial () [inline, inherited]

See rtMaterialCreate.

Definition at line 1762 of file optixpp_namespace.h.

1.2.3.22 Program optix::ContextObj::createProgramFromPTXFile (const std::string & *ptx*, const std::string & *program_name*) [inline, inherited]

See rtProgramCreateFromPTXFile.

Definition at line 1769 of file optixpp_namespace.h.

1.2.3.23 Program optix::ContextObj::createProgramFromPTXString (const std::string & *ptx*, const std::string & *program_name*) [inline, inherited]

See rtProgramCreateFromPTXString.

Definition at line 1776 of file optixpp_namespace.h.

1.2.3.24 Selector optix::ContextObj::createSelector () [inline, inherited]

See rtSelectorCreate.

Definition at line 1783 of file optixpp_namespace.h.

1.2.3.25 TextureSampler optix::ContextObj::createTextureSampler () [inline, inherited]

See rtTextureSamplerCreate.

Definition at line 1790 of file optixpp_namespace.h.

1.2.3.26 TextureSampler optix::ContextObj::createTextureSamplerFromGLImage (unsigned int *id*, RTgltarget *target*) [inline, inherited]

Create TextureSampler from GL image. See rtTextureSamplerCreateFromGLImage.

Definition at line 1677 of file optixpp_namespace.h.

1.2.3.27 Transform optix::ContextObj::createTransform () [inline, inherited]

See rtTransformCreate.

Definition at line 1755 of file optixpp_namespace.h.

1.2.3.28 Variable optix::MaterialObj::declareVariable (const std::string & *name*) [inline, virtual, inherited]

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 2729 of file optixpp_namespace.h.

1.2.3.29 Variable optix::GeometryObj::declareVariable (const std::string & *name*) [inline, virtual, inherited]

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 2637 of file optixpp_namespace.h.

1.2.3.30 Variable optix::GeometryInstanceObj::declareVariable (const std::string & *name*) [inline, virtual, inherited]

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 2545 of file optixpp_namespace.h.

1.2.3.31 Variable optix::SelectorObj::declareVariable (const std::string & *name*) [inline, inherited]

Definition at line 2207 of file optixpp_namespace.h.

1.2.3.32 Variable optix::ProgramObj::declareVariable (const std::string & *name*) [inline, virtual, inherited]

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 2095 of file optixpp_namespace.h.

1.2.3.33 Variable optix::ContextObj::declareVariable (const std::string & *name*) [inline, virtual, inherited]

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 2038 of file optixpp_namespace.h.

1.2.3.34 void optix::BufferObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2928 of file optixpp_namespace.h.

1.2.3.35 void optix::TextureSamplerObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2767 of file optixpp_namespace.h.

1.2.3.36 void optix::MaterialObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2687 of file optixpp_namespace.h.

1.2.3.37 void optix::GeometryObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2583 of file optixpp_namespace.h.

1.2.3.38 void optix::GeometryInstanceObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2482 of file optixpp_namespace.h.

1.2.3.39 void optix::AccelerationObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2394 of file optixpp_namespace.h.

1.2.3.40 void optix::TransformObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2347 of file optixpp_namespace.h.

1.2.3.41 void optix::GeometryGroupObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2288 of file optixpp_namespace.h.

1.2.3.42 void optix::SelectorObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2151 of file optixpp_namespace.h.

1.2.3.43 void optix::GroupObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2133 of file optixpp_namespace.h.

1.2.3.44 void optix::ProgramObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 2077 of file optixpp_namespace.h.

1.2.3.45 void optix::ContextObj::destroy () [inline, virtual, inherited]

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

Implements optix::DestroyableObj.

Definition at line 1546 of file optixpp_namespace.h.

1.2.3.46 RTvariable optix::VariableObj::get () [inline, inherited]

Get the OptiX C API object wrapped by this instance.

Definition at line 3410 of file optixpp_namespace.h.

1.2.3.47 RTbuffer optix::BufferObj::get () [inline, inherited]

Get the underlying OptiX C API RTbuffer opaque pointer.

Definition at line 3102 of file optixpp_namespace.h.

1.2.3.48 RTtexturesampler optix::TextureSamplerObj::get () [inline, inherited]

Get the underlying OptiX C API RTtexturesampler opaque pointer.

Definition at line 2879 of file optixpp_namespace.h.

1.2.3.49 RTmaterial optix::MaterialObj::get () [inline, inherited]

Get the underlying OptiX C API RTmaterial opaque pointer.

Definition at line 2762 of file optixpp_namespace.h.

1.2.3.50 RTgeometry optix::GeometryObj::get () [inline, inherited]

Get the underlying OptiX C API RTgeometry opaque pointer.

Definition at line 2682 of file optixpp_namespace.h.

1.2.3.51 RTgeometryinstance optix::GeometryInstanceObj::get () [inline, inherited]

Get the underlying OptiX C API RTgeometryinstance opaque pointer.

Definition at line 2578 of file optixpp_namespace.h.

1.2.3.52 RTacceleration optix::AccelerationObj::get () [inline, inherited]

Get the underlying OptiX C API RTacceleration opaque pointer.

Definition at line 2477 of file optixpp_namespace.h.

1.2.3.53 RTtransform optix::TransformObj::get () [inline, inherited]

Get the underlying OptiX C API RTtransform opaque pointer.

Definition at line 2389 of file optixpp_namespace.h.

1.2.3.54 RTgeometrygroup optix::GeometryGroupObj::get () [inline, inherited]

Get the underlying OptiX C API RTgeometrygroup opaque pointer.

Definition at line 2342 of file optixpp_namespace.h.

1.2.3.55 RTgroup optix::GroupObj::get () [inline, inherited]

Get the underlying OptiX C API RTgroup opaque pointer.

Definition at line 2283 of file optixpp_namespace.h.

1.2.3.56 RTselector optix::SelectorObj::get () [inline, inherited]

Get the underlying OptiX C API RTselector opaque pointer.

Definition at line 2240 of file optixpp_namespace.h.

1.2.3.57 RTprogram optix::ProgramObj::get () [inline, inherited]

Definition at line 2128 of file optixpp_namespace.h.

1.2.3.58 RTcontext optix::ContextObj::get () [inline, inherited]

Return the OptiX C API RTcontext object.

Definition at line 2072 of file optixpp_namespace.h.

1.2.3.59 Acceleration optix::GeometryGroupObj::getAcceleration () [inline, inherited]

Query the Acceleration structure for this group. See rtGeometryGroupGetAcceleration.

Definition at line 2311 of file optixpp_namespace.h.

1.2.3.60 Acceleration optix::GroupObj::getAcceleration () [inline, inherited]

Query the Acceleration structure for this group. See rtGroupGetAcceleration.

Definition at line 2250 of file optixpp_namespace.h.

1.2.3.61 std::string optix::VariableObj::getAnnotation () [inline, inherited]

Retrieve the annotation associated with the variable.

Definition at line 3396 of file optixpp_namespace.h.

1.2.3.62 Program optix::MaterialObj::getAnyHitProgram (unsigned int *ray_type_index*) [inline, inherited]

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

Definition at line 2722 of file optixpp_namespace.h.

1.2.3.63 unsigned int optix::TextureSamplerObj::getArraySize () [inline, inherited]

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

Definition at line 2802 of file optixpp_namespace.h.

1.2.3.64 RTsize optix::ContextObj::getAvailableDeviceMemory (int *ordinal*) [inline, inherited]

See rtContextGetAttribute.

Definition at line 1862 of file optixpp_namespace.h.

1.2.3.65 Program optix::GeometryObj::getBoundingBoxProgram () [inline, inherited]

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

Definition at line 2618 of file optixpp_namespace.h.

1.2.3.66 Buffer optix::VariableObj::getBuffer () [inline, inherited]

Definition at line 3381 of file optixpp_namespace.h.

1.2.3.67 Buffer optix::TextureSamplerObj::getBuffer (unsigned int *texture_array_idx*, unsigned int *mip_level*) [inline, inherited]

Get the underlying buffer used for texture storage. rtTextureSamplerGetBuffer.

Definition at line 2872 of file optixpp_namespace.h.

1.2.3.68 std::string optix::AccelerationObj::getBuilder () [inline, inherited]

Query the acceleration structure builder. See rtAccelerationGetBuilder.

Definition at line 2441 of file optixpp_namespace.h.

1.2.3.69 template<typename T > T optix::TransformObj::getChild () [inline, inherited]

Set the child node of this transform. See rtTransformGetChild.

Definition at line 2372 of file optixpp_namespace.h.

1.2.3.70 GeometryInstance optix::GeometryGroupObj::getChild (unsigned int *index*) [inline, inherited]

Query an indexed GeometryInstance within this group. See rtGeometryGroupGetChild.

Definition at line 2335 of file optixpp_namespace.h.

1.2.3.71 template<typename T > T optix::GroupObj::getChild (unsigned int *index*) [inline, inherited]

Query an indexed child within this group. See rtGroupGetChild.

Definition at line 2276 of file optixpp_namespace.h.

1.2.3.72 template<typename T > T optix::SelectorObj::getChild (unsigned int *index*) [inline, inherited]

Query an indexed child within this group. See rtSelectorGetChild.

Definition at line 2200 of file optixpp_namespace.h.

1.2.3.73 unsigned int optix::GeometryGroupObj::getChildCount () [inline, inherited]

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

Definition at line 2323 of file optixpp_namespace.h.

1.2.3.74 unsigned int optix::GroupObj::getChildCount () [inline, inherited]

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

Definition at line 2262 of file optixpp_namespace.h.

1.2.3.75 unsigned int optix::SelectorObj::getChildCount () [inline, inherited]

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

Definition at line 2186 of file optixpp_namespace.h.

1.2.3.76 Program optix::MaterialObj::getClosestHitProgram (unsigned int *ray_type_index*) [inline, inherited]

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

Definition at line 2710 of file optixpp_namespace.h.

1.2.3.77 Context optix::VariableObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 3107 of file optixpp_namespace.h.

1.2.3.78 Context optix::BufferObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2939 of file optixpp_namespace.h.

1.2.3.79 Context optix::TextureSamplerObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2778 of file optixpp_namespace.h.

1.2.3.80 Context optix::MaterialObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2698 of file optixpp_namespace.h.

1.2.3.81 Context optix::GeometryObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2594 of file optixpp_namespace.h.

1.2.3.82 Context optix::GeometryInstanceObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2493 of file optixpp_namespace.h.

1.2.3.83 Context optix::AccelerationObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2405 of file optixpp_namespace.h.

1.2.3.84 Context optix::TransformObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2358 of file optixpp_namespace.h.

1.2.3.85 Context optix::GeometryGroupObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2299 of file optixpp_namespace.h.

1.2.3.86 Context optix::SelectorObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2162 of file optixpp_namespace.h.

1.2.3.87 Context optix::GroupObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2144 of file optixpp_namespace.h.

1.2.3.88 Context optix::ProgramObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 2088 of file optixpp_namespace.h.

1.2.3.89 Context optix::ContextObj::getContext () [inline, virtual, inherited]

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

Implements optix::APIObj.

Definition at line 1501 of file optixpp_namespace.h.

1.2.3.90 int optix::ContextObj::getCPUNumThreads () [inline, inherited]

See rtContextGetAttribute.

Definition at line 1834 of file optixpp_namespace.h.

1.2.3.91 void optix::AccelerationObj::getData (void * *data*) [inline, inherited]

Get the marshalled acceleration data. See rtAccelerationGetData.

Definition at line 2467 of file optixpp_namespace.h.

1.2.3.92 RTsize optix::AccelerationObj::getDataSize () [inline, inherited]

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

Definition at line 2460 of file optixpp_namespace.h.

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

Call rtDeviceGetAttribute and return the desired attribute value.

Definition at line 1531 of file optixpp_namespace.h.

1.2.3.94 unsigned int optix::ContextObj::getDeviceCount () [inline, static, inherited]

Call rtDeviceGetDeviceCount and returns number of valid devices.

Definition at line 1512 of file optixpp_namespace.h.

1.2.3.95 std::string optix::ContextObj::getDeviceName (int *ordinal*) [inline, static, inherited]

Call rtDeviceGetAttribute and return the name of the device.

Definition at line 1521 of file optixpp_namespace.h.

1.2.3.96 unsigned int optix::BufferObj::getDimensionality () [inline, inherited]

Query dimensionality of buffer. See rtBufferGetDimensionality.

Definition at line 3010 of file optixpp_namespace.h.

1.2.3.97 RTsize optix::BufferObj::getElementSize () [inline, inherited]

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

Definition at line 2963 of file optixpp_namespace.h.

1.2.3.98 unsigned int optix::ContextObj::getEnabledDeviceCount () [inline, inherited]

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

Definition at line 1820 of file optixpp_namespace.h.

1.2.3.99 std::vector< int > optix::ContextObj::getEnabledDevices () [inline, inherited]

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

Definition at line 1812 of file optixpp_namespace.h.

1.2.3.100 unsigned int optix::ContextObj::getEntryPointCount () [inline, inherited]

See rtContextgetEntryPointCount.

Definition at line 1903 of file optixpp_namespace.h.

1.2.3.101 std::string optix::ContextObj::getErrorString (RTresult *code*) [inline, inherited]

See rtContextGetErrroString.

Definition at line 1797 of file optixpp_namespace.h.

1.2.3.102 bool optix::ContextObj::getExceptionEnabled (RTexception *exception*) [inline, inherited]

See rtContextGetExceptionEnabled.

Definition at line 1942 of file optixpp_namespace.h.

1.2.3.103 Program optix::ContextObj::getExceptionProgram (unsigned int *entry_point_index*) [inline, inherited]

See rtContextGetExceptionProgram.

Definition at line 1929 of file optixpp_namespace.h.

1.2.3.104 void optix::TextureSamplerObj::getFilteringModes (RTfiltermode & *minification*, RTfiltermode & *magnification*, RTfiltermode & *mipmapping*) [inline, inherited]

Query filtering modes for this sampler. See rtTextureSamplerGetFilteringModes.

Definition at line 2826 of file optixpp_namespace.h.

1.2.3.105 float optix::VariableObj::getFloat () [inline, inherited]

Definition at line 3310 of file optixpp_namespace.h.

1.2.3.106 RTformat optix::BufferObj::getFormat () [inline, inherited]

Query the data format for the buffer. See rtBufferGetFormat.

Definition at line 2951 of file optixpp_namespace.h.

1.2.3.107 Geometry optix::GeometryInstanceObj::getGeometry () [inline, inherited]

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

Definition at line 2505 of file optixpp_namespace.h.

1.2.3.108 unsigned int optix::BufferObj::getGLBOId () [inline, inherited]

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

Definition at line 3017 of file optixpp_namespace.h.

1.2.3.109 int optix::ContextObj::getGPUPagingActive () [inline, inherited]

See rtContextGetAttribute.

Definition at line 1848 of file optixpp_namespace.h.

1.2.3.110 int optix::ContextObj::getGPUPagingForcedOff () [inline, inherited]

See rtContextGetAttribute.

Definition at line 1855 of file optixpp_namespace.h.

1.2.3.111 RTtextureindexmode optix::TextureSamplerObj::getIndexingMode () [inline, inherited]

Query texture indexing mode for this sampler. See rtTextureSamplerGetIndexingMode.

Definition at line 2860 of file optixpp_namespace.h.

1.2.3.112 int optix::VariableObj::getInt () [inline, inherited]

Definition at line 3324 of file optixpp_namespace.h.

1.2.3.113 Program optix::GeometryObj::getIntersectionProgram () [inline, inherited]

Get the intersection program for this geometry. See rtGeometryGetIntersectionProgram.

Definition at line 2630 of file optixpp_namespace.h.

1.2.3.114 Material optix::GeometryInstanceObj::getMaterial (unsigned int *idx*) [inline, inherited]

Get the material at given index. See rtGeometryInstanceGetMaterial.

Definition at line 2529 of file optixpp_namespace.h.

1.2.3.115 unsigned int optix::GeometryInstanceObj::getMaterialCount () [inline, inherited]

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

Definition at line 2517 of file optixpp_namespace.h.

1.2.3.116 void optix::TransformObj::getMatrix (bool *transpose*, float * *matrix*, float * *inverse_matrix*) [inline, inherited]

Get the transform matrix for this node. See rtTransformGetMatrix.

Definition at line 2384 of file optixpp_namespace.h.

1.2.3.117 float optix::TextureSamplerObj::getMaxAnisotropy () [inline, inherited]

Query maximum anisotropy for this sampler. See rtTextureSamplerGetMaxAnisotropy.

Definition at line 2836 of file optixpp_namespace.h.

1.2.3.118 int optix::ContextObj::getMaxTextureCount () [inline, inherited]

See rtContextGetAttribute

Definition at line 1827 of file optixpp_namespace.h.

1.2.3.119 unsigned int optix::TextureSamplerObj::getMipLevelCount () [inline, inherited]

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

Definition at line 2790 of file optixpp_namespace.h.

1.2.3.120 Program optix::ContextObj::getMissProgram (unsigned int *ray_type_index*) [inline, inherited]

See rtContextGetMissProgram.

Definition at line 1967 of file optixpp_namespace.h.

1.2.3.121 std::string optix::VariableObj::getName () [inline, inherited]

Retrieve the name of the variable.

Definition at line 3389 of file optixpp_namespace.h.

1.2.3.122 unsigned int optix::GeometryObj::getPrimitiveCount () [inline, inherited]

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

Definition at line 2606 of file optixpp_namespace.h.

1.2.3.123 RTsize optix::ContextObj::getPrintBufferSize () [inline, inherited]

See rtContextGetPrintBufferSize.

Definition at line 2019 of file optixpp_namespace.h.

1.2.3.124 bool optix::ContextObj::getPrintEnabled () [inline, inherited]

See rtContextGetPrintEnabled.

Definition at line 2007 of file optixpp_namespace.h.

1.2.3.125 optix::int3 optix::ContextObj::getPrintLaunchIndex () [inline, inherited]

See rtContextGetPrintLaunchIndex.

Definition at line 2031 of file optixpp_namespace.h.

1.2.3.126 std::string optix::AccelerationObj::getProperty (const std::string & *name*) [inline, inherited]

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

Definition at line 2429 of file optixpp_namespace.h.

1.2.3.127 Program optix::ContextObj::getRayGenerationProgram (unsigned int *entry_point_index*) [inline, inherited]

See rtContextGetRayGenerationProgram.

Definition at line 1916 of file optixpp_namespace.h.

1.2.3.128 unsigned int optix::ContextObj::getRayTypeCount () [inline, inherited]

See rtContextGetRayTypeCount.

Definition at line 1955 of file optixpp_namespace.h.

1.2.3.129 RTtexturereadmode optix::TextureSamplerObj::getReadMode () [inline, inherited]

Query texture read mode for this sampler. See rtTextureSamplerGetReadMode.

Definition at line 2848 of file optixpp_namespace.h.

1.2.3.130 int optix::ContextObj::getRunningState () [inline, inherited]

See rtContextGetRunningState.

Definition at line 1995 of file optixpp_namespace.h.

1.2.3.131 RTsize optix::VariableObj::getSize () [inline, inherited]

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

Definition at line 3415 of file optixpp_namespace.h.

1.2.3.132 void optix::BufferObj::getSize (unsigned int *dimensionality*, RTsize ∗ *dims*) [inline, inherited]

Query dimensions of buffer. See rtBufferGetSizev.

Definition at line 3005 of file optixpp_namespace.h.

1.2.3.133 void optix::BufferObj::getSize (RTsize & *width*, RTsize & *height*, RTsize & *depth*) [inline, inherited]

Query 3D buffer dimension. See rtBufferGetSize3D.

Definition at line 2995 of file optixpp_namespace.h.

1.2.3.134 void optix::BufferObj::getSize (RTsize & *width*, RTsize & *height*) [inline, inherited]

Query 2D buffer dimension. See rtBufferGetSize2D.

Definition at line 2985 of file optixpp_namespace.h.

1.2.3.135 void optix::BufferObj::getSize (RTsize & *width*) [inline, inherited]

Query 1D buffer dimension. See rtBufferGetSize1D.

Definition at line 2975 of file optixpp_namespace.h.

1.2.3.136 RTsize optix::ContextObj::getStackSize () [inline, inherited]

See rtContextGetStackSize.

Definition at line 1886 of file optixpp_namespace.h.

1.2.3.137 optix::TextureSampler optix::VariableObj::getTextureSampler () [inline, inherited]

Definition at line 3422 of file optixpp_namespace.h.

1.2.3.138 std::string optix::AccelerationObj::getTraverser () [inline, inherited]

Query the acceleration structure traverser. See rtAccelerationGetTraverser.

Definition at line 2453 of file optixpp_namespace.h.

1.2.3.139 RTobjecttype optix::VariableObj::getType () [inline, inherited]

Query the object type of the variable.

Definition at line 3403 of file optixpp_namespace.h.

1.2.3.140 unsigned int optix::VariableObj::getUint () [inline, inherited]

Definition at line 3317 of file optixpp_namespace.h.

1.2.3.141 RTsize optix::ContextObj::getUsedHostMemory () [inline, inherited]

See rtContextGetAttribute.

Definition at line 1841 of file optixpp_namespace.h.

1.2.3.142 void optix::VariableObj::getUserData (RTsize *size*, void * *ptr*) [inline, inherited]

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

Definition at line 3346 of file optixpp_namespace.h.

1.2.3.143 Variable optix::MaterialObj::getVariable (unsigned int *index*) [inline, virtual, inherited]

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

Implements optix::ScopedObj.

Definition at line 2755 of file optixpp_namespace.h.

1.2.3.144 Variable optix::GeometryObj::getVariable (unsigned int *index*) [inline, virtual, inherited]

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

Implements optix::ScopedObj.

Definition at line 2663 of file optixpp_namespace.h.

1.2.3.145 Variable optix::GeometryInstanceObj::getVariable (unsigned int *index*) [inline, virtual, inherited]

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

Implements optix::ScopedObj.

Definition at line 2571 of file optixpp_namespace.h.

1.2.3.146 Variable optix::SelectorObj::getVariable (unsigned int *index*) [inline, inherited]

Definition at line 2233 of file optixpp_namespace.h.

1.2.3.147 Variable optix::ProgramObj::getVariable (unsigned int *index*) [inline, virtual, inherited]

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

Implements optix::ScopedObj.

Definition at line 2121 of file optixpp_namespace.h.

1.2.3.148 Variable optix::ContextObj::getVariable (unsigned int *index*) [inline, virtual, inherited]

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

Implements optix::ScopedObj.

Definition at line 2064 of file optixpp_namespace.h.

1.2.3.149 unsigned int optix::MaterialObj::getVariableCount () [inline, virtual, inherited]

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 2748 of file optixpp_namespace.h.

1.2.3.150 unsigned int optix::GeometryObj::getVariableCount () [inline, virtual, inherited]

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 2656 of file optixpp_namespace.h.

1.2.3.151 unsigned int optix::GeometryInstanceObj::getVariableCount () [inline, virtual, inherited]

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 2564 of file optixpp_namespace.h.

1.2.3.152 unsigned int optix::SelectorObj::getVariableCount () [inline, inherited]

Definition at line 2226 of file optixpp_namespace.h.

1.2.3.153 unsigned int optix::ProgramObj::getVariableCount () [inline, virtual, inherited]

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 2114 of file optixpp_namespace.h.

1.2.3.154 unsigned int optix::ContextObj::getVariableCount () [inline, virtual, inherited]

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 2057 of file optixpp_namespace.h.

1.2.3.155 Program optix::SelectorObj::getVisitProgram () [inline, inherited]

Get the visitor program for this selector. See rtSelectorGetVisitProgram.

Definition at line 2174 of file optixpp_namespace.h.

1.2.3.156 RTwrapmode optix::TextureSamplerObj::getWrapMode (unsigned int *dim*) [inline, inherited]

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

Definition at line 2814 of file optixpp_namespace.h.

1.2.3.157 bool optix::GeometryObj::isDirty () [inline, inherited]

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

Definition at line 2675 of file optixpp_namespace.h.

1.2.3.158 bool optix::AccelerationObj::isDirty () [inline, inherited]

Query if the acceleration needs a rebuild. See rtAccelerationIsDirty.

Definition at line 2417 of file optixpp_namespace.h.

1.2.3.159 void optix::ContextObj::launch (unsigned int *entry_point_index*, RTsize *image_width*, RTsize *image_height*, RTsize *image_depth*) [inline, inherited]

See rtContextLaunch3D.

Definition at line 1989 of file optixpp_namespace.h.

1.2.3.160 void optix::ContextObj::launch (unsigned int *entry_point_index*, RTsize *image_width*, RTsize *image_height*) [inline, inherited]

See rtContextLaunch2D.

Definition at line 1984 of file optixpp_namespace.h.

1.2.3.161 void optix::ContextObj::launch (unsigned int *entry_point_index*, RTsize *image_width*) [inline, inherited]

See rtContextLaunch1D

Definition at line 1979 of file optixpp_namespace.h.

1.2.3.162 Exception optix::APIObj::makeException (RTresult *code*, RTcontext *context*) [inline, static, inherited]

For backwards compatability. Use Exception::makeException instead.

Definition at line 313 of file optixpp_namespace.h.

1.2.3.163 Exception optix::Exception::makeException (RTresult *code*, RTcontext *context*) [inline, static, inherited]

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

Definition at line 259 of file optixpp_namespace.h.

1.2.3.164 void * optix::BufferObj::map () [inline, inherited]

Maps a buffer object for host access. See rtBufferMap.

Definition at line 3089 of file optixpp_namespace.h.

1.2.3.165 void optix::GeometryObj::markDirty () [inline, inherited]

Mark this geometry as dirty, causing rebuild of parent groups acceleration. See rtGeometryMarkDirty.

Definition at line 2670 of file optixpp_namespace.h.

1.2.3.166 void optix::AccelerationObj::markDirty () [inline, inherited]

Mark the acceleration as needing a rebuild. See rtAccelerationMarkDirty.

Definition at line 2412 of file optixpp_namespace.h.

1.2.3.167 template<class T > Handle< VariableObj > optix::Handle< T >::operator[ ] (const char * *varname*) [inline, inherited]

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

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

or

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

Definition at line 598 of file optixpp_namespace.h.

1.2.3.168 template<class T > Handle< VariableObj > optix::Handle< T >::operator[ ] (const std::string & *varname*) [inline, inherited]

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 589 of file optixpp_namespace.h.

1.2.3.169 Variable optix::MaterialObj::queryVariable (const std::string & *name*) [inline, virtual, inherited]

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 2736 of file optixpp_namespace.h.

1.2.3.170 Variable optix::GeometryObj::queryVariable (const std::string & *name*) [inline, virtual, inherited]

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 2644 of file optixpp_namespace.h.

1.2.3.171 Variable optix::GeometryInstanceObj::queryVariable (const std::string & *name*) [inline, virtual, inherited]

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 2552 of file optixpp_namespace.h.

1.2.3.172 Variable optix::SelectorObj::queryVariable (const std::string & *name*) [inline, inherited]

Definition at line 2214 of file optixpp_namespace.h.

1.2.3.173 Variable optix::ProgramObj::queryVariable (const std::string & *name*) [inline, virtual, inherited]

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 2102 of file optixpp_namespace.h.

1.2.3.174 Variable optix::ContextObj::queryVariable (const std::string & *name*) [inline, virtual, inherited]

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 2045 of file optixpp_namespace.h.

1.2.3.175 void optix::BufferObj::registerGLBuffer () [inline, inherited]

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

Definition at line 3024 of file optixpp_namespace.h.

1.2.3.176 void optix::TextureSamplerObj::registerGLTexture () [inline, inherited]

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

Definition at line 2884 of file optixpp_namespace.h.

1.2.3.177 void optix::MaterialObj::removeVariable (Variable *v*) [inline, virtual, inherited]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2743 of file optixpp_namespace.h.

1.2.3.178 void optix::GeometryObj::removeVariable (Variable *v*) [inline, virtual, inherited]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2651 of file optixpp_namespace.h.

1.2.3.179 void optix::GeometryInstanceObj::removeVariable (Variable *v*) [inline, virtual, inherited]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2559 of file optixpp_namespace.h.

1.2.3.180 void optix::SelectorObj::removeVariable (Variable *v*) [inline, inherited]

Definition at line 2221 of file optixpp_namespace.h.

1.2.3.181 void optix::ProgramObj::removeVariable (Variable *v*) [inline, virtual, inherited]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2109 of file optixpp_namespace.h.

1.2.3.182 void optix::ContextObj::removeVariable (Variable *v*) [inline, virtual, inherited]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2052 of file optixpp_namespace.h.

1.2.3.183 void optix::VariableObj::set (Buffer *buffer*) [inline, inherited]

Definition at line 3336 of file optixpp_namespace.h.

1.2.3.184 void optix::VariableObj::set1fv (const float * *f*) [inline, inherited]

Set variable value to a scalar float.

Definition at line 3234 of file optixpp_namespace.h.

1.2.3.185 void optix::VariableObj::set1iv (const int * *i*) [inline, inherited]

Definition at line 3290 of file optixpp_namespace.h.

1.2.3.186 void optix::VariableObj::set1uiv (const unsigned int * *u*) [inline, inherited]

Definition at line 3134 of file optixpp_namespace.h.

1.2.3.187 void optix::VariableObj::set2fv (const float * f) [inline, inherited]

Set variable value to a float2.

Definition at line 3239 of file optixpp_namespace.h.

1.2.3.188 void optix::VariableObj::set2iv (const int * i) [inline, inherited]

Definition at line 3295 of file optixpp_namespace.h.

1.2.3.189 void optix::VariableObj::set2uiv (const unsigned int * u) [inline, inherited]

Definition at line 3139 of file optixpp_namespace.h.

1.2.3.190 void optix::VariableObj::set3fv (const float * f) [inline, inherited]

Set variable value to a float3.

Definition at line 3244 of file optixpp_namespace.h.

1.2.3.191 void optix::VariableObj::set3iv (const int * i) [inline, inherited]

Definition at line 3300 of file optixpp_namespace.h.

1.2.3.192 void optix::VariableObj::set3uiv (const unsigned int * u) [inline, inherited]

Definition at line 3144 of file optixpp_namespace.h.

1.2.3.193 void optix::VariableObj::set4fv (const float * f) [inline, inherited]

Set variable value to a float4.

Definition at line 3249 of file optixpp_namespace.h.

1.2.3.194 void optix::VariableObj::set4iv (const int * i) [inline, inherited]

Definition at line 3305 of file optixpp_namespace.h.

1.2.3.195 void optix::VariableObj::set4uiv (const unsigned int * *u*) [inline, inherited]

Definition at line 3149 of file optixpp_namespace.h.

1.2.3.196 void optix::GeometryGroupObj::setAcceleration (Acceleration *acceleration*) [inline, inherited]

Set the Acceleration structure for this group. See rtGeometryGroupSetAcceleration.

Definition at line 2306 of file optixpp_namespace.h.

1.2.3.197 void optix::GroupObj::setAcceleration (Acceleration *acceleration*) [inline, inherited]

Set the Acceleration structure for this group. See rtGroupSetAcceleration.

Definition at line 2245 of file optixpp_namespace.h.

1.2.3.198 void optix::MaterialObj::setAnyHitProgram (unsigned int *ray_type_index*, Program *program*) [inline, inherited]

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

Definition at line 2717 of file optixpp_namespace.h.

1.2.3.199 void optix::TextureSamplerObj::setArraySize (unsigned int *num_textures_in_array*) [inline, inherited]

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

Definition at line 2797 of file optixpp_namespace.h.

1.2.3.200 void optix::GeometryObj::setBoundingBoxProgram (Program *program*) [inline, inherited]

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

Definition at line 2613 of file optixpp_namespace.h.

1.2.3.201 void optix::VariableObj::setBuffer (Buffer *buffer*) [inline, inherited]

Definition at line 3331 of file optixpp_namespace.h.

1.2.3.202 void optix::TextureSamplerObj::setBuffer (unsigned int *texture_array_idx*, unsigned int *mip_level*, Buffer *buffer*) [inline, inherited]

Set the underlying buffer used for texture storage. rtTextureSamplerSetBuffer.

Definition at line 2867 of file optixpp_namespace.h.

1.2.3.203 void optix::AccelerationObj::setBuilder (const std::string & *builder*) [inline, inherited]

Specify the acceleration structure builder. See rtAccelerationSetBuilder.

Definition at line 2436 of file optixpp_namespace.h.

1.2.3.204 template<typename T > void optix::TransformObj::setChild (T *child*) [inline, inherited]

Set the child node of this transform. See rtTransformSetChild.

Definition at line 2366 of file optixpp_namespace.h.

1.2.3.205 void optix::GeometryGroupObj::setChild (unsigned int *index*, GeometryInstance *geometryinstance*) [inline, inherited]

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

Definition at line 2330 of file optixpp_namespace.h.

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

Set an indexed child within this group. See rtGroupSetChild.

Definition at line 2270 of file optixpp_namespace.h.

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

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

Definition at line 2194 of file optixpp_namespace.h.

1.2.3.208 void optix::GeometryGroupObj::setChildCount (unsigned int *count*) [inline, inherited]

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

Definition at line 2318 of file optixpp_namespace.h.

1.2.3.209 void optix::GroupObj::setChildCount (unsigned int *count*) [inline, inherited]

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

Definition at line 2257 of file optixpp_namespace.h.

1.2.3.210 void optix::SelectorObj::setChildCount (unsigned int *count*) [inline, inherited]

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

Definition at line 2181 of file optixpp_namespace.h.

1.2.3.211 void optix::MaterialObj::setClosestHitProgram (unsigned int *ray_type_index*, Program *program*) [inline, inherited]

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

Definition at line 2705 of file optixpp_namespace.h.

1.2.3.212 void optix::ContextObj::setCPUNumThreads (int *cpu_num_threads*) [inline, inherited]

See rtContextSetAttribute

Definition at line 1871 of file optixpp_namespace.h.

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

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

Definition at line 2472 of file optixpp_namespace.h.

1.2.3.214 template<class Iterator > void optix::ContextObj::setDevices (Iterator *begin*, Iterator *end*) [inline, inherited]

See rtContextSetDevices

Definition at line 1805 of file optixpp_namespace.h.

1.2.3.215 void optix::BufferObj::setElementSize (RTsize *size_of_element*) [inline, inherited]

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

Definition at line 2958 of file optixpp_namespace.h.

1.2.3.216 void optix::ContextObj::setEntryPointCount (unsigned int *num_entry_points*) [inline, inherited]

See rtContextSetEntryPointCount.

Definition at line 1898 of file optixpp_namespace.h.

1.2.3.217 void optix::ContextObj::setExceptionEnabled (RTexception *exception*, bool *enabled*) [inline, inherited]

See rtContextSetExceptionEnabled.

Definition at line 1937 of file optixpp_namespace.h.

1.2.3.218 void optix::ContextObj::setExceptionProgram (unsigned int *entry_point_index*, Program *program*) [inline, inherited]

See rtContextSetExceptionProgram.

Definition at line 1924 of file optixpp_namespace.h.

1.2.3.219 void optix::TextureSamplerObj::setFilteringModes (RTfiltermode *minification*, RTfiltermode *magnification*, RTfiltermode *mipmapping*) [inline, inherited]

Set filtering modes for this sampler. See rtTextureSamplerSetFilteringModes.

Definition at line 2821 of file optixpp_namespace.h.

1.2.3.220 void optix::VariableObj::setFloat (float *f1*, float *f2*, float *f3*, float *f4*) [inline, inherited]

Set variable value to a float4.

Definition at line 3229 of file optixpp_namespace.h.

1.2.3.221 void optix::VariableObj::setFloat (optix::float4 *f*) [inline, inherited]

Set variable value to a float4.

Definition at line 3224 of file optixpp_namespace.h.

1.2.3.222 void optix::VariableObj::setFloat (float *f1*, float *f2*, float *f3*) [inline, inherited]

Set variable value to a float3.

Definition at line 3219 of file optixpp_namespace.h.

1.2.3.223 void optix::VariableObj::setFloat (optix::float3 *f*) [inline, inherited]

Set variable value to a float3.

Definition at line 3214 of file optixpp_namespace.h.

1.2.3.224 void optix::VariableObj::setFloat (float *f1*, float *f2*) [inline, inherited]

Set variable value to a float2.

Definition at line 3209 of file optixpp_namespace.h.

1.2.3.225 void optix::VariableObj::setFloat (optix::float2 *f*) [inline, inherited]

Set variable value to a float2.

Definition at line 3204 of file optixpp_namespace.h.

1.2.3.226 void optix::VariableObj::setFloat (float *f1*) [inline, inherited]

Set variable value to a scalar float.

Definition at line 3199 of file optixpp_namespace.h.

1.2.3.227 void optix::BufferObj::setFormat (RTformat *format*) [inline, inherited]

Set the data format for the buffer. See rtBufferSetFormat.

Definition at line 2946 of file optixpp_namespace.h.

1.2.3.228 void optix::GeometryInstanceObj::setGeometry (Geometry *geometry*) [inline, inherited]

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

Definition at line 2500 of file optixpp_namespace.h.

1.2.3.229 void optix::ContextObj::setGPUPagingForcedOff (int *gpu_paging_forced_off*) [inline, inherited]

See rtContextSetAttribute.

Definition at line 1876 of file optixpp_namespace.h.

1.2.3.230 void optix::TextureSamplerObj::setIndexingMode (RTtextureindexmode *indexmode*) [inline, inherited]

Set texture indexing mode for this sampler. See rtTextureSamplerSetIndexingMode.

Definition at line 2855 of file optixpp_namespace.h.

1.2.3.231 void optix::VariableObj::setInt (int *i1*, int *i2*, int *i3*, int *i4*) [inline, inherited]

Definition at line 3285 of file optixpp_namespace.h.

1.2.3.232 void optix::VariableObj::setInt (optix::int4 *i*) [inline, inherited]

Definition at line 3280 of file optixpp_namespace.h.

1.2.3.233 void optix::VariableObj::setInt (int *i1*, int *i2*, int *i3*) [inline, inherited]

Definition at line 3275 of file optixpp_namespace.h.

1.2.3.234 void optix::VariableObj::setInt (optix::int3 *i*) [inline, inherited]

Definition at line 3270 of file optixpp_namespace.h.

1.2.3.235 void optix::VariableObj::setInt (int *i1*, int *i2*) [inline, inherited]

Definition at line 3265 of file optixpp_namespace.h.

1.2.3.236 void optix::VariableObj::setInt (optix::int2 *i*) [inline, inherited]

Definition at line 3260 of file optixpp_namespace.h.

1.2.3.237 void optix::VariableObj::setInt (int *i1*) [inline, inherited]

Definition at line 3255 of file optixpp_namespace.h.

1.2.3.238 void optix::GeometryObj::setIntersectionProgram (Program *program*) [inline, inherited]

Set the intersection program for this geometry. See rtGeometrySetIntersectionProgram.

Definition at line 2625 of file optixpp_namespace.h.

1.2.3.239 void optix::GeometryInstanceObj::setMaterial (unsigned int *idx*, Material *material*) [inline, inherited]

Set the material at given index. See rtGeometryInstanceSetMaterial.

Definition at line 2524 of file optixpp_namespace.h.

1.2.3.240 void optix::GeometryInstanceObj::setMaterialCount (unsigned int *count*) [inline, inherited]

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

Definition at line 2512 of file optixpp_namespace.h.

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

Set the transform matrix for this node. See rtTransformSetMatrix.

Definition at line 2379 of file optixpp_namespace.h.

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

Definition at line 3154 of file optixpp_namespace.h.

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

Definition at line 3159 of file optixpp_namespace.h.

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

Definition at line 3164 of file optixpp_namespace.h.

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

Definition at line 3169 of file optixpp_namespace.h.

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

Definition at line 3174 of file optixpp_namespace.h.

1.2.3.247 void optix::VariableObj::setMatrix3x4fv (bool *transpose*, const float * *m*) [inline, inherited]

Definition at line 3179 of file optixpp_namespace.h.

1.2.3.248 void optix::VariableObj::setMatrix4x2fv (bool *transpose*, const float * *m*) [inline, inherited]

Definition at line 3184 of file optixpp_namespace.h.

1.2.3.249 void optix::VariableObj::setMatrix4x3fv (bool *transpose*, const float * *m*) [inline, inherited]

Definition at line 3189 of file optixpp_namespace.h.

1.2.3.250 void optix::VariableObj::setMatrix4x4fv (bool *transpose*, const float * *m*) [inline, inherited]

Definition at line 3194 of file optixpp_namespace.h.

1.2.3.251 void optix::TextureSamplerObj::setMaxAnisotropy (float *value*) [inline, inherited]

Set maximum anisotropy for this sampler. See rtTextureSamplerSetMaxAnisotropy.

Definition at line 2831 of file optixpp_namespace.h.

1.2.3.252 void optix::TextureSamplerObj::setMipLevelCount (unsigned int *num_mip_levels*) [inline, inherited]

Set the number of mip levels for this sampler. See rtTextureSamplerSetMipLevelCount.

Definition at line 2785 of file optixpp_namespace.h.

1.2.3.253 void optix::ContextObj::setMissProgram (unsigned int *ray_type_index*, Program *program*) [inline, inherited]

See rtContextSetMissProgram.

Definition at line 1962 of file optixpp_namespace.h.

1.2.3.254 void optix::GeometryObj::setPrimitiveCount (unsigned int *num_primitives*) [inline, inherited]

Set the number of primitives in this geometry objects (eg, number of triangles in mesh). See rtGeometrySetPrimitiveCount

Definition at line 2601 of file optixpp_namespace.h.

1.2.3.255 void optix::ContextObj::setPrintBufferSize (RTsize *buffer_size_bytes*) [inline, inherited]

See rtContextSetPrintBufferSize.

Definition at line 2014 of file optixpp_namespace.h.

1.2.3.256 void optix::ContextObj::setPrintEnabled (bool *enabled*) [inline, inherited]

See rtContextSetPrintEnabled

Definition at line 2002 of file optixpp_namespace.h.

1.2.3.257 void optix::ContextObj::setPrintLaunchIndex (int *x*, int *y* = -1, int *z* = -1) [inline, inherited]

See rtContextSetPrintLaunchIndex.

Definition at line 2026 of file optixpp_namespace.h.

1.2.3.258 void optix::AccelerationObj::setProperty (const std::string & *name*, const std::string & *value*) [inline, inherited]

Set properties specifying Acceleration builder/traverser behavior. See rtAccelerationSetProperty.

Definition at line 2424 of file optixpp_namespace.h.

1.2.3.259 void optix::ContextObj::setRayGenerationProgram (unsigned int *entry_point_index*, Program *program*) [inline, inherited]

See rtContextSetRayGenerationProgram

Definition at line 1911 of file optixpp_namespace.h.

1.2.3.260 void optix::ContextObj::setRayTypeCount (unsigned int *num_ray_types*) [inline, inherited]

See rtContextSetRayTypeCount.

Definition at line 1950 of file optixpp_namespace.h.

1.2.3.261 void optix::TextureSamplerObj::setReadMode (RTtexturereadmode *readmode*) [inline, inherited]

Set texture read mode for this sampler. See rtTextureSamplerSetReadMode.

Definition at line 2843 of file optixpp_namespace.h.

1.2.3.262 void optix::BufferObj::setSize (unsigned int *dimensionality*, const RTsize * *dims*) [inline, inherited]

Set buffer dimensionality and dimensions to specified values. See rtBufferSetSizev.

Definition at line 3000 of file optixpp_namespace.h.

1.2.3.263 void optix::BufferObj::setSize (RTsize *width*, RTsize *height*, RTsize *depth*) [inline, inherited]

Set buffer dimensionality to three and buffer dimensions to specified width,height,depth. See rtBufferSetSize3D.

Definition at line 2990 of file optixpp_namespace.h.

1.2.3.264 void optix::BufferObj::setSize (RTsize *width*, RTsize *height*) [inline, inherited]

Set buffer dimensionality to two and buffer dimensions to specified width,height. See rtBufferSetSize2D.

Definition at line 2980 of file optixpp_namespace.h.

1.2.3.265 void optix::BufferObj::setSize (RTsize *width*) [inline, inherited]

Set buffer dimensionality to one and buffer width to specified width. See rtBufferSetSize1D.

Definition at line 2970 of file optixpp_namespace.h.

1.2.3.266 void optix::ContextObj::setStackSize (RTsize *stack_size_bytes*) [inline, inherited]

See rtContextSetStackSize

Definition at line 1881 of file optixpp_namespace.h.

1.2.3.267 void optix::VariableObj::setTextureSampler (TextureSampler *texturesample*) [inline, inherited]

Definition at line 3351 of file optixpp_namespace.h.

1.2.3.268 void optix::ContextObj::setTimeoutCallback (RTtimeoutcallback *callback*, double *min_polling_seconds*) [inline, inherited]

See rtContextSetTimeoutCallback RTtimeoutcallback is defined as typedef int (∗RTtimeoutcallback)(void).

Definition at line 1893 of file optixpp_namespace.h.

1.2.3.269 void optix::AccelerationObj::setTraverser (const std::string & *traverser*) [inline, inherited]

Specify the acceleration structure traverser. See rtAccelerationSetTraverser.

Definition at line 2448 of file optixpp_namespace.h.

1.2.3.270 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*, unsigned int *u3*, unsigned int *u4*) [inline, inherited]

Definition at line 3129 of file optixpp_namespace.h.

1.2.3.271 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*, unsigned int *u3*) [inline, inherited]

Definition at line 3124 of file optixpp_namespace.h.

1.2.3.272 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*) [inline, inherited]

Definition at line 3119 of file optixpp_namespace.h.

1.2.3.273 void optix::VariableObj::setUint (unsigned int *u1*) [inline, inherited]

Definition at line 3114 of file optixpp_namespace.h.

1.2.3.274 void optix::VariableObj::setUserData (RTsize *size*, const void * *ptr*) [inline, inherited]

Set the variable to a user defined type given the sizeof the user object.

Definition at line 3341 of file optixpp_namespace.h.

1.2.3.275 void optix::SelectorObj::setVisitProgram (Program *program*) [inline, inherited]

Set the visitor program for this selector. See rtSelectorSetVisitProgram

Definition at line 2169 of file optixpp_namespace.h.

1.2.3.276 void optix::TextureSamplerObj::setWrapMode (unsigned int *dim*, RTwrapmode *wrapmode*) [inline, inherited]

Set the texture wrap mode for this sampler. See rtTextureSamplerSetWrapMode.

Definition at line 2809 of file optixpp_namespace.h.

1.2.3.277 void optix::BufferObj::unmap () [inline, inherited]

Unmaps a buffer object. See rtBufferUnmap.

Definition at line 3096 of file optixpp_namespace.h.

1.2.3.278 void optix::BufferObj::unregisterGLBuffer () [inline, inherited]

Unregister the buffer, re-enabling OptiX operations. See rtTextureSamplerGLUnregister.

Definition at line 3029 of file optixpp_namespace.h.

1.2.3.279 void optix::TextureSamplerObj::unregisterGLTexture () [inline, inherited]

Unregister the texture's buffer, re-enabling OptiX operations. See rtTextureSamplerGLUnregister.

Definition at line 2889 of file optixpp_namespace.h.

1.2.3.280 void optix::BufferObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2934 of file optixpp_namespace.h.

1.2.3.281 void optix::TextureSamplerObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2773 of file optixpp_namespace.h.

1.2.3.282 void optix::MaterialObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2693 of file optixpp_namespace.h.

1.2.3.283 void optix::GeometryObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2589 of file optixpp_namespace.h.

1.2.3.284 void optix::GeometryInstanceObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2488 of file optixpp_namespace.h.

1.2.3.285 void optix::AccelerationObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2400 of file optixpp_namespace.h.

1.2.3.286 void optix::TransformObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2353 of file optixpp_namespace.h.

1.2.3.287 void optix::GeometryGroupObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2294 of file optixpp_namespace.h.

1.2.3.288 void optix::SelectorObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2157 of file optixpp_namespace.h.

1.2.3.289 void optix::GroupObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2139 of file optixpp_namespace.h.

1.2.3.290 void optix::ProgramObj::validate () [inline, virtual, inherited]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2083 of file optixpp_namespace.h.

1.2.3.291 void optix::ContextObj::validate () [inline, virtual, inherited]

See rtContextValidate.

Implements optix::DestroyableObj.

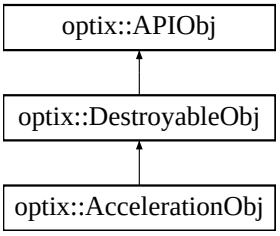
Definition at line 1552 of file optixpp_namespace.h.

2 Class Documentation

2.1 optix::AccelerationObj Class Reference

Acceleration wraps the OptiX C API RTacceleration opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::AccelerationObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTacceleration get ()

Friends

- class Handle< AccelerationObj >

- void markDirty ()
- bool isDirty ()
- void setProperty (const std::string &name, const std::string &value)
- std::string getProperty (const std::string &name)
- void setBuilder (const std::string &builder)
- std::string getBuilder ()
- void setTraverser (const std::string &traverser)
- std::string getTraverser ()
- RTsize getDataSize ()
- void getData (void *data)
- 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 1078 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 2394 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 2477 of file optixpp_namespace.h.

2.1.2.3 std::string optix::AccelerationObj::getBuilder () [inline]

Query the acceleration structure builder. See rtAccelerationGetBuilder.

Definition at line 2441 of file optixpp_namespace.h.

2.1.2.4 Context optix::AccelerationObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2405 of file optixpp_namespace.h.

2.1.2.5 void optix::AccelerationObj::getData (void * *data*) [inline]

Get the marshalled acceleration data. See rtAccelerationGetData.

Definition at line 2467 of file optixpp_namespace.h.

2.1.2.6 RTsize optix::AccelerationObj::getDataSize () [inline]

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

Definition at line 2460 of file optixpp_namespace.h.

2.1.2.7 std::string optix::AccelerationObj::getProperty (const std::string & *name*) [inline]

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

Definition at line 2429 of file optixpp_namespace.h.

2.1.2.8 std::string optix::AccelerationObj::getTraverser () [inline]

Query the acceleration structure traverser. See rtAccelerationGetTraverser.

Definition at line 2453 of file optixpp_namespace.h.

2.1.2.9 bool optix::AccelerationObj::isDirty () [inline]

Query if the acceleration needs a rebuild. See rtAccelerationIsDirty.

Definition at line 2417 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 2412 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 2436 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 2472 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 2424 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 2448 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 2400 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 1126 of file optixpp_namespace.h.

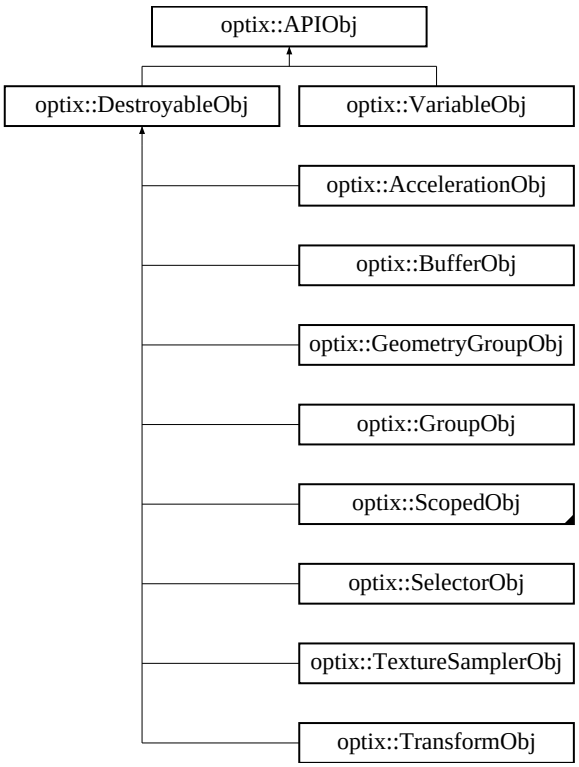
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.2 optix::APIObj Class Reference

Base class for all reference counted wrappers around OptiX C API opaque types.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::APIObj::



Public Member Functions

- APIObj ()
- virtual ~APIObj ()
- void addReference ()
- int removeReference ()
- virtual Context getContext ()=0
- virtual void checkError (RTresult code)
- void checkErrorNoGetContext (RTresult code)

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 288 of file optixpp_namespace.h.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 optix::APIObj::APIObj () [inline]

Definition at line 290 of file optixpp_namespace.h.

2.2.2.2 virtual optix::APIObj::~APIObj () [inline, virtual]

Definition at line 291 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 294 of file optixpp_namespace.h.

2.2.3.2 void optix::APIObj::checkError (RTresult *code*) [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 1486 of file optixpp_namespace.h.

2.2.3.3 void optix::APIObj::checkErrorNoGetContext (RTresult *code*) [inline]

Definition at line 1494 of file optixpp_namespace.h.

2.2.3.4 virtual Context optix::APIObj::getContext () [pure virtual]

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

Implemented in optix::VariableObj, optix::ContextObj, optix::ProgramObj, optix::GroupObj, optix::GeometryGroupObj, optix::TransformObj, optix::SelectorObj, optix::AccelerationObj, optix::GeometryInstanceObj, optix::GeometryObj, optix::MaterialObj, optix::TextureSamplerObj, and optix::BufferObj.

2.2.3.5 Exception optix::APIObj::makeException (RTresult *code*, RTcontext *context*) [inline, static]

For backwards compatability. Use Exception::makeException instead.

Definition at line 313 of file optixpp_namespace.h.

2.2.3.6 int optix::APIObj::removeReference () [inline]

Decrement the reference count for this object.

Definition at line 296 of file optixpp_namespace.h.

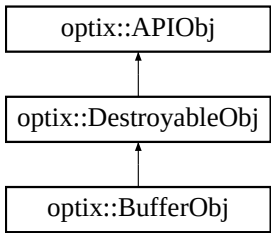
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.3 optix::BufferObj Class Reference

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

`#include <optixpp_namespace.h>`Inheritance diagram for optix::BufferObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTbuffer get ()

Friends

- class Handle< BufferObj >

- void setFormat (RTformat format)
- RTformat getFormat ()
- void setElementSize (RTsize size_of_element)
- RTsize getElementSize ()
- void setSize (RTsize width)
- void getSize (RTsize &width)
- void setSize (RTsize width, RTsize height)
- void getSize (RTsize &width, RTsize &height)
- void setSize (RTsize width, RTsize height, RTsize depth)
- void getSize (RTsize &width, RTsize &height, RTsize &depth)
- void setSize (unsigned int dimensionality, const RTsize *dims)
- void getSize (unsigned int dimensionality, RTsize *dims)
- unsigned int getDimensionality ()
- unsigned int getGLBOId ()
- 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 1387 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 2928 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 3102 of file optixpp_namespace.h.

2.3.2.3 Context optix::BufferObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2939 of file optixpp_namespace.h.

2.3.2.4 unsigned int optix::BufferObj::getDimensionality () [inline]

Query dimensionality of buffer. See rtBufferGetDimensionality.

Definition at line 3010 of file optixpp_namespace.h.

2.3.2.5 RTsize optix::BufferObj::getElementSize () [inline]

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

Definition at line 2963 of file optixpp_namespace.h.

2.3.2.6 RTformat optix::BufferObj::getFormat () [inline]

Query the data format for the buffer. See rtBufferGetFormat.

Definition at line 2951 of file optixpp_namespace.h.

2.3.2.7 unsigned int optix::BufferObj::getGLBOId () [inline]

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

Definition at line 3017 of file optixpp_namespace.h.

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

Query dimensions of buffer. See rtBufferGetSizev.

Definition at line 3005 of file optixpp_namespace.h.

2.3.2.9 void optix::BufferObj::getSize (RTsize & *width*, RTsize & *height*, RTsize & *depth*) [inline]

Query 3D buffer dimension. See rtBufferGetSize3D.

Definition at line 2995 of file optixpp_namespace.h.

2.3.2.10 void optix::BufferObj::getSize (RTsize & *width*, RTsize & *height*) [inline]

Query 2D buffer dimension. See rtBufferGetSize2D.

Definition at line 2985 of file optixpp_namespace.h.

2.3.2.11 void optix::BufferObj::getSize (RTsize & *width*) [inline]

Query 1D buffer dimension. See rtBufferGetSize1D.

Definition at line 2975 of file optixpp_namespace.h.

2.3.2.12 void * optix::BufferObj::map () [inline]

Maps a buffer object for host access. See rtBufferMap.

Definition at line 3089 of file optixpp_namespace.h.

2.3.2.13 void optix::BufferObj::registerGLBuffer () [inline]

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

Definition at line 3024 of file optixpp_namespace.h.

2.3.2.14 void optix::BufferObj::setElementSize (RTsize *size_of_element*) [inline]

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

Definition at line 2958 of file optixpp_namespace.h.

2.3.2.15 void optix::BufferObj::setFormat (RTformat *format*) [inline]

Set the data format for the buffer. See rtBufferSetFormat.

Definition at line 2946 of file optixpp_namespace.h.

2.3.2.16 void optix::BufferObj::setSize (unsigned int *dimensionality*, const RTsize * *dims*) [inline]

Set buffer dimensionality and dimensions to specified values. See rtBufferSetSizev.

Definition at line 3000 of file optixpp_namespace.h.

2.3.2.17 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 2990 of file optixpp_namespace.h.

2.3.2.18 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 2980 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 2970 of file optixpp_namespace.h.

2.3.2.20 void optix::BufferObj::unmap () [inline]

Unmaps a buffer object. See rtBufferUnmap.

Definition at line 3096 of file optixpp_namespace.h.

2.3.2.21 void optix::BufferObj::unregisterGLBuffer () [inline]

Unregister the buffer, re-enabling OptiX operations. See rtTextureSamplerGLUnregister.

Definition at line 3029 of file optixpp_namespace.h.

2.3.2.22 void optix::BufferObj::validate () [inline, virtual]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2934 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 1479 of file optixpp_namespace.h.

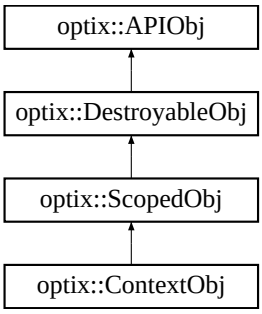
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.4 optix::ContextObj Class Reference

Context object wraps the OptiX C API RTcontext opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::ContextObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- void compile ()
- int getRunningState ()
- 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)
- std::string getErrorString (RTresult code)
- 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 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 ()
- unsigned int getEnabledDeviceCount ()
- int getMaxTextureCount ()
- int getCPUNumThreads ()
- RTsize getUsedHostMemory ()
- int getGPUPagingActive ()
- int getGPUPagingForcedOff ()
- RTsize getAvailableDeviceMemory (int ordinal)
- void setCPUNumThreads (int cpu_num_threads)
- void setGPUPagingForcedOff (int gpu_paging_forced_off)
- void setStackSize (RTsize stack_size_bytes)
- RTsize getStackSize ()
- void setTimeoutCallback (RTtimeoutcallback callback, double min_polling_seconds)
- void setEntryPointCount (unsigned int num_entry_points)
- unsigned int getEntryPointCount ()
- void setRayTypeCount (unsigned int num_ray_types)
- unsigned int getRayTypeCount ()

- void setRayGenerationProgram (unsigned int entry_point_index, Program program)
- Program getRayGenerationProgram (unsigned int entry_point_index)
- void setExceptionProgram (unsigned int entry_point_index, Program program)
- Program getExceptionProgram (unsigned int entry_point_index)
- void setExceptionEnabled (RTexception exception, bool enabled)
- bool getExceptionEnabled (RTexception exception)
- void setMissProgram (unsigned int ray_type_index, Program program)
- Program getMissProgram (unsigned int ray_type_index)
- 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 ()
- void setPrintBufferSize (RTsize buffer_size_bytes)
- RTsize getPrintBufferSize ()
- void setPrintLaunchIndex (int x, int y=-1, int z=-1)
- optix::int3 getPrintLaunchIndex ()
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)

2.4.1 Detailed Description

Context object wraps the OptiX C API RTcontext opaque type and its associated function set.

Definition at line 610 of file optixpp_namespace.h.

2.4.2 Member Function Documentation

2.4.2.1 void optix::ContextObj::checkError (RTresult *code*) [inline, virtual]

See APIObj::checkError

Reimplemented from optix::APIObj.

Definition at line 1506 of file optixpp_namespace.h.

2.4.2.2 void optix::ContextObj::compile () [inline]

See rtContextCompile.

Definition at line 1974 of file optixpp_namespace.h.

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

Creates a Context object. See rtContextCreate.

Definition at line 1537 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 1557 of file optixpp_namespace.h.

2.4.2.5 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 1600 of file optixpp_namespace.h.

2.4.2.6 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 1591 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 1582 of file optixpp_namespace.h.

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

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

Definition at line 1574 of file optixpp_namespace.h.

2.4.2.9 Buffer optix::ContextObj::createBuffer (unsigned int *type*) [inline]

Create a buffer with given RTbuffertype. See rtBufferCreate.

Definition at line 1567 of file optixpp_namespace.h.

2.4.2.10 Buffer optix::ContextObj::createBufferFromGLBO (unsigned int *type*, unsigned int *vbo*) [inline]

Create buffer from GL buffer object. See rtBufferCreateFromGLBO.

Definition at line 1609 of file optixpp_namespace.h.

2.4.2.11 Geometry optix::ContextObj::createGeometry () [inline]

See rtGeometryCreate.

Definition at line 1684 of file optixpp_namespace.h.

2.4.2.12 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 1742 of file optixpp_namespace.h.

2.4.2.13 GeometryGroup optix::ContextObj::createGeometryGroup () [inline]

See rtGeometryGroupCreate.

Definition at line 1734 of file optixpp_namespace.h.

2.4.2.14 template<class Iterator > GeometryInstance optix::ContextObj::createGeometryInstance (Geometry *geometry*, Iterator *matlbegin*, Iterator *matlend*) [inline]

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

Definition at line 1699 of file optixpp_namespace.h.

2.4.2.15 GeometryInstance optix::ContextObj::createGeometryInstance () [inline]

See rtGeometryInstanceCreate.

Definition at line 1691 of file optixpp_namespace.h.

2.4.2.16 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 1721 of file optixpp_namespace.h.

2.4.2.17 Group optix::ContextObj::createGroup () [inline]

See rtGroupCreate.

Definition at line 1713 of file optixpp_namespace.h.

2.4.2.18 Material optix::ContextObj::createMaterial () [inline]

See rtMaterialCreate.

Definition at line 1762 of file optixpp_namespace.h.

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

See rtProgramCreateFromPTXFile.

Definition at line 1769 of file optixpp_namespace.h.

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

See rtProgramCreateFromPTXString.

Definition at line 1776 of file optixpp_namespace.h.

2.4.2.21 Selector optix::ContextObj::createSelector () [inline]

See rtSelectorCreate.

Definition at line 1783 of file optixpp_namespace.h.

2.4.2.22 TextureSampler optix::ContextObj::createTextureSampler () [inline]

See rtTextureSamplerCreate.

Definition at line 1790 of file optixpp_namespace.h.

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

Create TextureSampler from GL image. See rtTextureSamplerCreateFromGLImage.

Definition at line 1677 of file optixpp_namespace.h.

2.4.2.24 Transform optix::ContextObj::createTransform () [inline]

See rtTransformCreate.

Definition at line 1755 of file optixpp_namespace.h.

2.4.2.25 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 2038 of file optixpp_namespace.h.

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

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

Implements optix::DestroyableObj.

Definition at line 1546 of file optixpp_namespace.h.

2.4.2.27 RTcontext optix::ContextObj::get () [inline]

Return the OptiX C API RTcontext object.

Definition at line 2072 of file optixpp_namespace.h.

2.4.2.28 RTsize optix::ContextObj::getAvailableDeviceMemory (int *ordinal*) [inline]

See rtContextGetAttribute.

Definition at line 1862 of file optixpp_namespace.h.

2.4.2.29 Context optix::ContextObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 1501 of file optixpp_namespace.h.

2.4.2.30 int optix::ContextObj::getCPUNumThreads () [inline]

See rtContextGetAttribute.

Definition at line 1834 of file optixpp_namespace.h.

2.4.2.31 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 1531 of file optixpp_namespace.h.

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

Call rtDeviceGetDeviceCount and returns number of valid devices.

Definition at line 1512 of file optixpp_namespace.h.

2.4.2.33 std::string optix::ContextObj::getDeviceName (int *ordinal*) [inline, static]

Call rtDeviceGetAttribute and return the name of the device.

Definition at line 1521 of file optixpp_namespace.h.

2.4.2.34 unsigned int optix::ContextObj::getEnabledDeviceCount () [inline]

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

Definition at line 1820 of file optixpp_namespace.h.

2.4.2.35 std::vector< int > optix::ContextObj::getEnabledDevices () [inline]

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

Definition at line 1812 of file optixpp_namespace.h.

2.4.2.36 unsigned int optix::ContextObj::getEntryPointCount () [inline]

See rtContextgetEntryPointCount.

Definition at line 1903 of file optixpp_namespace.h.

2.4.2.37 std::string optix::ContextObj::getErrorString (RTresult *code*) [inline]

See rtContextGetErrroString.

Definition at line 1797 of file optixpp_namespace.h.

2.4.2.38 bool optix::ContextObj::getExceptionEnabled (RTexception *exception*) [inline]

See rtContextGetExceptionEnabled.

Definition at line 1942 of file optixpp_namespace.h.

2.4.2.39 Program optix::ContextObj::getExceptionProgram (unsigned int *entry_point_index*) [inline]

See rtContextGetExceptionProgram.

Definition at line 1929 of file optixpp_namespace.h.

2.4.2.40 int optix::ContextObj::getGPUPagingActive () [inline]

See rtContextGetAttribute.

Definition at line 1848 of file optixpp_namespace.h.

2.4.2.41 int optix::ContextObj::getGPUPagingForcedOff () [inline]

See rtContextGetAttribute.

Definition at line 1855 of file optixpp_namespace.h.

2.4.2.42 int optix::ContextObj::getMaxTextureCount () [inline]

See rtContextGetAttribute

Definition at line 1827 of file optixpp_namespace.h.

2.4.2.43 Program optix::ContextObj::getMissProgram (unsigned int *ray_type_index*) [inline]

See rtContextGetMissProgram.

Definition at line 1967 of file optixpp_namespace.h.

2.4.2.44 RTsize optix::ContextObj::getPrintBufferSize () [inline]

See rtContextGetPrintBufferSize.

Definition at line 2019 of file optixpp_namespace.h.

2.4.2.45 bool optix::ContextObj::getPrintEnabled () [inline]

See rtContextGetPrintEnabled.

Definition at line 2007 of file optixpp_namespace.h.

2.4.2.46 optix::int3 optix::ContextObj::getPrintLaunchIndex () [inline]

See rtContextGetPrintLaunchIndex.

Definition at line 2031 of file optixpp_namespace.h.

2.4.2.47 Program optix::ContextObj::getRayGenerationProgram (unsigned int *entry_point_index*) [inline]

See rtContextGetRayGenerationProgram.

Definition at line 1916 of file optixpp_namespace.h.

2.4.2.48 unsigned int optix::ContextObj::getRayTypeCount () [inline]

See rtContextGetRayTypeCount.

Definition at line 1955 of file optixpp_namespace.h.

2.4.2.49 int optix::ContextObj::getRunningState () [inline]

See rtContextGetRunningState.

Definition at line 1995 of file optixpp_namespace.h.

2.4.2.50 RTsize optix::ContextObj::getStackSize () [inline]

See rtContextGetStackSize.

Definition at line 1886 of file optixpp_namespace.h.

2.4.2.51 RTsize optix::ContextObj::getUsedHostMemory () [inline]

See rtContextGetAttribute.

Definition at line 1841 of file optixpp_namespace.h.

2.4.2.52 Variable optix::ContextObj::getVariable (unsigned int *index*) [inline, virtual]

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

Implements optix::ScopedObj.

Definition at line 2064 of file optixpp_namespace.h.

2.4.2.53 unsigned int optix::ContextObj::getVariableCount () [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 2057 of file optixpp_namespace.h.

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

See rtContextLaunch3D.

Definition at line 1989 of file optixpp_namespace.h.

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

See rtContextLaunch2D.

Definition at line 1984 of file optixpp_namespace.h.

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

See rtContextLaunch1D

Definition at line 1979 of file optixpp_namespace.h.

2.4.2.57 Variable optix::ContextObj::queryVariable (const std::string & *name*) [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 2045 of file optixpp_namespace.h.

2.4.2.58 void optix::ContextObj::removeVariable (Variable *v*) [inline, virtual]

Remove a variable associated with this object.

Implements optix::ScopedObj.

Definition at line 2052 of file optixpp_namespace.h.

2.4.2.59 void optix::ContextObj::setCPUNumThreads (int *cpu_num_threads*) [inline]

See rtContextSetAttribute

Definition at line 1871 of file optixpp_namespace.h.

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

See rtContextSetDevices

Definition at line 1805 of file optixpp_namespace.h.

2.4.2.61 void optix::ContextObj::setEntryPointCount (unsigned int *num_entry_points*) [inline]

See rtContextSetEntryPointCount.

Definition at line 1898 of file optixpp_namespace.h.

2.4.2.62 void optix::ContextObj::setExceptionEnabled (RTexception *exception*, bool *enabled*) [inline]

See rtContextSetExceptionEnabled.

Definition at line 1937 of file optixpp_namespace.h.

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

See rtContextSetExceptionProgram.

Definition at line 1924 of file optixpp_namespace.h.

2.4.2.64 void optix::ContextObj::setGPUPagingForcedOff (int *gpu_paging_forced_off*) [inline]

See rtContextSetAttribute.

Definition at line 1876 of file optixpp_namespace.h.

2.4.2.65 void optix::ContextObj::setMissProgram (unsigned int *ray_type_index*, Program *program*) [inline]

See rtContextSetMissProgram.

Definition at line 1962 of file optixpp_namespace.h.

2.4.2.66 void optix::ContextObj::setPrintBufferSize (RTsize *buffer_size_bytes*) [inline]

See rtContextSetPrintBufferSize.

Definition at line 2014 of file optixpp_namespace.h.

2.4.2.67 void optix::ContextObj::setPrintEnabled (bool *enabled*) [inline]

See rtContextSetPrintEnabled

Definition at line 2002 of file optixpp_namespace.h.

2.4.2.68 void optix::ContextObj::setPrintLaunchIndex (int *x*, int *y* = -1, int *z* = -1) [inline]

See rtContextSetPrintLaunchIndex.

Definition at line 2026 of file optixpp_namespace.h.

2.4.2.69 void optix::ContextObj::setRayGenerationProgram (unsigned int *entry_point_index*, Program *program*) [inline]

See rtContextSetRayGenerationProgram

Definition at line 1911 of file optixpp_namespace.h.

2.4.2.70 void optix::ContextObj::setRayTypeCount (unsigned int *num_ray_types*) [inline]

See rtContextSetRayTypeCount.

Definition at line 1950 of file optixpp_namespace.h.

2.4.2.71 void optix::ContextObj::setStackSize (RTsize *stack_size_bytes*) [inline]

See rtContextSetStackSize

Definition at line 1881 of file optixpp_namespace.h.

2.4.2.72 void optix::ContextObj::setTimeoutCallback (RTtimeoutcallback *callback*, double *min_polling_seconds*) [inline]

See rtContextSetTimeoutCallback RTtimeoutcallback is defined as typedef int (∗RTtimeoutcallback)(void).

Definition at line 1893 of file optixpp_namespace.h.

2.4.2.73 void optix::ContextObj::validate () [inline, virtual]

See rtContextValidate.

Implements optix::DestroyableObj.

Definition at line 1552 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 863 of file optixpp_namespace.h.

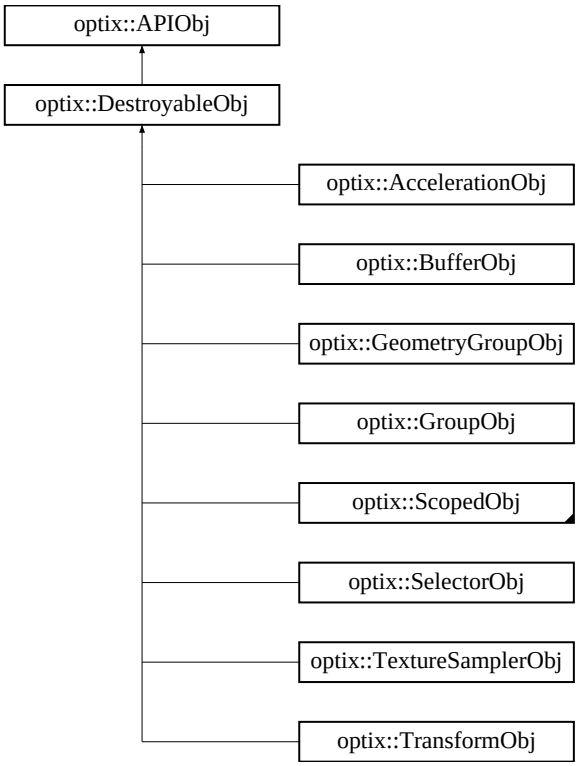
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.5 optix::DestroyableObj Class Reference

Base class for all wrapper objects which can be destroyed and validated.

`#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 337 of file optixpp_namespace.h.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 virtual optix::DestroyableObj::~DestroyableObj () [inline, virtual]

Definition at line 339 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::ContextObj, optix::ProgramObj, optix::GroupObj, optix::GeometryGroupObj, optix::TransformObj, optix::SelectorObj, optix::AccelerationObj, optix::GeometryInstanceObj, optix::GeometryObj, optix::MaterialObj, optix::TextureSamplerObj, and optix::BufferObj.

2.5.3.2 virtual void optix::DestroyableObj::validate () [pure virtual]

call rt[ObjectType]Validate on the underlying OptiX C object

Implemented in optix::ContextObj, optix::ProgramObj, optix::GroupObj, optix::GeometryGroupObj, optix::TransformObj, optix::SelectorObj, optix::AccelerationObj, optix::GeometryInstanceObj, optix::GeometryObj, optix::MaterialObj, optix::TextureSamplerObj, and optix::BufferObj.

The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.6 optix::Exception Class Reference

Exception class for error reporting from the OptiXpp API.

`#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 232 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 235 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 240 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 246 of file optixpp_namespace.h.

2.6.3.2 const std::string& optix::Exception::getErrorString () const [inline]

Retrieve the error message.

Definition at line 243 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 259 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 253 of file optixpp_namespace.h.

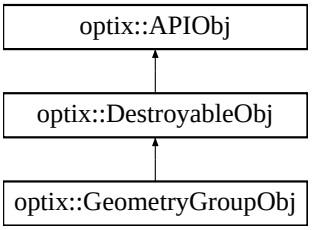
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.7 optix::GeometryGroupObj Class Reference

GeometryGroup wraps the OptiX C API RTgeometrygroup opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::GeometryGroupObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTgeometrygroup get ()

Friends

- class Handle< GeometryGroupObj >

- void setAcceleration (Acceleration acceleration)
- Acceleration getAcceleration ()
- void setChildCount (unsigned int count)
- unsigned int getChildCount ()
- void setChild (unsigned int index, GeometryInstance geometryinstance)
- GeometryInstance getChild (unsigned int index)

2.7.1 Detailed Description

GeometryGroup wraps the OptiX C API RTgeometrygroup opaque type and its associated function set.

Definition at line 946 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 2288 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 2342 of file optixpp_namespace.h.

2.7.2.3 Acceleration optix::GeometryGroupObj::getAcceleration () [inline]

Query the Acceleration structure for this group. See rtGeometryGroupGetAcceleration.

Definition at line 2311 of file optixpp_namespace.h.

2.7.2.4 GeometryInstance optix::GeometryGroupObj::getChild (unsigned int *index*) [inline]

Query an indexed GeometryInstance within this group. See rtGeometryGroupGetChild.

Definition at line 2335 of file optixpp_namespace.h.

2.7.2.5 unsigned int optix::GeometryGroupObj::getChildCount () [inline]

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

Definition at line 2323 of file optixpp_namespace.h.

2.7.2.6 Context optix::GeometryGroupObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2299 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 2306 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 2330 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 2318 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 2294 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 979 of file optixpp_namespace.h.

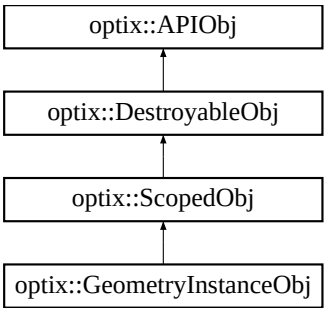
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.8 optix::GeometryInstanceObj Class Reference

GeometryInstance wraps the OptiX C API RTgeometryinstance acceleration opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::GeometryInstanceObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTgeometryinstance get ()

Friends

- class Handle< GeometryInstanceObj >

- void setGeometry (Geometry geometry)
- Geometry getGeometry ()
- void setMaterialCount (unsigned int count)
- unsigned int getMaterialCount ()
- void setMaterial (unsigned int idx, Material material)
- Material getMaterial (unsigned int idx)
- unsigned int addMaterial (Material material)
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)

2.8.1 Detailed Description

GeometryInstance wraps the OptiX C API RTgeometryinstance acceleration opaque type and its associated function set.

Definition at line 1137 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 2537 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 2545 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 2482 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 2578 of file optixpp_namespace.h.

2.8.2.5 Context optix::GeometryInstanceObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2493 of file optixpp_namespace.h.

2.8.2.6 Geometry optix::GeometryInstanceObj::getGeometry () [inline]

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

Definition at line 2505 of file optixpp_namespace.h.

2.8.2.7 Material optix::GeometryInstanceObj::getMaterial (unsigned int *idx*) [inline]

Get the material at given index. See rtGeometryInstanceGetMaterial.

Definition at line 2529 of file optixpp_namespace.h.

2.8.2.8 unsigned int optix::GeometryInstanceObj::getMaterialCount () [inline]

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

Definition at line 2517 of file optixpp_namespace.h.

2.8.2.9 Variable optix::GeometryInstanceObj::getVariable (unsigned int *index*) [inline, virtual]

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

Implements optix::ScopedObj.

Definition at line 2571 of file optixpp_namespace.h.

2.8.2.10 unsigned int optix::GeometryInstanceObj::getVariableCount () [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 2564 of file optixpp_namespace.h.

2.8.2.11 Variable optix::GeometryInstanceObj::queryVariable (const std::string & *name*) [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 2552 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 2559 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 2500 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 2524 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 2512 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 2488 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 1179 of file optixpp_namespace.h.

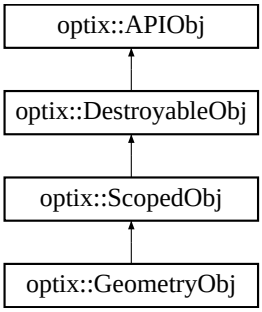
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.9 optix::GeometryObj Class Reference

Geometry wraps the OptiX C API RTgeometry opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::GeometryObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTgeometry get ()

Friends

- class Handle< GeometryObj >

- void markDirty ()
- bool isDirty ()

- void setPrimitiveCount (unsigned int num_primitives)
- unsigned int getPrimitiveCount ()
- void setBoundingBoxProgram (Program program)
- Program getBoundingBoxProgram ()
- void setIntersectionProgram (Program program)
- Program getIntersectionProgram ()
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)

2.9.1 Detailed Description

Geometry wraps the OptiX C API RTgeometry opaque type and its associated function set.

Definition at line 1189 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 2637 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 2583 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 2682 of file optixpp_namespace.h.

2.9.2.4 Program optix::GeometryObj::getBoundingBoxProgram () [inline]

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

Definition at line 2618 of file optixpp_namespace.h.

2.9.2.5 Context optix::GeometryObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2594 of file optixpp_namespace.h.

2.9.2.6 Program optix::GeometryObj::getIntersectionProgram () [inline]

Get the intersection program for this geometry. See rtGeometryGetIntersectionProgram.

Definition at line 2630 of file optixpp_namespace.h.

2.9.2.7 unsigned int optix::GeometryObj::getPrimitiveCount () [inline]

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

Definition at line 2606 of file optixpp_namespace.h.

2.9.2.8 Variable optix::GeometryObj::getVariable (unsigned int *index*) [inline, virtual]

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

Implements optix::ScopedObj.

Definition at line 2663 of file optixpp_namespace.h.

2.9.2.9 unsigned int optix::GeometryObj::getVariableCount () [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 2656 of file optixpp_namespace.h.

2.9.2.10 bool optix::GeometryObj::isDirty () [inline]

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

Definition at line 2675 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 2670 of file optixpp_namespace.h.

2.9.2.12 Variable optix::GeometryObj::queryVariable (const std::string & *name*) [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 2644 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 2651 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 2613 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 2625 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 2601 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 2589 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 1239 of file optixpp_namespace.h.

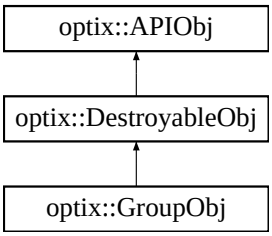
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.10 optix::GroupObj Class Reference

Group wraps the OptiX C API RTgroup opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::GroupObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTgroup get ()

Friends

- class Handle< GroupObj >

- void setAcceleration (Acceleration acceleration)
- Acceleration getAcceleration ()
- void setChildCount (unsigned int count)
- unsigned int getChildCount ()
- template<typename T >
  void setChild (unsigned int index, T child)
- template<typename T >
  T getChild (unsigned int index)

2.10.1 Detailed Description

Group wraps the OptiX C API RTgroup opaque type and its associated function set.

Definition at line 902 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 2133 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 2283 of file optixpp_namespace.h.

2.10.2.3 Acceleration optix::GroupObj::getAcceleration () [inline]

Query the Acceleration structure for this group. See rtGroupGetAcceleration.

Definition at line 2250 of file optixpp_namespace.h.

2.10.2.4 template<typename T > T optix::GroupObj::getChild (unsigned int *index*) [inline]

Query an indexed child within this group. See rtGroupGetChild.

Definition at line 2276 of file optixpp_namespace.h.

2.10.2.5 unsigned int optix::GroupObj::getChildCount () [inline]

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

Definition at line 2262 of file optixpp_namespace.h.

2.10.2.6 Context optix::GroupObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2144 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 2245 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 2270 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 2257 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 2139 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 936 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

The Handle class is a reference counted handle class used to manipulate API objects.

`#include <optixpp_namespace.h>`

Public Member Functions

- Handle ()
- Handle (T *ptr)
- template<class U >
  Handle (U *ptr)
- Handle (const Handle< T > &copy)
- template<class U >
  Handle (const Handle< U > &copy)

- Handle< T > & operator= (const Handle< T > &copy)
- template<class U >
  Handle< T > & operator= (const Handle< U > &copy)
- ~Handle ()
- T * operator-> ()
- T * get ()
- 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 (RTobject 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 122 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 125 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 128 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 132 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 135 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 139 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 151 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 194 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 163 of file optixpp_namespace.h.

2.11.3.3 template<class T> static unsigned int optix::Handle< T >::getDeviceCount () [inline, static]

Query the machine device count. Only valid for contexts.

Definition at line 197 of file optixpp_namespace.h.

2.11.3.4 template<class T> optix::Handle< T >::operator bool () const [inline]

implicit bool cast based on NULLness of wrapped pointer

Definition at line 166 of file optixpp_namespace.h.

2.11.3.5 template<class T> T* optix::Handle< T >::operator-> () [inline]

Dereferences the handle.

Definition at line 160 of file optixpp_namespace.h.

2.11.3.6 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 147 of file optixpp_namespace.h.

2.11.3.7 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 142 of file optixpp_namespace.h.

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

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

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

or

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

Definition at line 598 of file optixpp_namespace.h.

2.11.3.9 template<class T > Handle< VariableObj > optix::Handle< T >::operator[ ] (const std::string & *varname*) [inline]

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 589 of file optixpp_namespace.h.

2.11.3.10 template<class T> static Handle<T> optix::Handle< T >::take (RTobject *p*) [inline, static]

Special version that takes an RTobject which must be cast up to the appropriate OptiX API opaque type.

Definition at line 157 of file optixpp_namespace.h.

2.11.3.11 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 154 of file optixpp_namespace.h.

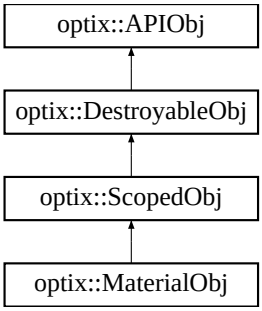
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.12 optix::MaterialObj Class Reference

Material wraps the OptiX C API RTmaterial opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::MaterialObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTmaterial get ()

Friends

- class Handle< MaterialObj >

- void setClosestHitProgram (unsigned int ray_type_index, Program program)
- Program getClosestHitProgram (unsigned int ray_type_index)
- void setAnyHitProgram (unsigned int ray_type_index, Program program)
- Program getAnyHitProgram (unsigned int ray_type_index)
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)

2.12.1 Detailed Description

Material wraps the OptiX C API RTmaterial opaque type and its associated function set.

Definition at line 1249 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 2729 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 2687 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 2762 of file optixpp_namespace.h.

2.12.2.4 Program optix::MaterialObj::getAnyHitProgram (unsigned int *ray_type_index*) [inline]

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

Definition at line 2722 of file optixpp_namespace.h.

2.12.2.5 Program optix::MaterialObj::getClosestHitProgram (unsigned int *ray_type_index*) [inline]

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

Definition at line 2710 of file optixpp_namespace.h.

2.12.2.6 Context optix::MaterialObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2698 of file optixpp_namespace.h.

2.12.2.7 Variable optix::MaterialObj::getVariable (unsigned int *index*) [inline, virtual]

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

Implements optix::ScopedObj.

Definition at line 2755 of file optixpp_namespace.h.

2.12.2.8 unsigned int optix::MaterialObj::getVariableCount () [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 2748 of file optixpp_namespace.h.

2.12.2.9 Variable optix::MaterialObj::queryVariable (const std::string & *name*) [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 2736 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 2743 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 2717 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 2705 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 2693 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 1282 of file optixpp_namespace.h.

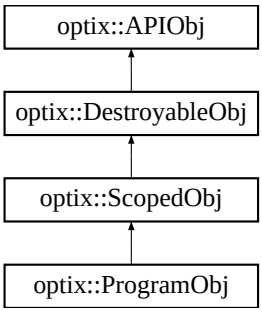
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.13 optix::ProgramObj Class Reference

Program object wraps the OptiX C API RTprogram opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::ProgramObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)
- 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 873 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 2095 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 2077 of file optixpp_namespace.h.

2.13.2.3 RTprogram optix::ProgramObj::get () [inline]

Definition at line 2128 of file optixpp_namespace.h.

2.13.2.4 Context optix::ProgramObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2088 of file optixpp_namespace.h.

2.13.2.5 Variable optix::ProgramObj::getVariable (unsigned int *index*) [inline, virtual]

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

Implements optix::ScopedObj.

Definition at line 2121 of file optixpp_namespace.h.

2.13.2.6 unsigned int optix::ProgramObj::getVariableCount () [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 2114 of file optixpp_namespace.h.

2.13.2.7 Variable optix::ProgramObj::queryVariable (const std::string & *name*) [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 2102 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 2109 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 2083 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 892 of file optixpp_namespace.h.

The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.14 RTUtraversalresult Struct Reference

Structure encapsulating the result of a single ray query.

`#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 interesected 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.

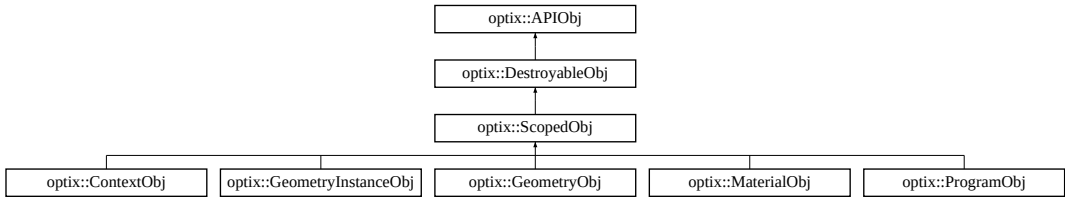
The documentation for this struct was generated from the following file:

- optixu_traversal.h

2.15 optix::ScopedObj Class Reference

Base class for all objects which are OptiX variable containers.

`#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)=0
- virtual void removeVariable (Variable v)=0
- virtual unsigned int getVariableCount ()=0
- virtual Variable getVariable (unsigned int index)=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 363 of file optixpp_namespace.h.

2.15.2 Constructor & Destructor Documentation

2.15.2.1 virtual optix::ScopedObj::~ScopedObj () [inline, virtual]

Definition at line 365 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::ContextObj, optix::ProgramObj, optix::GeometryInstanceObj, optix::GeometryObj, and optix::MaterialObj.

2.15.3.2 virtual Variable optix::ScopedObj::getVariable (unsigned int *index*) [pure virtual]

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

Implemented in optix::ContextObj, optix::ProgramObj, optix::GeometryInstanceObj, optix::GeometryObj, and optix::MaterialObj.

2.15.3.3 virtual unsigned int optix::ScopedObj::getVariableCount () [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::ContextObj, optix::ProgramObj, optix::GeometryInstanceObj, optix::GeometryObj, and optix::MaterialObj.

2.15.3.4 virtual Variable optix::ScopedObj::queryVariable (const std::string & *name*) [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::ContextObj, optix::ProgramObj, optix::GeometryInstanceObj, optix::GeometryObj, and optix::MaterialObj.

2.15.3.5 virtual void optix::ScopedObj::removeVariable (Variable *v*) [pure virtual]

Remove a variable associated with this object.

Implemented in optix::ContextObj, optix::ProgramObj, optix::GeometryInstanceObj, optix::GeometryObj, and optix::MaterialObj.

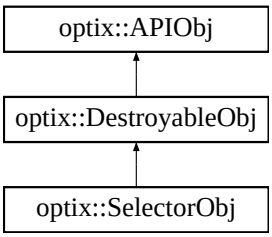
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.16 optix::SelectorObj Class Reference

Selector wraps the OptiX C API RTselector opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::SelectorObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTselector get ()

Friends

- class Handle< SelectorObj >

- void setVisitProgram (Program program)
- Program getVisitProgram ()
- void setChildCount (unsigned int count)
- unsigned int getChildCount ()
- template<typename T >
  void setChild (unsigned int index, T child)
- template<typename T >
  T getChild (unsigned int index)
- Variable declareVariable (const std::string &name)
- Variable queryVariable (const std::string &name)
- void removeVariable (Variable v)
- unsigned int getVariableCount ()
- Variable getVariable (unsigned int index)

2.16.1 Detailed Description

Selector wraps the OptiX C API RTselector opaque type and its associated function set.

Definition at line 1027 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 2207 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 2151 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 2240 of file optixpp_namespace.h.

2.16.2.4 template<typename T > T optix::SelectorObj::getChild (unsigned int *index*) [inline]

Query an indexed child within this group. See rtSelectorGetChild.

Definition at line 2200 of file optixpp_namespace.h.

2.16.2.5 unsigned int optix::SelectorObj::getChildCount () [inline]

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

Definition at line 2186 of file optixpp_namespace.h.

2.16.2.6 Context optix::SelectorObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2162 of file optixpp_namespace.h.

2.16.2.7 Variable optix::SelectorObj::getVariable (unsigned int *index*) [inline]

Definition at line 2233 of file optixpp_namespace.h.

2.16.2.8 unsigned int optix::SelectorObj::getVariableCount () [inline]

Definition at line 2226 of file optixpp_namespace.h.

2.16.2.9 Program optix::SelectorObj::getVisitProgram () [inline]

Get the visitor program for this selector. See rtSelectorGetVisitProgram.

Definition at line 2174 of file optixpp_namespace.h.

2.16.2.10 Variable optix::SelectorObj::queryVariable (const std::string & *name*) [inline]

Definition at line 2214 of file optixpp_namespace.h.

2.16.2.11 void optix::SelectorObj::removeVariable (Variable *v*) [inline]

Definition at line 2221 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 child of this group. See rtSelectorSetChild.

Definition at line 2194 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 2181 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 2169 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 2157 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 1068 of file optixpp_namespace.h.

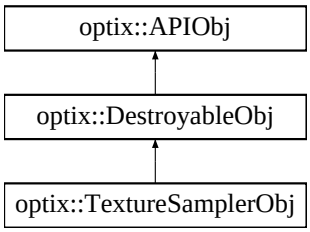
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.17 optix::TextureSamplerObj Class Reference

TextureSampler wraps the OptiX C API RTtexturesampler opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::TextureSamplerObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTtexturesampler get ()

Friends

- class Handle< TextureSamplerObj >

- void setMipLevelCount (unsigned int num_mip_levels)
- unsigned int getMipLevelCount ()
- void setArraySize (unsigned int num_textures_in_array)
- unsigned int getArraySize ()
- void setWrapMode (unsigned int dim, RTwrapmode wrapmode)
- RTwrapmode getWrapMode (unsigned int dim)
- void setFilteringModes (RTfiltermode minification, RTfiltermode magnification, RTfiltermode mipmapping)
- void getFilteringModes (RTfiltermode &minification, RTfiltermode &magnification, RTfiltermode &mipmapping)
- void setMaxAnisotropy (float value)
- float getMaxAnisotropy ()
- void setReadMode (RTtexturereadmode readmode)

- RTtexturereadmode getReadMode ()
- void setIndexingMode (RTtextureindexmode indexmode)
- RTtextureindexmode getIndexingMode ()
- void setBuffer (unsigned int texture_array_idx, unsigned int mip_level, Buffer buffer)
- Buffer getBuffer (unsigned int texture_array_idx, unsigned int mip_level)
- 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 1292 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 2767 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 2879 of file optixpp_namespace.h.

2.17.2.3 unsigned int optix::TextureSamplerObj::getArraySize () [inline]

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

Definition at line 2802 of file optixpp_namespace.h.

2.17.2.4 Buffer optix::TextureSamplerObj::getBuffer (unsigned int *texture_array_idx*, unsigned int *mip_level*) [inline]

Get the underlying buffer used for texture storage. rtTextureSamplerGetBuffer.

Definition at line 2872 of file optixpp_namespace.h.

2.17.2.5 Context optix::TextureSamplerObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2778 of file optixpp_namespace.h.

2.17.2.6 void optix::TextureSamplerObj::getFilteringModes (RTfiltermode & *minification*, RTfiltermode & *magnification*, RTfiltermode & *mipmapping*) [inline]

Query filtering modes for this sampler. See rtTextureSamplerGetFilteringModes.

Definition at line 2826 of file optixpp_namespace.h.

2.17.2.7 RTtextureindexmode optix::TextureSamplerObj::getIndexingMode () [inline]

Query texture indexing mode for this sampler. See rtTextureSamplerGetIndexingMode.

Definition at line 2860 of file optixpp_namespace.h.

2.17.2.8 float optix::TextureSamplerObj::getMaxAnisotropy () [inline]

Query maximum anisotropy for this sampler. See rtTextureSamplerGetMaxAnisotropy.

Definition at line 2836 of file optixpp_namespace.h.

2.17.2.9 unsigned int optix::TextureSamplerObj::getMipLevelCount () [inline]

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

Definition at line 2790 of file optixpp_namespace.h.

2.17.2.10 RTtexturereadmode optix::TextureSamplerObj::getReadMode () [inline]

Query texture read mode for this sampler. See rtTextureSamplerGetReadMode.

Definition at line 2848 of file optixpp_namespace.h.

2.17.2.11 RTwrapmode optix::TextureSamplerObj::getWrapMode (unsigned int *dim*) [inline]

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

Definition at line 2814 of file optixpp_namespace.h.

2.17.2.12 void optix::TextureSamplerObj::registerGLTexture () [inline]

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

Definition at line 2884 of file optixpp_namespace.h.

2.17.2.13 void optix::TextureSamplerObj::setArraySize (unsigned int *num_textures_in_array*) [inline]

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

Definition at line 2797 of file optixpp_namespace.h.

2.17.2.14 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 2867 of file optixpp_namespace.h.

2.17.2.15 void optix::TextureSamplerObj::setFilteringModes (RTfiltermode *minification*, RTfiltermode *magnification*, RTfiltermode *mipmapping*) [inline]

Set filtering modes for this sampler. See rtTextureSamplerSetFilteringModes.

Definition at line 2821 of file optixpp_namespace.h.

2.17.2.16 void optix::TextureSamplerObj::setIndexingMode (RTtextureindexmode *indexmode*) [inline]

Set texture indexing mode for this sampler. See rtTextureSamplerSetIndexingMode.

Definition at line 2855 of file optixpp_namespace.h.

2.17.2.17 void optix::TextureSamplerObj::setMaxAnisotropy (float *value*) [inline]

Set maximum anisotropy for this sampler. See rtTextureSamplerSetMaxAnisotropy.

Definition at line 2831 of file optixpp_namespace.h.

2.17.2.18 void optix::TextureSamplerObj::setMipLevelCount (unsigned int *num_mip_levels*) [inline]

Set the number of mip levels for this sampler. See rtTextureSamplerSetMipLevelCount.

Definition at line 2785 of file optixpp_namespace.h.

2.17.2.19 void optix::TextureSamplerObj::setReadMode (RTtexturereadmode *readmode*) [inline]

Set texture read mode for this sampler. See rtTextureSamplerSetReadMode.

Definition at line 2843 of file optixpp_namespace.h.

2.17.2.20 void optix::TextureSamplerObj::setWrapMode (unsigned int *dim*, RTwrapmode *wrapmode*) [inline]

Set the texture wrap mode for this sampler. See rtTextureSamplerSetWrapMode.

Definition at line 2809 of file optixpp_namespace.h.

2.17.2.21 void optix::TextureSamplerObj::unregisterGLTexture () [inline]

Unregister the texture's buffer, re-enabling OptiX operations. See rtTextureSamplerGLUnregister.

Definition at line 2889 of file optixpp_namespace.h.

2.17.2.22 void optix::TextureSamplerObj::validate () [inline, virtual]

call rt[ObjectType]Validate on the underlying OptiX C object

Implements optix::DestroyableObj.

Definition at line 2773 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 1377 of file optixpp_namespace.h.

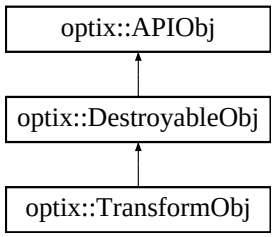
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.18 optix::TransformObj Class Reference

Transform wraps the OptiX C API RTtransform opaque type and its associated function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::TransformObj::



Public Member Functions

- void destroy ()
- void validate ()
- Context getContext ()
- RTtransform get ()

Friends

- class Handle< TransformObj >

- template<typename T >
  void setChild (T child)
- template<typename T >
  T getChild ()
- void setMatrix (bool transpose, const float *matrix, const float *inverse_matrix)
- void getMatrix (bool transpose, float *matrix, float *inverse_matrix)

2.18.1 Detailed Description

Transform wraps the OptiX C API RTtransform opaque type and its associated function set.

Definition at line 989 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 2347 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 2389 of file optixpp_namespace.h.

2.18.2.3 template<typename T > T optix::TransformObj::getChild () [inline]

Set the child node of this transform. See rtTransformGetChild.

Definition at line 2372 of file optixpp_namespace.h.

2.18.2.4 Context optix::TransformObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 2358 of file optixpp_namespace.h.

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

Get the transform matrix for this node. See rtTransformGetMatrix.

Definition at line 2384 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 2366 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 2379 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 2353 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 1017 of file optixpp_namespace.h.

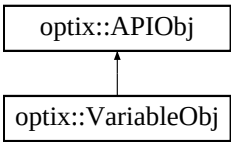
The documentation for this class was generated from the following file:

- optixpp_namespace.h

2.19 optix::VariableObj Class Reference

Variable object wraps OptiX C API RTvariable type and its related function set.

`#include <optixpp_namespace.h>`Inheritance diagram for optix::VariableObj::



Public Member Functions

- Context getContext ()
- std::string getName ()
- std::string getAnnotation ()
- RTobjecttype getType ()
- RTvariable get ()
- RTsize getSize ()

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 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 ()
- unsigned int getUint ()
- int getInt ()

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 (Selector selector)
- void set (Transform transform)

OptiX API object getters

Reitrieve OptiX API object value from a variable

- Buffer getBuffer ()
- TextureSampler getTextureSampler ()

User data variable accessors

- void setUserData (RTsize size, const void *ptr)
- void getUserData (RTsize size, void *ptr)

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 402 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 3410 of file optixpp_namespace.h.

2.19.2.2 std::string optix::VariableObj::getAnnotation () [inline]

Retrieve the annotation associated with the variable.

Definition at line 3396 of file optixpp_namespace.h.

2.19.2.3 Buffer optix::VariableObj::getBuffer () [inline]

Definition at line 3381 of file optixpp_namespace.h.

2.19.2.4 Context optix::VariableObj::getContext () [inline, virtual]

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

Implements optix::APIObj.

Definition at line 3107 of file optixpp_namespace.h.

2.19.2.5 float optix::VariableObj::getFloat () [inline]

Definition at line 3310 of file optixpp_namespace.h.

2.19.2.6 int optix::VariableObj::getInt () [inline]

Definition at line 3324 of file optixpp_namespace.h.

2.19.2.7 std::string optix::VariableObj::getName () [inline]

Retrieve the name of the variable.

Definition at line 3389 of file optixpp_namespace.h.

2.19.2.8 RTsize optix::VariableObj::getSize () [inline]

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

Definition at line 3415 of file optixpp_namespace.h.

2.19.2.9 optix::TextureSampler optix::VariableObj::getTextureSampler () [inline]

Definition at line 3422 of file optixpp_namespace.h.

2.19.2.10 RTobjecttype optix::VariableObj::getType () [inline]

Query the object type of the variable.

Definition at line 3403 of file optixpp_namespace.h.

2.19.2.11 unsigned int optix::VariableObj::getUint () [inline]

Definition at line 3317 of file optixpp_namespace.h.

2.19.2.12 void optix::VariableObj::getUserData (RTsize *size*, void * *ptr*) [inline]

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

Definition at line 3346 of file optixpp_namespace.h.

2.19.2.13 void optix::VariableObj::set (Transform *transform*)

2.19.2.14 void optix::VariableObj::set (Selector *selector*)

2.19.2.15 void optix::VariableObj::set (Group *group*)

2.19.2.16 void optix::VariableObj::set (GeometryGroup *group*)

2.19.2.17 void optix::VariableObj::set (TextureSampler *texturesample*)

2.19.2.18 void optix::VariableObj::set (Buffer *buffer*) [inline]

Definition at line 3336 of file optixpp_namespace.h.

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

Set variable value to a scalar float.

Definition at line 3234 of file optixpp_namespace.h.

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

Definition at line 3290 of file optixpp_namespace.h.

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

Definition at line 3134 of file optixpp_namespace.h.

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

Set variable value to a float2.

Definition at line 3239 of file optixpp_namespace.h.

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

Definition at line 3295 of file optixpp_namespace.h.

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

Definition at line 3139 of file optixpp_namespace.h.

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

Set variable value to a float3.

Definition at line 3244 of file optixpp_namespace.h.

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

Definition at line 3300 of file optixpp_namespace.h.

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

Definition at line 3144 of file optixpp_namespace.h.

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

Set variable value to a float4.

Definition at line 3249 of file optixpp_namespace.h.

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

Definition at line 3305 of file optixpp_namespace.h.

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

Definition at line 3149 of file optixpp_namespace.h.

2.19.2.31 void optix::VariableObj::setBuffer (Buffer *buffer*) [inline]

Definition at line 3331 of file optixpp_namespace.h.

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

Set variable value to a float4.

Definition at line 3229 of file optixpp_namespace.h.

2.19.2.33 void optix::VariableObj::setFloat (optix::float4 *f*) [inline]

Set variable value to a float4.

Definition at line 3224 of file optixpp_namespace.h.

2.19.2.34 void optix::VariableObj::setFloat (float *f1*, float *f2*, float *f3*) [inline]

Set variable value to a float3.

Definition at line 3219 of file optixpp_namespace.h.

2.19.2.35 void optix::VariableObj::setFloat (optix::float3 *f*) [inline]

Set variable value to a float3.

Definition at line 3214 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 3209 of file optixpp_namespace.h.

2.19.2.37 void optix::VariableObj::setFloat (optix::float2 *f*) [inline]

Set variable value to a float2.

Definition at line 3204 of file optixpp_namespace.h.

2.19.2.38 void optix::VariableObj::setFloat (float *f1*) [inline]

Set variable value to a scalar float.

Definition at line 3199 of file optixpp_namespace.h.

2.19.2.39 void optix::VariableObj::setInt (optix::int4 *i*) [inline]

Definition at line 3280 of file optixpp_namespace.h.

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

Definition at line 3285 of file optixpp_namespace.h.

2.19.2.41 void optix::VariableObj::setInt (optix::int3 *i*) [inline]

Definition at line 3270 of file optixpp_namespace.h.

2.19.2.42 void optix::VariableObj::setInt (int *i1*, int *i2*, int *i3*) [inline]

Definition at line 3275 of file optixpp_namespace.h.

2.19.2.43 void optix::VariableObj::setInt (optix::int2 *i*) [inline]

Definition at line 3260 of file optixpp_namespace.h.

2.19.2.44 void optix::VariableObj::setInt (int *i1*, int *i2*) [inline]

Definition at line 3265 of file optixpp_namespace.h.

2.19.2.45 void optix::VariableObj::setInt (int *i1*) [inline]

Definition at line 3255 of file optixpp_namespace.h.

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

Definition at line 3154 of file optixpp_namespace.h.

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

Definition at line 3159 of file optixpp_namespace.h.

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

Definition at line 3164 of file optixpp_namespace.h.

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

Definition at line 3169 of file optixpp_namespace.h.

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

Definition at line 3174 of file optixpp_namespace.h.

2.19.2.51 void optix::VariableObj::setMatrix3x4fv (bool *transpose*, const float * *m*) [inline]

Definition at line 3179 of file optixpp_namespace.h.

2.19.2.52 void optix::VariableObj::setMatrix4x2fv (bool *transpose*, const float * *m*) [inline]

Definition at line 3184 of file optixpp_namespace.h.

2.19.2.53 void optix::VariableObj::setMatrix4x3fv (bool *transpose*, const float * *m*) [inline]

Definition at line 3189 of file optixpp_namespace.h.

2.19.2.54 void optix::VariableObj::setMatrix4x4fv (bool *transpose*, const float * *m*) [inline]

Definition at line 3194 of file optixpp_namespace.h.

2.19.2.55 void optix::VariableObj::setTextureSampler (TextureSampler *texturesample*) [inline]

Definition at line 3351 of file optixpp_namespace.h.

2.19.2.56 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*, unsigned int *u3*, unsigned int *u4*) [inline]

Definition at line 3129 of file optixpp_namespace.h.

2.19.2.57 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*, unsigned int *u3*) [inline]

Definition at line 3124 of file optixpp_namespace.h.

2.19.2.58 void optix::VariableObj::setUint (unsigned int *u1*, unsigned int *u2*) [inline]

Definition at line 3119 of file optixpp_namespace.h.

2.19.2.59 void optix::VariableObj::setUint (unsigned int *u1*) [inline]

Definition at line 3114 of file optixpp_namespace.h.

2.19.2.60 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 3341 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 584 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

A C++ wrapper around the OptiX API. `#include "../optix.h"`

```
#include "../optix_gl_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 prop
      rietary
00006  * rights in and to this software, related documentation and any modifications th
      ereto.
00007  * Any use, reproduction, disclosure or distribution of this software and related

00008  * documentation without an express license agreement from NVIDIA Corporation is
      strictly
00009  * prohibited.
00010  *
00011  * TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THIS SOFTWARE IS PROVIDED *
      AS IS*
00012  * AND NVIDIA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EITHER EXPRESS OR IMPLIE
      D,
00013  * INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNE
      SS FOR A
00014  * PARTICULAR PURPOSE.  IN NO EVENT SHALL NVIDIA OR ITS SUPPLIERS BE LIABLE FOR A
      NY
00015  * SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING,
       WITHOUT
00016  * LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS
      OF
00017  * BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF O
      R
00018  * INABILITY TO USE THIS SOFTWARE, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBI
      LITY OF
00019  * SUCH DAMAGES
00020  */
00021
00022
00052
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
00075 #include <string>
00076 #include <vector>
00077 #include <iterator>
00078 #include "optixu_vector_types.h"
00079
00080 //-----------------------------------------------------------------------------
00081 //
00082 // Doxygen group specifications
00083 //
00084 //-----------------------------------------------------------------------------
00085
00086 //-----------------------------------------------------------------------------
```

```
00087 //
00088 // C++ API
00089 //
00090 //-----------------------------------------------------------------------------
00091
00092 namespace optix {
00095
00096   class AccelerationObj;
00097   class BufferObj;
00098   class ContextObj;
00099   class GeometryObj;
00100   class GeometryGroupObj;
00101   class GeometryInstanceObj;
00102   class GroupObj;
00103   class MaterialObj;
00104   class ProgramObj;
00105   class SelectorObj;
00106   class TextureSamplerObj;
00107   class TransformObj;
00108   class VariableObj;
00109
00110   class APIObj;
00111   class ScopedObj;
00112
00113
00121   template<class T>
00122   class Handle {
00123   public:
00125     Handle() : ptr(0) {}
00126
00128     Handle(T* ptr) : ptr(ptr) { ref(); }
00129
00131     template<class U>
00132     Handle(U* ptr) : ptr(ptr) { ref(); }
00133
00135     Handle(const Handle<T>& copy) : ptr(copy.ptr) { ref(); }
00136
00138     template<class U>
00139     Handle(const Handle<U>& copy) : ptr(copy.ptr) { ref(); }
00140
00142     Handle<T>& operator=(const Handle<T>& copy)
00143     { if(ptr != copy.ptr) { unref(); ptr = copy.ptr; ref(); } return *this; }
00144
00146     template<class U>
00147     Handle<T>& operator=( const Handle<U>& copy)
00148     { if(ptr != copy.ptr) { unref(); ptr = copy.ptr; ref(); } return *this; }
00149
00151     ~Handle() { unref(); }
00152
00154     static Handle<T> take( typename T::api_t p ) { return p? new T(p) : 0; }
00157     static Handle<T> take( RTobject p ) { return p? new T(static_cast<typename T:
      :api_t>(p)) : 0; }
00158
00160     T* operator->() { return ptr; }
00161
00163     T* get() { return ptr; }
00164
00166     operator bool() const  { return ptr != 0; }
00167
00171     Handle<VariableObj> operator[](const std::string& varname);
00172
00191     Handle<VariableObj> operator[](const char* varname);
00192
00194     static Handle<T> create() { return T::create(); }
00195
00197     static unsigned int getDeviceCount() { return T::getDeviceCount(); }
00198
```

```
00199   private:
00200     inline void ref() { if(ptr) ptr->addReference(); }
00201     inline void unref() { if(ptr && ptr->removeReference() == 0) delete ptr; }
00202     T* ptr;
00203   };
00204
00205
00206   //------------------------------------------------------------------------
00207
00208   typedef Handle<AccelerationObj>     Acceleration;
00209   typedef Handle<BufferObj>           Buffer;
00210   typedef Handle<ContextObj>          Context;
00211   typedef Handle<GeometryObj>         Geometry;
00212   typedef Handle<GeometryGroupObj>    GeometryGroup;
00213   typedef Handle<GeometryInstanceObj> GeometryInstance;
00214   typedef Handle<GroupObj>            Group;
00215   typedef Handle<MaterialObj>         Material;
00216   typedef Handle<ProgramObj>          Program;
00217   typedef Handle<SelectorObj>         Selector;
00218   typedef Handle<TextureSamplerObj>   TextureSampler;
00219   typedef Handle<TransformObj>        Transform;
00220   typedef Handle<VariableObj>         Variable;
00221
00222
00223   //------------------------------------------------------------------------
00224
00225
00232   class Exception: public std::exception {
00233   public:
00235     Exception( const std::string& message, RTresult error_code = RT_ERROR_UNKNOWN
      )
00236       : m_message(message), m_error_code( error_code ) {}
00237
00240     virtual ~Exception() throw() {}
00241
00243     const std::string& getErrorString() const { return m_message; }
00244
00246     RTresult getErrorCode() const { return m_error_code; }
00247
00250     static Exception makeException( RTresult code, RTcontext context );
00251
00253     virtual const char* what() const throw() { return getErrorString().c_str(); }

00254   private:
00255     std::string m_message;
00256     RTresult    m_error_code;
00257   };
00258
00259   inline Exception Exception::makeException( RTresult code, RTcontext context )
00260   {
00261     const char* str;
00262     rtContextGetErrorString( context, code, &str);
00263     return Exception( std::string(str), code );
00264   }
00265
00266
00267   //------------------------------------------------------------------------
00268
00269
00288   class APIObj {
00289   public:
00290     APIObj() : ref_count(0) {}
00291     virtual ~APIObj() {}
00292
00294     void addReference()    { ++ref_count; }
00296     int  removeReference() { return --ref_count; }
00297
```

```
00299     virtual Context getContext()=0;
00300
00303     virtual void checkError(RTresult code);
00304
00305     void checkErrorNoGetContext(RTresult code);
00306
00308     static Exception makeException( RTresult code, RTcontext context );
00309   private:
00310     int ref_count;
00311   };
00312
00313   inline Exception APIObj::makeException( RTresult code, RTcontext context )
00314   {
00315     return Exception::makeException( code, context );
00316   }
00317
00318
00319   //-----------------------------------------------------------------------------
00320
00321
00337   class DestroyableObj : public APIObj {
00338   public:
00339     virtual ~DestroyableObj() {}
00340
00342     virtual void destroy() = 0;
00343
00345     virtual void validate() = 0;
00346   };
00347
00348
00349
00350   //-----------------------------------------------------------------------------
00351
00352
00363   class ScopedObj : public DestroyableObj {
00364   public:
00365     virtual ~ScopedObj() {}
00366
00369     virtual Variable declareVariable (const std::string& name) = 0;
00372     virtual Variable queryVariable   (const std::string& name) = 0;
00374     virtual void     removeVariable  (Variable v) = 0;
00378     virtual unsigned int getVariableCount() = 0;
00380     virtual Variable getVariable     (unsigned int index) = 0;
00381   };
00382
00383
00384
00385   //-----------------------------------------------------------------------------
00386
00387
00402   class VariableObj : public APIObj {
00403   public:
00404
00405     Context getContext();
00406
00409
00410
00411     void setFloat(float f1);
00413     void setFloat(optix::float2 f);
00415     void setFloat(float f1, float f2);
00417     void setFloat(optix::float3 f);
00419     void setFloat(float f1, float f2, float f3);
00421     void setFloat(optix::float4 f);
00423     void setFloat(float f1, float f2, float f3, float f4);
00425     void set1fv(const float* f);
00427     void set2fv(const float* f);
00429     void set3fv(const float* f);
```

```
00431     void set4fv(const float* f);
00433
00436
00437     void setInt(int i1);
00438     void setInt(int i1, int i2);
00439     void setInt(optix::int2 i);
00440     void setInt(int i1, int i2, int i3);
00441     void setInt(optix::int3 i);
00442     void setInt(int i1, int i2, int i3, int i4);
00443     void setInt(optix::int4 i);
00444     void set1iv(const int* i);
00445     void set2iv(const int* i);
00446     void set3iv(const int* i);
00447     void set4iv(const int* i);
00449
00452
00453     void setUint(unsigned int u1);
00454     void setUint(unsigned int u1, unsigned int u2);
00455     void setUint(unsigned int u1, unsigned int u2, unsigned int u3);
00456     void setUint(unsigned int u1, unsigned int u2, unsigned int u3, unsigned int
      u4);
00457     void set1uiv(const unsigned int* u);
00458     void set2uiv(const unsigned int* u);
00459     void set3uiv(const unsigned int* u);
00460     void set4uiv(const unsigned int* u);
00462
00465
00466     void setMatrix2x2fv(bool transpose, const float* m);
00467     void setMatrix2x3fv(bool transpose, const float* m);
00468     void setMatrix2x4fv(bool transpose, const float* m);
00469     void setMatrix3x2fv(bool transpose, const float* m);
00470     void setMatrix3x3fv(bool transpose, const float* m);
00471     void setMatrix3x4fv(bool transpose, const float* m);
00472     void setMatrix4x2fv(bool transpose, const float* m);
00473     void setMatrix4x3fv(bool transpose, const float* m);
00474     void setMatrix4x4fv(bool transpose, const float* m);
00476
00479
00480     float getFloat();
00481     unsigned int getUint();
00482     int getInt();
00484
00485 #if 0
00486     // Not implemented yet...
00487
00488     // The getFloat functions can be overloaded by parameter type.
00489     void getFloat(float* f);
00490     void getFloat(float* f1, float* f2);
00491     void getFloat(optix::float2* f);
00492     void getFloat(float* f1, float* f2, float* f3);
00493     void getFloat(optix::float3* f);
00494     void getFloat(float* f1, float* f2, float* f3, float* f4);
00495     void getFloat(optix::float4* f);
00496     // This one will need a different name to distinquish it from 'float getFloat
      ()'.
00497     optix::float2 getFloat2();
00498     optix::float3 getFloat3();
00499     optix::float4 getFloat4();
00500
00501     void get1fv(float* f);
00502     void get2fv(float* f);
00503     void get3fv(float* f);
00504     void get4fv(float* f);
00505
00506     get1i (int* i1);
00507     get2i (int* i1, int* i2);
00508     get3i (int* i1, int* i2, int* i3);
```

```
00509     get4i (int* i1, int* i2, int* i3, int* i4);
00510     get1iv(int* i);
00511     get2iv(int* i);
00512     get3iv(int* i);
00513     get4iv(int* i);
00514
00515     get1ui (unsigned int* u1);
00516     get2ui (unsigned int* u1, unsigned int* u2);
00517     get3ui (unsigned int* u1, unsigned int* u2, unsigned int* u3);
00518     get4ui (unsigned int* u1, unsigned int* u2, unsigned int* u3, unsigned int* u
      4);
00519     get1uiv(unsigned int* u);
00520     get2uiv(unsigned int* u);
00521     get3uiv(unsigned int* u);
00522     get4uiv(unsigned int* u);
00523
00524     getMatrix2x2fv(bool transpose, float* m);
00525     getMatrix2x3fv(bool transpose, float* m);
00526     getMatrix2x4fv(bool transpose, float* m);
00527     getMatrix3x2fv(bool transpose, float* m);
00528     getMatrix3x3fv(bool transpose, float* m);
00529     getMatrix3x4fv(bool transpose, float* m);
00530     getMatrix4x2fv(bool transpose, float* m);
00531     getMatrix4x3fv(bool transpose, float* m);
00532     getMatrix4x4fv(bool transpose, float* m);
00533 #endif
00534
00535
00538
00539     void setBuffer(Buffer buffer);
00540     void set(Buffer buffer);
00541     void setTextureSampler(TextureSampler texturesample);
00542     void set(TextureSampler texturesample);
00543     void set(GeometryGroup group);
00544     void set(Group group);
00545     void set(Selector selector);
00546     void set(Transform transform);
00548
00551
00552     Buffer getBuffer();
00553     TextureSampler getTextureSampler();
00555
00557
00558
00559     void setUserData(RTsize size, const void* ptr);
00561     void getUserData(RTsize size,       void* ptr);
00563
00565     std::string getName();
00566
00568     std::string getAnnotation();
00569
00571     RTobjecttype getType();
00572
00574     RTvariable get();
00575
00577     RTsize getSize();
00578
00579   private:
00580     typedef RTvariable api_t;
00581
00582     RTvariable m_variable;
00583     VariableObj(RTvariable variable) : m_variable(variable) {}
00584     friend class Handle<VariableObj>;
00585
00586   };
00587
00588   template<class T>
```

```
00589   Handle<VariableObj> Handle<T>::operator[](const std::string& varname)
00590   {
00591     Variable v = ptr->queryVariable( varname );
00592     if( v.operator->() == 0)
00593       v = ptr->declareVariable( varname );
00594     return v;
00595   }
00596
00597   template<class T>
00598   Handle<VariableObj> Handle<T>::operator[](const char* varname)
00599   {
00600     return (*this)[ std::string( varname ) ];
00601   }
00602
00603
00604   //------------------------------------------------------------------------
00605
00606
00610   class ContextObj : public ScopedObj {
00611   public:
00612
00614     static unsigned int getDeviceCount();
00615
00617     static std::string getDeviceName(int ordinal);
00618
00620     static void getDeviceAttribute(int ordinal, RTdeviceattribute attrib, RTsize
      size, void* p);
00621
00623     static Context create();
00624
00626     void destroy();
00627
00629     void validate();
00630
00633     Context getContext();
00634
00637     void checkError(RTresult code);
00638
00640     std::string getErrorString( RTresult code );
00642
00645     Acceleration createAcceleration(const char* builder, const char* traverser);
00646
00648     Buffer createBuffer(unsigned int type);
00650     Buffer createBuffer(unsigned int type, RTformat format);
00653     Buffer createBuffer(unsigned int type, RTformat format, RTsize width);
00656     Buffer createBuffer(unsigned int type, RTformat format, RTsize width, RTsize
      height);
00659     Buffer createBuffer(unsigned int type, RTformat format, RTsize width, RTsize
      height, RTsize depth);
00660
00662     Buffer createBufferFromGLBO(unsigned int type, unsigned int vbo);
00663
00665     TextureSampler createTextureSamplerFromGLImage(unsigned int id, RTgltarget ta
      rget);
00666
00667 #ifdef _WIN32
00669     Buffer createBufferFromD3D9Resource(unsigned int type, IDirect3DResource9 *pR
      esource);
00671     Buffer createBufferFromD3D10Resource(unsigned int type, ID3D10Resource *pReso
      urce);
00673     Buffer createBufferFromD3D11Resource(unsigned int type, ID3D11Resource *pReso
      urce);
00674
00676     TextureSampler createTextureSamplerFromD3D9Resource(IDirect3DResource9 *pReso
      urce);
00678     TextureSampler createTextureSamplerFromD3D10Resource(ID3D10Resource *pResourc
      e);
```

```
00680     TextureSampler createTextureSamplerFromD3D11Resource(ID3D11Resource *pResourc
      e);
00681 #endif
00682
00684     Geometry createGeometry();
00686     GeometryInstance createGeometryInstance();
00689     template<class Iterator>
00690     GeometryInstance createGeometryInstance( Geometry geometry, Iterator matlbegi
      n, Iterator matlend );
00691
00693     Group createGroup();
00696     template<class Iterator>
00697     Group createGroup( Iterator childbegin, Iterator childend );
00698
00700     GeometryGroup createGeometryGroup();
00703     template<class Iterator>
00704     GeometryGroup createGeometryGroup( Iterator childbegin, Iterator childend );
00705
00707     Transform createTransform();
00708
00710     Material createMaterial();
00711
00713     Program createProgramFromPTXFile  ( const std::string& ptx, const std::string
      & program_name );
00715     Program createProgramFromPTXString( const std::string& ptx, const std::string
      & program_name );
00716
00718     Selector createSelector();
00719
00721     TextureSampler createTextureSampler();
00723
00726     template<class Iterator>
00727     void setDevices(Iterator begin, Iterator end);
00728
00729 #ifdef _WIN32
00731     void setD3D9Device(IDirect3DDevice9* device);
00733     void setD3D10Device(ID3D10Device* device);
00735     void setD3D11Device(ID3D11Device* device);
00736 #endif
00737
00739     std::vector<int> getEnabledDevices();
00740
00743     unsigned int getEnabledDeviceCount();
00745
00748     int getMaxTextureCount();
00749
00751     int getCPUNumThreads();
00752
00754     RTsize getUsedHostMemory();
00755
00757     int getGPUPagingActive();
00758
00760     int getGPUPagingForcedOff();
00761
00763     RTsize getAvailableDeviceMemory(int ordinal);
00765
00768     void setCPUNumThreads(int cpu_num_threads);
00769
00771     void setGPUPagingForcedOff(int gpu_paging_forced_off);
00773
00776     void setStackSize(RTsize  stack_size_bytes);
00778     RTsize getStackSize();
00779
00782     void setTimeoutCallback(RTtimeoutcallback callback, double min_polling_second
      s);
00783
00785     void setEntryPointCount(unsigned int  num_entry_points);
```

```
00787     unsigned int getEntryPointCount();
00788
00790     void setRayTypeCount(unsigned int  num_ray_types);
00792     unsigned int getRayTypeCount();
00794
00797     void setRayGenerationProgram(unsigned int entry_point_index, Program  program
      );
00799     Program getRayGenerationProgram(unsigned int entry_point_index);
00800
00802     void setExceptionProgram(unsigned int entry_point_index, Program  program);
00804     Program getExceptionProgram(unsigned int entry_point_index);
00805
00807     void setExceptionEnabled( RTexception exception, bool enabled );
00809     bool getExceptionEnabled( RTexception exception );
00810
00812     void setMissProgram(unsigned int ray_type_index, Program  program);
00814     Program getMissProgram(unsigned int ray_type_index);
00816
00818     void compile();
00819
00822     void launch(unsigned int entry_point_index, RTsize image_width);
00824     void launch(unsigned int entry_point_index, RTsize image_width, RTsize image_
      height);
00826     void launch(unsigned int entry_point_index, RTsize image_width, RTsize image_
      height, RTsize image_depth);
00828
00830     int getRunningState();
00831
00834     void setPrintEnabled(bool enabled);
00836     bool getPrintEnabled();
00838     void setPrintBufferSize(RTsize buffer_size_bytes);
00840     RTsize getPrintBufferSize();
00842     void setPrintLaunchIndex(int x, int y=-1, int z=-1);
00844     optix::int3 getPrintLaunchIndex();
00846
00848     Variable declareVariable (const std::string& name);
00849     Variable queryVariable   (const std::string& name);
00850     void     removeVariable  (Variable v);
00851     unsigned int getVariableCount();
00852     Variable getVariable     (unsigned int index);
00854
00856     RTcontext get();
00857   private:
00858     typedef RTcontext api_t;
00859
00860     virtual ~ContextObj() {}
00861     RTcontext m_context;
00862     ContextObj(RTcontext context) : m_context(context) {}
00863     friend class Handle<ContextObj>;
00864   };
00865
00866
00867   //-----------------------------------------------------------------------------
00868
00869
00873   class ProgramObj : public ScopedObj {
00874   public:
00875     void destroy();
00876     void validate();
00877
00878     Context getContext();
00879
00880     Variable declareVariable (const std::string& name);
00881     Variable queryVariable   (const std::string& name);
00882     void     removeVariable  (Variable v);
00883     unsigned int getVariableCount();
00884     Variable getVariable     (unsigned int index);
```

```
00885
00886     RTprogram get();
00887   private:
00888     typedef RTprogram api_t;
00889     virtual ~ProgramObj() {}
00890     RTprogram m_program;
00891     ProgramObj(RTprogram program) : m_program(program) {}
00892     friend class Handle<ProgramObj>;
00893   };
00894
00895
00896   //------------------------------------------------------------------------
00897
00898
00902   class GroupObj : public DestroyableObj {
00903   public:
00904     void destroy();
00905     void validate();
00906
00907     Context getContext();
00908
00911     void setAcceleration(Acceleration acceleration);
00913     Acceleration getAcceleration();
00915
00918     void setChildCount(unsigned int  count);
00920     unsigned int getChildCount();
00921
00923     template< typename T > void setChild(unsigned int index, T child);
00925     template< typename T > T getChild(unsigned int index);
00927
00929     RTgroup get();
00930
00931   private:
00932     typedef RTgroup api_t;
00933     virtual ~GroupObj() {}
00934     RTgroup m_group;
00935     GroupObj(RTgroup group) : m_group(group) {}
00936     friend class Handle<GroupObj>;
00937   };
00938
00939
00940   //------------------------------------------------------------------------
00941
00942
00946   class GeometryGroupObj : public DestroyableObj {
00947   public:
00948     void destroy();
00949     void validate();
00950     Context getContext();
00951
00954     void setAcceleration(Acceleration acceleration);
00956     Acceleration getAcceleration();
00958
00961     void setChildCount(unsigned int  count);
00963     unsigned int getChildCount();
00964
00966     void setChild(unsigned int index, GeometryInstance geometryinstance);
00968     GeometryInstance getChild(unsigned int index);
00970
00972     RTgeometrygroup get();
00973
00974   private:
00975     typedef RTgeometrygroup api_t;
00976     virtual ~GeometryGroupObj() {}
00977     RTgeometrygroup m_geometrygroup;
00978     GeometryGroupObj(RTgeometrygroup geometrygroup) : m_geometrygroup(geometrygro
      up) {}
```

```
00979     friend class Handle<GeometryGroupObj>;
00980   };
00981
00982
00983   //------------------------------------------------------------------------
00984
00985
00989   class TransformObj  : public DestroyableObj {
00990   public:
00991     void destroy();
00992     void validate();
00993     Context getContext();
00994
00997     template< typename T > void setChild(T child);
00999     template< typename T > T getChild();
01001
01004     void setMatrix(bool transpose, const float* matrix, const float* inverse_matr
      ix);
01006     void getMatrix(bool transpose, float* matrix, float* inverse_matrix);
01008
01010     RTtransform get();
01011
01012   private:
01013     typedef RTtransform api_t;
01014     virtual ~TransformObj() {}
01015     RTtransform m_transform;
01016     TransformObj(RTtransform transform) : m_transform(transform) {}
01017     friend class Handle<TransformObj>;
01018   };
01019
01020
01021   //------------------------------------------------------------------------
01022
01023
01027   class SelectorObj : public DestroyableObj {
01028   public:
01029     void destroy();
01030     void validate();
01031     Context getContext();
01032
01035     void setVisitProgram(Program  program);
01037     Program getVisitProgram();
01039
01042     void setChildCount(unsigned int  count);
01044     unsigned int getChildCount();
01045
01047     template< typename T > void setChild(unsigned int index, T child);
01049     template< typename T > T getChild(unsigned int index);
01051
01053     Variable declareVariable (const std::string& name);
01054     Variable queryVariable   (const std::string& name);
01055     void     removeVariable  (Variable v);
01056     unsigned int getVariableCount();
01057     Variable getVariable     (unsigned int index);
01059
01061     RTselector get();
01062
01063   private:
01064     typedef RTselector api_t;
01065     virtual ~SelectorObj() {}
01066     RTselector m_selector;
01067     SelectorObj(RTselector selector) : m_selector(selector) {}
01068     friend class Handle<SelectorObj>;
01069   };
01070
01071
01072   //------------------------------------------------------------------------
```

```
01073
01074
01078   class AccelerationObj : public DestroyableObj {
01079   public:
01080     void destroy();
01081     void validate();
01082     Context getContext();
01083
01086     void markDirty();
01088     bool isDirty();
01090
01094     void        setProperty( const std::string& name, const std::string& value );

01097     std::string getProperty( const std::string& name );
01098
01100     void        setBuilder(const std::string& builder);
01102     std::string getBuilder();
01104     void        setTraverser(const std::string& traverser);
01106     std::string getTraverser();
01108
01111     RTsize getDataSize();
01113     void   getData( void* data );
01115     void   setData( const void* data, RTsize size );
01117
01119     RTacceleration get();
01120
01121   private:
01122     typedef RTacceleration api_t;
01123     virtual ~AccelerationObj() {}
01124     RTacceleration m_acceleration;
01125     AccelerationObj(RTacceleration acceleration) : m_acceleration(acceleration) {
      }
01126     friend class Handle<AccelerationObj>;
01127   };
01128
01129
01130   //-----------------------------------------------------------------------------
01131
01132
01137   class GeometryInstanceObj : public ScopedObj {
01138   public:
01139     void destroy();
01140     void validate();
01141     Context getContext();
01142
01145     void setGeometry(Geometry  geometry);
01147     Geometry getGeometry();
01148
01150     void setMaterialCount(unsigned int  count);
01152     unsigned int getMaterialCount();
01153
01155     void setMaterial(unsigned int idx, Material  material);
01157     Material getMaterial(unsigned int idx);
01158
01160     unsigned int addMaterial(Material material);
01162
01164     Variable declareVariable (const std::string& name);
01165     Variable queryVariable   (const std::string& name);
01166     void     removeVariable  (Variable v);
01167     unsigned int getVariableCount();
01168     Variable getVariable     (unsigned int index);
01170
01172     RTgeometryinstance get();
01173
01174   private:
01175     typedef RTgeometryinstance api_t;
01176     virtual ~GeometryInstanceObj() {}
```

```
01177     RTgeometryinstance m_geometryinstance;
01178     GeometryInstanceObj(RTgeometryinstance geometryinstance) : m_geometryinstance
      (geometryinstance) {}
01179     friend class Handle<GeometryInstanceObj>;
01180   };
01181
01182
01183   //------------------------------------------------------------------------
01184
01185
01189   class GeometryObj : public ScopedObj {
01190   public:
01191     void destroy();
01192     void validate();
01193     Context getContext();
01194
01197     void markDirty();
01199     bool isDirty();
01201
01205     void setPrimitiveCount(unsigned int  num_primitives);
01208     unsigned int getPrimitiveCount();
01210
01213     void setBoundingBoxProgram(Program  program);
01215     Program getBoundingBoxProgram();
01216
01218     void setIntersectionProgram(Program  program);
01220     Program getIntersectionProgram();
01222
01224     Variable declareVariable (const std::string& name);
01225     Variable queryVariable   (const std::string& name);
01226     void     removeVariable  (Variable v);
01227     unsigned int getVariableCount();
01228     Variable getVariable     (unsigned int index);
01230
01232     RTgeometry get();
01233
01234   private:
01235     typedef RTgeometry api_t;
01236     virtual ~GeometryObj() {}
01237     RTgeometry m_geometry;
01238     GeometryObj(RTgeometry geometry) : m_geometry(geometry) {}
01239     friend class Handle<GeometryObj>;
01240   };
01241
01242
01243   //------------------------------------------------------------------------
01244
01245
01249   class MaterialObj : public ScopedObj {
01250   public:
01251     void destroy();
01252     void validate();
01253     Context getContext();
01254
01257     void setClosestHitProgram(unsigned int ray_type_index, Program  program);
01259     Program getClosestHitProgram(unsigned int ray_type_index);
01260
01262     void setAnyHitProgram(unsigned int ray_type_index, Program  program);
01264     Program getAnyHitProgram(unsigned int ray_type_index);
01266
01268     Variable declareVariable (const std::string& name);
01269     Variable queryVariable   (const std::string& name);
01270     void     removeVariable  (Variable v);
01271     unsigned int getVariableCount();
01272     Variable getVariable     (unsigned int index);
01274
01276     RTmaterial get();
```

```
01277   private:
01278     typedef RTmaterial api_t;
01279     virtual ~MaterialObj() {}
01280     RTmaterial m_material;
01281     MaterialObj(RTmaterial material) : m_material(material) {}
01282     friend class Handle<MaterialObj>;
01283   };
01284
01285
01286   //------------------------------------------------------------------------
01287
01288
01292   class TextureSamplerObj : public DestroyableObj {
01293   public:
01294     void destroy();
01295     void validate();
01296     Context getContext();
01297
01300     void setMipLevelCount (unsigned int  num_mip_levels);
01302     unsigned int getMipLevelCount ();
01303
01305     void setArraySize(unsigned int  num_textures_in_array);
01307     unsigned int getArraySize();
01308
01310     void setWrapMode(unsigned int dim, RTwrapmode wrapmode);
01312     RTwrapmode getWrapMode(unsigned int dim);
01313
01315     void setFilteringModes(RTfiltermode  minification, RTfiltermode  magnificatio
      n, RTfiltermode  mipmapping);
01317     void getFilteringModes(RTfiltermode& minification, RTfiltermode& magnificatio
      n, RTfiltermode& mipmapping);
01318
01320     void setMaxAnisotropy(float value);
01322     float getMaxAnisotropy();
01323
01325     void setReadMode(RTtexturereadmode  readmode);
01327     RTtexturereadmode getReadMode();
01328
01330     void setIndexingMode(RTtextureindexmode  indexmode);
01332     RTtextureindexmode getIndexingMode();
01334
01337     void setBuffer(unsigned int texture_array_idx, unsigned int mip_level,
      Buffer buffer);
01339     Buffer getBuffer(unsigned int texture_array_idx, unsigned int mip_level);
01341
01343     RTtexturesampler get();
01344
01347     void registerGLTexture();
01349     void unregisterGLTexture();
01351
01352 #ifdef _WIN32
01353
01356     void registerD3D9Texture();
01358     void registerD3D10Texture();
01360     void registerD3D11Texture();
01361
01363     void unregisterD3D9Texture();
01365     void unregisterD3D10Texture();
01367     void unregisterD3D11Texture();
01369
01370 #endif
01371
01372   private:
01373     typedef RTtexturesampler api_t;
01374     virtual ~TextureSamplerObj() {}
01375     RTtexturesampler m_texturesampler;
01376     TextureSamplerObj(RTtexturesampler texturesampler) : m_texturesampler(texture
```

```
      sampler) {}
01377     friend class Handle<TextureSamplerObj>;
01378   };
01379
01380
01381   //-----------------------------------------------------------------------------
01382
01383
01387   class BufferObj : public DestroyableObj {
01388   public:
01389     void destroy();
01390     void validate();
01391     Context getContext();
01392
01395     void setFormat    (RTformat  format);
01397     RTformat getFormat();
01398
01400     void setElementSize  (RTsize size_of_element);
01402     RTsize getElementSize();
01403
01405     void setSize(RTsize  width);
01407     void getSize(RTsize& width);
01409     void setSize(RTsize  width, RTsize  height);
01411     void getSize(RTsize& width, RTsize& height);
01414     void setSize(RTsize  width, RTsize  height, RTsize  depth);
01416     void getSize(RTsize& width, RTsize& height, RTsize& depth);
01417
01419     void setSize(unsigned int dimensionality, const RTsize* dims);
01421     void getSize(unsigned int dimensionality,       RTsize* dims);
01422
01424     unsigned int getDimensionality();
01426
01429     unsigned int getGLBOId();
01430
01432     void registerGLBuffer();
01434     void unregisterGLBuffer();
01436
01437 #ifdef _WIN32
01438
01441     void registerD3D9Buffer();
01443     void registerD3D10Buffer();
01445     void registerD3D11Buffer();
01446
01448     void unregisterD3D9Buffer();
01450     void unregisterD3D10Buffer();
01452     void unregisterD3D11Buffer();
01453
01455     IDirect3DResource9* getD3D9Resource();
01457     ID3D10Resource* getD3D10Resource();
01459     ID3D11Resource* getD3D11Resource();
01461
01462 #endif
01463
01466     void* map();
01468     void unmap();
01470
01472     RTbuffer get();
01473
01474   private:
01475     typedef RTbuffer api_t;
01476     virtual ~BufferObj() {}
01477     RTbuffer m_buffer;
01478     BufferObj(RTbuffer buffer) : m_buffer(buffer) {}
01479     friend class Handle<BufferObj>;
01480   };
01481
01482
```

```
01483   //-----------------------------------------------------------------------------
01484
01485
01486   inline void APIObj::checkError( RTresult code )
01487   {
01488     if( code != RT_SUCCESS) {
01489       RTcontext c = this->getContext()->get();
01490       throw Exception::makeException( code, c );
01491     }
01492   }
01493
01494   inline void APIObj::checkErrorNoGetContext( RTresult code )
01495   {
01496     if( code != RT_SUCCESS) {
01497       throw Exception::makeException( code, 0u );
01498     }
01499   }
01500
01501   inline Context ContextObj::getContext()
01502   {
01503     return Context::take( m_context );
01504   }
01505
01506   inline void ContextObj::checkError(RTresult code)
01507   {
01508     if( code != RT_SUCCESS && code != RT_TIMEOUT_CALLBACK )
01509       throw Exception::makeException( code, m_context );
01510   }
01511
01512   inline unsigned int ContextObj::getDeviceCount()
01513   {
01514     unsigned int count;
01515     if( RTresult code = rtDeviceGetDeviceCount(&count) )
01516       throw Exception::makeException( code, 0 );
01517
01518     return count;
01519   }
01520
01521   inline std::string ContextObj::getDeviceName(int ordinal)
01522   {
01523     const RTsize max_string_size = 256;
01524     char name[max_string_size];
01525     if( RTresult code = rtDeviceGetAttribute(ordinal, RT_DEVICE_ATTRIBUTE_NAME,
01526                                              max_string_size, name) )
01527       throw Exception::makeException( code, 0 );
01528     return std::string(name);
01529   }
01530
01531   inline void ContextObj::getDeviceAttribute(int ordinal, RTdeviceattribute attri
      b, RTsize size, void* p)
01532   {
01533     if( RTresult code = rtDeviceGetAttribute(ordinal, attrib, size, p) )
01534       throw Exception::makeException( code, 0 );
01535   }
01536
01537   inline Context ContextObj::create()
01538   {
01539     RTcontext c;
01540     if( RTresult code = rtContextCreate(&c) )
01541       throw Exception::makeException( code, 0 );
01542
01543     return Context::take(c);
01544   }
01545
01546   inline void ContextObj::destroy()
01547   {
01548     checkError( rtContextDestroy( m_context ) );
```

```
01549     m_context = 0;
01550   }
01551
01552   inline void ContextObj::validate()
01553   {
01554     checkError( rtContextValidate( m_context ) );
01555   }
01556
01557   inline Acceleration ContextObj::createAcceleration(const char* builder, const c
      har* traverser)
01558   {
01559     RTacceleration acceleration;
01560     checkError( rtAccelerationCreate( m_context, &acceleration ) );
01561     checkError( rtAccelerationSetBuilder( acceleration, builder ) );
01562     checkError( rtAccelerationSetTraverser( acceleration, traverser ) );
01563     return Acceleration::take(acceleration);
01564   }
01565
01566
01567   inline Buffer ContextObj::createBuffer(unsigned int type)
01568   {
01569     RTbuffer buffer;
01570     checkError( rtBufferCreate( m_context, type, &buffer ) );
01571     return Buffer::take(buffer);
01572   }
01573
01574   inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format)
01575   {
01576     RTbuffer buffer;
01577     checkError( rtBufferCreate( m_context, type, &buffer ) );
01578     checkError( rtBufferSetFormat( buffer, format ) );
01579     return Buffer::take(buffer);
01580   }
01581
01582   inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format, RTsi
      ze width)
01583   {
01584     RTbuffer buffer;
01585     checkError( rtBufferCreate( m_context, type, &buffer ) );
01586     checkError( rtBufferSetFormat( buffer, format ) );
01587     checkError( rtBufferSetSize1D( buffer, width ) );
01588     return Buffer::take(buffer);
01589   }
01590
01591   inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format, RTsi
      ze width, RTsize height)
01592   {
01593     RTbuffer buffer;
01594     checkError( rtBufferCreate( m_context, type, &buffer ) );
01595     checkError( rtBufferSetFormat( buffer, format ) );
01596     checkError( rtBufferSetSize2D( buffer, width, height ) );
01597     return Buffer::take(buffer);
01598   }
01599
01600   inline Buffer ContextObj::createBuffer(unsigned int type, RTformat format, RTsi
      ze width, RTsize height, RTsize depth)
01601   {
01602     RTbuffer buffer;
01603     checkError( rtBufferCreate( m_context, type, &buffer ) );
01604     checkError( rtBufferSetFormat( buffer, format ) );
01605     checkError( rtBufferSetSize3D( buffer, width, height, depth ) );
01606     return Buffer::take(buffer);
01607   }
01608
01609   inline Buffer ContextObj::createBufferFromGLBO(unsigned int type, unsigned int
      vbo)
01610   {
```

```
01611     RTbuffer buffer;
01612     checkError( rtBufferCreateFromGLBO( m_context, type, vbo, &buffer ) );
01613     return Buffer::take(buffer);
01614   }
01615
01616 #ifdef _WIN32
01617
01618   inline Buffer ContextObj::createBufferFromD3D9Resource(unsigned int type, IDire
      ct3DResource9 *pResource)
01619   {
01620     RTbuffer buffer;
01621     checkError( rtBufferCreateFromD3D9Resource( m_context, type, pResource, &buff
      er ) );
01622     return Buffer::take(buffer);
01623   }
01624
01625   inline Buffer ContextObj::createBufferFromD3D10Resource(unsigned int type, ID3D
      10Resource *pResource)
01626   {
01627     RTbuffer buffer;
01628     checkError( rtBufferCreateFromD3D10Resource( m_context, type, pResource, &buf
      fer ) );
01629     return Buffer::take(buffer);
01630   }
01631
01632   inline Buffer ContextObj::createBufferFromD3D11Resource(unsigned int type, ID3D
      11Resource *pResource)
01633   {
01634     RTbuffer buffer;
01635     checkError( rtBufferCreateFromD3D11Resource( m_context, type, pResource, &buf
      fer ) );
01636     return Buffer::take(buffer);
01637   }
01638
01639   inline TextureSampler ContextObj::createTextureSamplerFromD3D9Resource(IDirect3
      DResource9 *pResource)
01640   {
01641     RTtexturesampler textureSampler;
01642     checkError( rtTextureSamplerCreateFromD3D9Resource(m_context, pResource, &tex
      tureSampler));
01643     return TextureSampler::take(textureSampler);
01644   }
01645
01646   inline TextureSampler ContextObj::createTextureSamplerFromD3D10Resource(ID3D10R
      esource *pResource)
01647   {
01648     RTtexturesampler textureSampler;
01649     checkError( rtTextureSamplerCreateFromD3D10Resource(m_context, pResource, &te
      xtureSampler));
01650     return TextureSampler::take(textureSampler);
01651   }
01652
01653   inline TextureSampler ContextObj::createTextureSamplerFromD3D11Resource(ID3D11R
      esource *pResource)
01654   {
01655     RTtexturesampler textureSampler;
01656     checkError( rtTextureSamplerCreateFromD3D11Resource(m_context, pResource, &te
      xtureSampler));
01657     return TextureSampler::take(textureSampler);
01658   }
01659
01660   inline void ContextObj::setD3D9Device(IDirect3DDevice9* device)
01661   {
01662     checkError( rtContextSetD3D9Device( m_context, device ) );
01663   }
01664
01665   inline void ContextObj::setD3D10Device(ID3D10Device* device)
```

```
01666   {
01667     checkError( rtContextSetD3D10Device( m_context, device ) );
01668   }
01669
01670   inline void ContextObj::setD3D11Device(ID3D11Device* device)
01671   {
01672     checkError( rtContextSetD3D11Device( m_context, device ) );
01673   }
01674
01675 #endif
01676
01677   inline TextureSampler ContextObj::createTextureSamplerFromGLImage(unsigned int
      id, RTgltarget target)
01678   {
01679     RTtexturesampler textureSampler;
01680     checkError( rtTextureSamplerCreateFromGLImage(m_context, id, target, &texture
      Sampler));
01681     return TextureSampler::take(textureSampler);
01682   }
01683
01684   inline Geometry ContextObj::createGeometry()
01685   {
01686     RTgeometry geometry;
01687     checkError( rtGeometryCreate( m_context, &geometry ) );
01688     return Geometry::take(geometry);
01689   }
01690
01691   inline GeometryInstance ContextObj::createGeometryInstance()
01692   {
01693     RTgeometryinstance geometryinstance;
01694     checkError( rtGeometryInstanceCreate( m_context, &geometryinstance ) );
01695     return GeometryInstance::take(geometryinstance);
01696   }
01697
01698   template<class Iterator>
01699     GeometryInstance ContextObj::createGeometryInstance( Geometry geometry, Itera
      tor matlbegin, Iterator matlend)
01700   {
01701     GeometryInstance result = createGeometryInstance();
01702     result->setGeometry( geometry );
01703     unsigned int count = 0;
01704     for( Iterator iter = matlbegin; iter != matlend; ++iter )
01705       ++count;
01706     result->setMaterialCount( count );
01707     unsigned int index = 0;
01708     for(Iterator iter = matlbegin; iter != matlend; ++iter, ++index )
01709       result->setMaterial( index, *iter );
01710     return result;
01711   }
01712
01713   inline Group ContextObj::createGroup()
01714   {
01715     RTgroup group;
01716     checkError( rtGroupCreate( m_context, &group ) );
01717     return Group::take(group);
01718   }
01719
01720   template<class Iterator>
01721     inline Group ContextObj::createGroup( Iterator childbegin, Iterator childend
      )
01722   {
01723     Group result = createGroup();
01724     unsigned int count = 0;
01725     for(Iterator iter = childbegin; iter != childend; ++iter )
01726       ++count;
01727     result->setChildCount( count );
01728     unsigned int index = 0;
```

```
01729     for(Iterator iter = childbegin; iter != childend; ++iter, ++index )
01730       result->setChild( index, *iter );
01731     return result;
01732   }
01733
01734   inline GeometryGroup ContextObj::createGeometryGroup()
01735   {
01736     RTgeometrygroup gg;
01737     checkError( rtGeometryGroupCreate( m_context, &gg ) );
01738     return GeometryGroup::take( gg );
01739   }
01740
01741   template<class Iterator>
01742   inline GeometryGroup ContextObj::createGeometryGroup( Iterator childbegin, Iter
      ator childend )
01743   {
01744     GeometryGroup result = createGeometryGroup();
01745     unsigned int count = 0;
01746     for(Iterator iter = childbegin; iter != childend; ++iter )
01747       ++count;
01748     result->setChildCount( count );
01749     unsigned int index = 0;
01750     for(Iterator iter = childbegin; iter != childend; ++iter, ++index )
01751       result->setChild( index, *iter );
01752     return result;
01753   }
01754
01755   inline Transform ContextObj::createTransform()
01756   {
01757     RTtransform t;
01758     checkError( rtTransformCreate( m_context, &t ) );
01759     return Transform::take( t );
01760   }
01761
01762   inline Material ContextObj::createMaterial()
01763   {
01764     RTmaterial material;
01765     checkError( rtMaterialCreate( m_context, &material ) );
01766     return Material::take(material);
01767   }
01768
01769   inline Program ContextObj::createProgramFromPTXFile( const std::string& filenam
      e, const std::string& program_name )
01770   {
01771     RTprogram program;
01772     checkError( rtProgramCreateFromPTXFile( m_context, filename.c_str(), program_
      name.c_str(), &program ) );
01773     return Program::take(program);
01774   }
01775
01776   inline Program ContextObj::createProgramFromPTXString( const std::string& ptx,
      const std::string& program_name )
01777   {
01778     RTprogram program;
01779     checkError( rtProgramCreateFromPTXString( m_context, ptx.c_str(), program_nam
      e.c_str(), &program ) );
01780     return Program::take(program);
01781   }
01782
01783   inline Selector ContextObj::createSelector()
01784   {
01785     RTselector selector;
01786     checkError( rtSelectorCreate( m_context, &selector ) );
01787     return Selector::take(selector);
01788   }
01789
01790   inline TextureSampler ContextObj::createTextureSampler()
```

```
01791   {
01792     RTtexturesampler texturesampler;
01793     checkError( rtTextureSamplerCreate( m_context, &texturesampler ) );
01794     return TextureSampler::take(texturesampler);
01795   }
01796
01797   inline std::string ContextObj::getErrorString( RTresult code )
01798   {
01799     const char* str;
01800     rtContextGetErrorString( m_context, code, &str);
01801     return std::string(str);
01802   }
01803
01804   template<class Iterator> inline
01805     void ContextObj::setDevices(Iterator begin, Iterator end)
01806   {
01807     std::vector<int> devices;
01808     std::copy( begin, end, std::insert_iterator<std::vector<int> >( devices, devi
      ces.begin() ) );
01809     checkError( rtContextSetDevices( m_context, static_cast<unsigned int>(devices
      .size()), &devices[0]) );
01810   }
01811
01812   inline std::vector<int> ContextObj::getEnabledDevices()
01813   {
01814     // Initialize with the number of enabled devices
01815     std::vector<int> devices(getEnabledDeviceCount());
01816     checkError( rtContextGetDevices( m_context, &devices[0] ) );
01817     return devices;
01818   }
01819
01820   inline unsigned int ContextObj::getEnabledDeviceCount()
01821   {
01822     unsigned int num;
01823     checkError( rtContextGetDeviceCount( m_context, &num ) );
01824     return num;
01825   }
01826
01827   inline int ContextObj::getMaxTextureCount()
01828   {
01829     int tex_count;
01830     checkError( rtContextGetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_MAX_TEXTUR
      E_COUNT, sizeof(tex_count), &tex_count) );
01831     return tex_count;
01832   }
01833
01834   inline int ContextObj::getCPUNumThreads()
01835   {
01836     int cpu_num_threads;
01837     checkError( rtContextGetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_CPU_NUM_TH
      READS, sizeof(cpu_num_threads), &cpu_num_threads) );
01838     return cpu_num_threads;
01839   }
01840
01841   inline RTsize ContextObj::getUsedHostMemory()
01842   {
01843     RTsize used_mem;
01844     checkError( rtContextGetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_USED_HOST_
      MEMORY, sizeof(used_mem), &used_mem) );
01845     return used_mem;
01846   }
01847
01848   inline int ContextObj::getGPUPagingActive()
01849   {
01850     int gpu_paging_active;
01851     checkError( rtContextGetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_GPU_PAGING
      _ACTIVE, sizeof(gpu_paging_active), &gpu_paging_active) );
```

```
01852     return gpu_paging_active;
01853   }
01854
01855   inline int ContextObj::getGPUPagingForcedOff()
01856   {
01857     int gpu_paging_forced_off;
01858     checkError( rtContextGetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_GPU_PAGING
      _FORCED_OFF, sizeof(gpu_paging_forced_off), &gpu_paging_forced_off) );
01859     return gpu_paging_forced_off;
01860   }
01861
01862   inline RTsize ContextObj::getAvailableDeviceMemory(int ordinal)
01863   {
01864     RTsize free_mem;
01865     checkError( rtContextGetAttribute( m_context,
01866                                        static_cast<RTcontextattribute>(RT_CONTEXT
      _ATTRIBUTE_AVAILABLE_DEVICE_MEMORY + ordinal),
01867                                        sizeof(free_mem), &free_mem) );
01868     return free_mem;
01869   }
01870
01871   inline void ContextObj::setCPUNumThreads(int cpu_num_threads)
01872   {
01873     checkError( rtContextSetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_CPU_NUM_TH
      READS, sizeof(cpu_num_threads), &cpu_num_threads) );
01874   }
01875
01876   inline void ContextObj::setGPUPagingForcedOff(int gpu_paging_forced_off)
01877   {
01878     checkError( rtContextSetAttribute( m_context, RT_CONTEXT_ATTRIBUTE_GPU_PAGING
      _FORCED_OFF, sizeof(gpu_paging_forced_off), &gpu_paging_forced_off) );
01879   }
01880
01881   inline void ContextObj::setStackSize(RTsize  stack_size_bytes)
01882   {
01883     checkError(rtContextSetStackSize(m_context, stack_size_bytes) );
01884   }
01885
01886   inline RTsize ContextObj::getStackSize()
01887   {
01888     RTsize result;
01889     checkError( rtContextGetStackSize( m_context, &result ) );
01890     return result;
01891   }
01892
01893   inline void ContextObj::setTimeoutCallback(RTtimeoutcallback callback, double m
      in_polling_seconds)
01894   {
01895     checkError( rtContextSetTimeoutCallback( m_context, callback, min_polling_sec
      onds ) );
01896   }
01897
01898   inline void ContextObj::setEntryPointCount(unsigned int  num_entry_points)
01899   {
01900     checkError( rtContextSetEntryPointCount( m_context, num_entry_points ) );
01901   }
01902
01903   inline unsigned int ContextObj::getEntryPointCount()
01904   {
01905     unsigned int result;
01906     checkError( rtContextGetEntryPointCount( m_context, &result ) );
01907     return result;
01908   }
01909
01910
01911   inline void ContextObj::setRayGenerationProgram(unsigned int entry_point_index,
       Program  program)
```

```
01912   {
01913     checkError( rtContextSetRayGenerationProgram( m_context, entry_point_index, p
      rogram->get() ) );
01914   }
01915
01916   inline Program ContextObj::getRayGenerationProgram(unsigned int entry_point_ind
      ex)
01917   {
01918     RTprogram result;
01919     checkError( rtContextGetRayGenerationProgram( m_context, entry_point_index, &
      result ) );
01920     return Program::take( result );
01921   }
01922
01923
01924   inline void ContextObj::setExceptionProgram(unsigned int entry_point_index,
      Program  program)
01925   {
01926     checkError( rtContextSetExceptionProgram( m_context, entry_point_index, progr
      am->get() ) );
01927   }
01928
01929   inline Program ContextObj::getExceptionProgram(unsigned int entry_point_index)
01930   {
01931     RTprogram result;
01932     checkError( rtContextGetExceptionProgram( m_context, entry_point_index, &resu
      lt ) );
01933     return Program::take( result );
01934   }
01935
01936
01937   inline void ContextObj::setExceptionEnabled( RTexception exception, bool enable
      d )
01938   {
01939     checkError( rtContextSetExceptionEnabled( m_context, exception, enabled ) );
01940   }
01941
01942   inline bool ContextObj::getExceptionEnabled( RTexception exception )
01943   {
01944     int enabled;
01945     checkError( rtContextGetExceptionEnabled( m_context, exception, &enabled ) );

01946     return enabled != 0;
01947   }
01948
01949
01950   inline void ContextObj::setRayTypeCount(unsigned int  num_ray_types)
01951   {
01952     checkError( rtContextSetRayTypeCount( m_context, num_ray_types ) );
01953   }
01954
01955   inline unsigned int ContextObj::getRayTypeCount()
01956   {
01957     unsigned int result;
01958     checkError( rtContextGetRayTypeCount( m_context, &result ) );
01959     return result;
01960   }
01961
01962   inline void ContextObj::setMissProgram(unsigned int ray_type_index, Program  pr
      ogram)
01963   {
01964     checkError( rtContextSetMissProgram( m_context, ray_type_index, program->get(
      ) ) );
01965   }
01966
01967   inline Program ContextObj::getMissProgram(unsigned int ray_type_index)
01968   {
```

```
01969     RTprogram result;
01970     checkError( rtContextGetMissProgram( m_context, ray_type_index, &result ) );
01971     return Program::take( result );
01972   }
01973
01974   inline void ContextObj::compile()
01975   {
01976     checkError( rtContextCompile( m_context ) );
01977   }
01978
01979   inline void ContextObj::launch(unsigned int entry_point_index, RTsize image_wid
      th)
01980   {
01981     checkError( rtContextLaunch1D( m_context, entry_point_index, image_width ) );

01982   }
01983
01984   inline void ContextObj::launch(unsigned int entry_point_index, RTsize image_wid
      th, RTsize image_height)
01985   {
01986     checkError( rtContextLaunch2D( m_context, entry_point_index, image_width, ima
      ge_height ) );
01987   }
01988
01989   inline void ContextObj::launch(unsigned int entry_point_index, RTsize image_wid
      th, RTsize image_height, RTsize image_depth)
01990   {
01991     checkError( rtContextLaunch3D( m_context, entry_point_index, image_width, ima
      ge_height, image_depth ) );
01992   }
01993
01994
01995   inline int ContextObj::getRunningState()
01996   {
01997     int result;
01998     checkError( rtContextGetRunningState( m_context, &result ) );
01999     return result;
02000   }
02001
02002   inline void ContextObj::setPrintEnabled(bool enabled)
02003   {
02004     checkError( rtContextSetPrintEnabled( m_context, enabled ) );
02005   }
02006
02007   inline bool ContextObj::getPrintEnabled()
02008   {
02009     int enabled;
02010     checkError( rtContextGetPrintEnabled( m_context, &enabled ) );
02011     return enabled != 0;
02012   }
02013
02014   inline void ContextObj::setPrintBufferSize(RTsize buffer_size_bytes)
02015   {
02016     checkError( rtContextSetPrintBufferSize( m_context, buffer_size_bytes ) );
02017   }
02018
02019   inline RTsize ContextObj::getPrintBufferSize()
02020   {
02021     RTsize result;
02022     checkError( rtContextGetPrintBufferSize( m_context, &result ) );
02023     return result;
02024   }
02025
02026   inline void ContextObj::setPrintLaunchIndex(int x, int y, int z)
02027   {
02028     checkError( rtContextSetPrintLaunchIndex( m_context, x, y, z ) );
02029   }
```

```
02030
02031   inline optix::int3 ContextObj::getPrintLaunchIndex()
02032   {
02033     optix::int3 result;
02034     checkError( rtContextGetPrintLaunchIndex( m_context, &result.x, &result.y, &r
      esult.z ) );
02035     return result;
02036   }
02037
02038   inline Variable ContextObj::declareVariable(const std::string& name)
02039   {
02040     RTvariable v;
02041     checkError( rtContextDeclareVariable( m_context, name.c_str(), &v ) );
02042     return Variable::take( v );
02043   }
02044
02045   inline Variable ContextObj::queryVariable(const std::string& name)
02046   {
02047     RTvariable v;
02048     checkError( rtContextQueryVariable( m_context, name.c_str(), &v ) );
02049     return Variable::take( v );
02050   }
02051
02052   inline void ContextObj::removeVariable(Variable v)
02053   {
02054     checkError( rtContextRemoveVariable( m_context, v->get() ) );
02055   }
02056
02057   inline unsigned int ContextObj::getVariableCount()
02058   {
02059     unsigned int result;
02060     checkError( rtContextGetVariableCount( m_context, &result ) );
02061     return result;
02062   }
02063
02064   inline Variable ContextObj::getVariable(unsigned int index)
02065   {
02066     RTvariable v;
02067     checkError( rtContextGetVariable( m_context, index, &v ) );
02068     return Variable::take( v );
02069   }
02070
02071
02072   inline RTcontext ContextObj::get()
02073   {
02074     return m_context;
02075   }
02076
02077   inline void ProgramObj::destroy()
02078   {
02079     checkError( rtProgramDestroy( m_program ) );
02080     m_program = 0;
02081   }
02082
02083   inline void ProgramObj::validate()
02084   {
02085     checkError( rtProgramValidate( m_program ) );
02086   }
02087
02088   inline Context ProgramObj::getContext()
02089   {
02090     RTcontext c;
02091     checkErrorNoGetContext( rtProgramGetContext( m_program, &c ) );
02092     return Context::take( c );
02093   }
02094
02095   inline Variable ProgramObj::declareVariable(const std::string& name)
```

```
02096   {
02097     RTvariable v;
02098     checkError( rtProgramDeclareVariable( m_program, name.c_str(), &v ) );
02099     return Variable::take( v );
02100   }
02101
02102   inline Variable ProgramObj::queryVariable(const std::string& name)
02103   {
02104     RTvariable v;
02105     checkError( rtProgramQueryVariable( m_program, name.c_str(), &v ) );
02106     return Variable::take( v );
02107   }
02108
02109   inline void ProgramObj::removeVariable(Variable v)
02110   {
02111     checkError( rtProgramRemoveVariable( m_program, v->get() ) );
02112   }
02113
02114   inline unsigned int ProgramObj::getVariableCount()
02115   {
02116     unsigned int result;
02117     checkError( rtProgramGetVariableCount( m_program, &result ) );
02118     return result;
02119   }
02120
02121   inline Variable ProgramObj::getVariable(unsigned int index)
02122   {
02123     RTvariable v;
02124     checkError( rtProgramGetVariable( m_program, index, &v ) );
02125     return Variable::take(v);
02126   }
02127
02128   inline RTprogram ProgramObj::get()
02129   {
02130     return m_program;
02131   }
02132
02133   inline void GroupObj::destroy()
02134   {
02135     checkError( rtGroupDestroy( m_group ) );
02136     m_group = 0;
02137   }
02138
02139   inline void GroupObj::validate()
02140   {
02141     checkError( rtGroupValidate( m_group ) );
02142   }
02143
02144   inline Context GroupObj::getContext()
02145   {
02146     RTcontext c;
02147     checkErrorNoGetContext( rtGroupGetContext( m_group, &c) );
02148     return Context::take(c);
02149   }
02150
02151   inline void SelectorObj::destroy()
02152   {
02153     checkError( rtSelectorDestroy( m_selector ) );
02154     m_selector = 0;
02155   }
02156
02157   inline void SelectorObj::validate()
02158   {
02159     checkError( rtSelectorValidate( m_selector ) );
02160   }
02161
02162   inline Context SelectorObj::getContext()
```

```
02163   {
02164     RTcontext c;
02165     checkErrorNoGetContext( rtSelectorGetContext( m_selector, &c ) );
02166     return Context::take( c );
02167   }
02168
02169   inline void SelectorObj::setVisitProgram(Program program)
02170   {
02171     checkError( rtSelectorSetVisitProgram( m_selector, program->get() ) );
02172   }
02173
02174   inline Program SelectorObj::getVisitProgram()
02175   {
02176     RTprogram result;
02177     checkError( rtSelectorGetVisitProgram( m_selector, &result ) );
02178     return Program::take( result );
02179   }
02180
02181   inline void SelectorObj::setChildCount(unsigned int count)
02182   {
02183     checkError( rtSelectorSetChildCount( m_selector, count) );
02184   }
02185
02186   inline unsigned int SelectorObj::getChildCount()
02187   {
02188     unsigned int result;
02189     checkError( rtSelectorGetChildCount( m_selector, &result ) );
02190     return result;
02191   }
02192
02193   template< typename T >
02194   inline void SelectorObj::setChild(unsigned int index, T child)
02195   {
02196     checkError( rtSelectorSetChild( m_selector, index, child->get() ) );
02197   }
02198
02199   template< typename T >
02200   inline T SelectorObj::getChild(unsigned int index)
02201   {
02202     RTobject result;
02203     checkError( rtSelectorGetChild( m_selector, index, &result ) );
02204     return T::take( result );
02205   }
02206
02207   inline Variable SelectorObj::declareVariable(const std::string& name)
02208   {
02209     RTvariable v;
02210     checkError( rtSelectorDeclareVariable( m_selector, name.c_str(), &v ) );
02211     return Variable::take( v );
02212   }
02213
02214   inline Variable SelectorObj::queryVariable(const std::string& name)
02215   {
02216     RTvariable v;
02217     checkError( rtSelectorQueryVariable( m_selector, name.c_str(), &v ) );
02218     return Variable::take( v );
02219   }
02220
02221   inline void SelectorObj::removeVariable(Variable v)
02222   {
02223     checkError( rtSelectorRemoveVariable( m_selector, v->get() ) );
02224   }
02225
02226   inline unsigned int SelectorObj::getVariableCount()
02227   {
02228     unsigned int result;
02229     checkError( rtSelectorGetVariableCount( m_selector, &result ) );
```

```
02230     return result;
02231   }
02232
02233   inline Variable SelectorObj::getVariable(unsigned int index)
02234   {
02235     RTvariable v;
02236     checkError( rtSelectorGetVariable( m_selector, index, &v ) );
02237     return Variable::take( v );
02238   }
02239
02240   inline RTselector SelectorObj::get()
02241   {
02242     return m_selector;
02243   }
02244
02245   inline void GroupObj::setAcceleration(Acceleration acceleration)
02246   {
02247     checkError( rtGroupSetAcceleration( m_group, acceleration->get() ) );
02248   }
02249
02250   inline Acceleration GroupObj::getAcceleration()
02251   {
02252     RTacceleration result;
02253     checkError( rtGroupGetAcceleration( m_group, &result ) );
02254     return Acceleration::take( result );
02255   }
02256
02257   inline void GroupObj::setChildCount(unsigned int  count)
02258   {
02259     checkError( rtGroupSetChildCount( m_group, count ) );
02260   }
02261
02262   inline unsigned int GroupObj::getChildCount()
02263   {
02264     unsigned int result;
02265     checkError( rtGroupGetChildCount( m_group, &result ) );
02266     return result;
02267   }
02268
02269   template< typename T >
02270   inline void GroupObj::setChild(unsigned int index, T child)
02271   {
02272     checkError( rtGroupSetChild( m_group, index, child->get() ) );
02273   }
02274
02275   template< typename T >
02276   inline T GroupObj::getChild(unsigned int index)
02277   {
02278     RTobject result;
02279     checkError( rtGroupGetChild( m_group, index, &result) );
02280     return T::take( result );
02281   }
02282
02283   inline RTgroup GroupObj::get()
02284   {
02285     return m_group;
02286   }
02287
02288   inline void GeometryGroupObj::destroy()
02289   {
02290     checkError( rtGeometryGroupDestroy( m_geometrygroup ) );
02291     m_geometrygroup = 0;
02292   }
02293
02294   inline void GeometryGroupObj::validate()
02295   {
02296     checkError( rtGeometryGroupValidate( m_geometrygroup ) );
```

```
02297   }
02298
02299   inline Context GeometryGroupObj::getContext()
02300   {
02301     RTcontext c;
02302     checkErrorNoGetContext( rtGeometryGroupGetContext( m_geometrygroup, &c) );
02303     return Context::take(c);
02304   }
02305
02306   inline void GeometryGroupObj::setAcceleration(Acceleration acceleration)
02307   {
02308     checkError( rtGeometryGroupSetAcceleration( m_geometrygroup, acceleration->
      get() ) );
02309   }
02310
02311   inline Acceleration GeometryGroupObj::getAcceleration()
02312   {
02313     RTacceleration result;
02314     checkError( rtGeometryGroupGetAcceleration( m_geometrygroup, &result ) );
02315     return Acceleration::take( result );
02316   }
02317
02318   inline void GeometryGroupObj::setChildCount(unsigned int  count)
02319   {
02320     checkError( rtGeometryGroupSetChildCount( m_geometrygroup, count ) );
02321   }
02322
02323   inline unsigned int GeometryGroupObj::getChildCount()
02324   {
02325     unsigned int result;
02326     checkError( rtGeometryGroupGetChildCount( m_geometrygroup, &result ) );
02327     return result;
02328   }
02329
02330   inline void GeometryGroupObj::setChild(unsigned int index, GeometryInstance chi
      ld)
02331   {
02332     checkError( rtGeometryGroupSetChild( m_geometrygroup, index, child->get() ) )
      ;
02333   }
02334
02335   inline GeometryInstance GeometryGroupObj::getChild(unsigned int index)
02336   {
02337     RTgeometryinstance result;
02338     checkError( rtGeometryGroupGetChild( m_geometrygroup, index, &result) );
02339     return GeometryInstance::take( result );
02340   }
02341
02342   inline RTgeometrygroup GeometryGroupObj::get()
02343   {
02344     return m_geometrygroup;
02345   }
02346
02347   inline void TransformObj::destroy()
02348   {
02349     checkError( rtTransformDestroy( m_transform ) );
02350     m_transform = 0;
02351   }
02352
02353   inline void TransformObj::validate()
02354   {
02355     checkError( rtTransformValidate( m_transform ) );
02356   }
02357
02358   inline Context TransformObj::getContext()
02359   {
02360     RTcontext c;
```

```
02361     checkErrorNoGetContext( rtTransformGetContext( m_transform, &c) );
02362     return Context::take(c);
02363   }
02364
02365   template< typename T >
02366   inline void TransformObj::setChild(T child)
02367   {
02368     checkError( rtTransformSetChild( m_transform, child->get() ) );
02369   }
02370
02371   template< typename T >
02372   inline T TransformObj::getChild()
02373   {
02374     RTobject result;
02375     checkError( rtTransformGetChild( m_transform, &result) );
02376     return T::take( result );
02377   }
02378
02379   inline void TransformObj::setMatrix(bool transpose, const float* matrix, const
      float* inverse_matrix)
02380   {
02381     rtTransformSetMatrix( m_transform, transpose, matrix, inverse_matrix );
02382   }
02383
02384   inline void TransformObj::getMatrix(bool transpose, float* matrix, float* inver
      se_matrix)
02385   {
02386     rtTransformGetMatrix( m_transform, transpose, matrix, inverse_matrix );
02387   }
02388
02389   inline RTtransform TransformObj::get()
02390   {
02391     return m_transform;
02392   }
02393
02394   inline void AccelerationObj::destroy()
02395   {
02396     checkError( rtAccelerationDestroy(m_acceleration) );
02397     m_acceleration = 0;
02398   }
02399
02400   inline void AccelerationObj::validate()
02401   {
02402     checkError( rtAccelerationValidate(m_acceleration) );
02403   }
02404
02405   inline Context AccelerationObj::getContext()
02406   {
02407     RTcontext c;
02408     checkErrorNoGetContext( rtAccelerationGetContext(m_acceleration, &c ) );
02409     return Context::take( c );
02410   }
02411
02412   inline void AccelerationObj::markDirty()
02413   {
02414     checkError( rtAccelerationMarkDirty(m_acceleration) );
02415   }
02416
02417   inline bool AccelerationObj::isDirty()
02418   {
02419     int dirty;
02420     checkError( rtAccelerationIsDirty(m_acceleration,&dirty) );
02421     return dirty != 0;
02422   }
02423
02424   inline void AccelerationObj::setProperty( const std::string& name, const std::s
      tring& value )
```

```
02425   {
02426     checkError( rtAccelerationSetProperty(m_acceleration, name.c_str(), value.c_s
      tr() ) );
02427   }
02428
02429   inline std::string AccelerationObj::getProperty( const std::string& name )
02430   {
02431     const char* s;
02432     checkError( rtAccelerationGetProperty(m_acceleration, name.c_str(), &s ) );
02433     return std::string( s );
02434   }
02435
02436   inline void AccelerationObj::setBuilder(const std::string& builder)
02437   {
02438     checkError( rtAccelerationSetBuilder(m_acceleration, builder.c_str() ) );
02439   }
02440
02441   inline std::string AccelerationObj::getBuilder()
02442   {
02443     const char* s;
02444     checkError( rtAccelerationGetBuilder(m_acceleration, &s ) );
02445     return std::string( s );
02446   }
02447
02448   inline void AccelerationObj::setTraverser(const std::string& traverser)
02449   {
02450     checkError( rtAccelerationSetTraverser(m_acceleration, traverser.c_str() ) );

02451   }
02452
02453   inline std::string AccelerationObj::getTraverser()
02454   {
02455     const char* s;
02456     checkError( rtAccelerationGetTraverser(m_acceleration, &s ) );
02457     return std::string( s );
02458   }
02459
02460   inline RTsize AccelerationObj::getDataSize()
02461   {
02462     RTsize sz;
02463     checkError( rtAccelerationGetDataSize(m_acceleration, &sz) );
02464     return sz;
02465   }
02466
02467   inline void AccelerationObj::getData( void* data )
02468   {
02469     checkError( rtAccelerationGetData(m_acceleration,data) );
02470   }
02471
02472   inline void AccelerationObj::setData( const void* data, RTsize size )
02473   {
02474     checkError( rtAccelerationSetData(m_acceleration,data,size) );
02475   }
02476
02477   inline RTacceleration AccelerationObj::get()
02478   {
02479     return m_acceleration;
02480   }
02481
02482   inline void GeometryInstanceObj::destroy()
02483   {
02484     checkError( rtGeometryInstanceDestroy( m_geometryinstance ) );
02485     m_geometryinstance = 0;
02486   }
02487
02488   inline void GeometryInstanceObj::validate()
02489   {
```

```
02490     checkError( rtGeometryInstanceValidate( m_geometryinstance ) );
02491   }
02492
02493   inline Context GeometryInstanceObj::getContext()
02494   {
02495     RTcontext c;
02496     checkErrorNoGetContext( rtGeometryInstanceGetContext( m_geometryinstance, &c
      ) );
02497     return Context::take( c );
02498   }
02499
02500   inline void GeometryInstanceObj::setGeometry(Geometry geometry)
02501   {
02502     checkError( rtGeometryInstanceSetGeometry( m_geometryinstance, geometry->get(
      ) ) );
02503   }
02504
02505   inline Geometry GeometryInstanceObj::getGeometry()
02506   {
02507     RTgeometry result;
02508     checkError( rtGeometryInstanceGetGeometry( m_geometryinstance, &result ) );
02509     return Geometry::take( result );
02510   }
02511
02512   inline void GeometryInstanceObj::setMaterialCount(unsigned int  count)
02513   {
02514     checkError( rtGeometryInstanceSetMaterialCount( m_geometryinstance, count ) )
      ;
02515   }
02516
02517   inline unsigned int GeometryInstanceObj::getMaterialCount()
02518   {
02519     unsigned int result;
02520     checkError( rtGeometryInstanceGetMaterialCount( m_geometryinstance, &result )
       );
02521     return result;
02522   }
02523
02524   inline void GeometryInstanceObj::setMaterial(unsigned int idx, Material  materi
      al)
02525   {
02526     checkError( rtGeometryInstanceSetMaterial( m_geometryinstance, idx, material-
      >get()) );
02527   }
02528
02529   inline Material GeometryInstanceObj::getMaterial(unsigned int idx)
02530   {
02531     RTmaterial result;
02532     checkError( rtGeometryInstanceGetMaterial( m_geometryinstance, idx, &result )
       );
02533     return Material::take( result );
02534   }
02535
02536   // Adds the material and returns the index to the added material.
02537   inline unsigned int GeometryInstanceObj::addMaterial(Material material)
02538   {
02539     unsigned int old_count = getMaterialCount();
02540     setMaterialCount(old_count+1);
02541     setMaterial(old_count, material);
02542     return old_count;
02543   }
02544
02545   inline Variable GeometryInstanceObj::declareVariable(const std::string& name)
02546   {
02547     RTvariable v;
02548     checkError( rtGeometryInstanceDeclareVariable( m_geometryinstance, name.c_str
      (), &v ) );
```

```
02549     return Variable::take( v );
02550   }
02551
02552   inline Variable GeometryInstanceObj::queryVariable(const std::string& name)
02553   {
02554     RTvariable v;
02555     checkError( rtGeometryInstanceQueryVariable( m_geometryinstance, name.c_str()
      , &v ) );
02556     return Variable::take( v );
02557   }
02558
02559   inline void GeometryInstanceObj::removeVariable(Variable v)
02560   {
02561     checkError( rtGeometryInstanceRemoveVariable( m_geometryinstance, v->get() )
     );
02562   }
02563
02564   inline unsigned int GeometryInstanceObj::getVariableCount()
02565   {
02566     unsigned int result;
02567     checkError( rtGeometryInstanceGetVariableCount( m_geometryinstance, &result )
      );
02568     return result;
02569   }
02570
02571   inline Variable GeometryInstanceObj::getVariable(unsigned int index)
02572   {
02573     RTvariable v;
02574     checkError( rtGeometryInstanceGetVariable( m_geometryinstance, index, &v ) );

02575     return Variable::take( v );
02576   }
02577
02578   inline RTgeometryinstance GeometryInstanceObj::get()
02579   {
02580     return m_geometryinstance;
02581   }
02582
02583   inline void GeometryObj::destroy()
02584   {
02585     checkError( rtGeometryDestroy( m_geometry ) );
02586     m_geometry = 0;
02587   }
02588
02589   inline void GeometryObj::validate()
02590   {
02591     checkError( rtGeometryValidate( m_geometry ) );
02592   }
02593
02594   inline Context GeometryObj::getContext()
02595   {
02596     RTcontext c;
02597     checkErrorNoGetContext( rtGeometryGetContext( m_geometry, &c ) );
02598     return Context::take( c );
02599   }
02600
02601   inline void GeometryObj::setPrimitiveCount(unsigned int  num_primitives)
02602   {
02603     checkError( rtGeometrySetPrimitiveCount( m_geometry, num_primitives ) );
02604   }
02605
02606   inline unsigned int GeometryObj::getPrimitiveCount()
02607   {
02608     unsigned int result;
02609     checkError( rtGeometryGetPrimitiveCount( m_geometry, &result ) );
02610     return result;
02611   }
```

```
02612
02613   inline void GeometryObj::setBoundingBoxProgram(Program  program)
02614   {
02615     checkError( rtGeometrySetBoundingBoxProgram( m_geometry, program->get() ) );
02616   }
02617
02618   inline Program GeometryObj::getBoundingBoxProgram()
02619   {
02620     RTprogram result;
02621     checkError( rtGeometryGetBoundingBoxProgram( m_geometry, &result ) );
02622     return Program::take( result );
02623   }
02624
02625   inline void GeometryObj::setIntersectionProgram(Program  program)
02626   {
02627     checkError( rtGeometrySetIntersectionProgram( m_geometry, program->get() ) );

02628   }
02629
02630   inline Program GeometryObj::getIntersectionProgram()
02631   {
02632     RTprogram result;
02633     checkError( rtGeometryGetIntersectionProgram( m_geometry, &result ) );
02634     return Program::take( result );
02635   }
02636
02637   inline Variable GeometryObj::declareVariable(const std::string& name)
02638   {
02639     RTvariable v;
02640     checkError( rtGeometryDeclareVariable( m_geometry, name.c_str(), &v ) );
02641     return Variable::take( v );
02642   }
02643
02644   inline Variable GeometryObj::queryVariable(const std::string& name)
02645   {
02646     RTvariable v;
02647     checkError( rtGeometryQueryVariable( m_geometry, name.c_str(), &v ) );
02648     return Variable::take( v );
02649   }
02650
02651   inline void GeometryObj::removeVariable(Variable v)
02652   {
02653     checkError( rtGeometryRemoveVariable( m_geometry, v->get() ) );
02654   }
02655
02656   inline unsigned int GeometryObj::getVariableCount()
02657   {
02658     unsigned int result;
02659     checkError( rtGeometryGetVariableCount( m_geometry, &result ) );
02660     return result;
02661   }
02662
02663   inline Variable GeometryObj::getVariable(unsigned int index)
02664   {
02665     RTvariable v;
02666     checkError( rtGeometryGetVariable( m_geometry, index, &v ) );
02667     return Variable::take( v );
02668   }
02669
02670   inline void GeometryObj::markDirty()
02671   {
02672     checkError( rtGeometryMarkDirty(m_geometry) );
02673   }
02674
02675   inline bool GeometryObj::isDirty()
02676   {
02677     int dirty;
```

```
02678     checkError( rtGeometryIsDirty(m_geometry,&dirty) );
02679     return dirty != 0;
02680   }
02681
02682   inline RTgeometry GeometryObj::get()
02683   {
02684     return m_geometry;
02685   }
02686
02687   inline void MaterialObj::destroy()
02688   {
02689     checkError( rtMaterialDestroy( m_material ) );
02690     m_material = 0;
02691   }
02692
02693   inline void MaterialObj::validate()
02694   {
02695     checkError( rtMaterialValidate( m_material ) );
02696   }
02697
02698   inline Context MaterialObj::getContext()
02699   {
02700     RTcontext c;
02701     checkErrorNoGetContext( rtMaterialGetContext( m_material, &c ) );
02702     return Context::take( c );
02703   }
02704
02705   inline void MaterialObj::setClosestHitProgram(unsigned int ray_type_index,
      Program  program)
02706   {
02707     checkError( rtMaterialSetClosestHitProgram( m_material, ray_type_index, progr
      am->get() ) );
02708   }
02709
02710   inline Program MaterialObj::getClosestHitProgram(unsigned int ray_type_index)
02711   {
02712     RTprogram result;
02713     checkError( rtMaterialGetClosestHitProgram( m_material, ray_type_index, &resu
      lt ) );
02714     return Program::take( result );
02715   }
02716
02717   inline void MaterialObj::setAnyHitProgram(unsigned int ray_type_index, Program
       program)
02718   {
02719     checkError( rtMaterialSetAnyHitProgram( m_material, ray_type_index, program->
      get() ) );
02720   }
02721
02722   inline Program MaterialObj::getAnyHitProgram(unsigned int ray_type_index)
02723   {
02724     RTprogram result;
02725     checkError( rtMaterialGetAnyHitProgram( m_material, ray_type_index, &result )
       );
02726     return Program::take( result );
02727   }
02728
02729   inline Variable MaterialObj::declareVariable(const std::string& name)
02730   {
02731     RTvariable v;
02732     checkError( rtMaterialDeclareVariable( m_material, name.c_str(), &v ) );
02733     return Variable::take( v );
02734   }
02735
02736   inline Variable MaterialObj::queryVariable(const std::string& name)
02737   {
02738     RTvariable v;
```

```
02739     checkError( rtMaterialQueryVariable( m_material, name.c_str(), &v) );
02740     return Variable::take( v );
02741   }
02742
02743   inline void MaterialObj::removeVariable(Variable v)
02744   {
02745     checkError( rtMaterialRemoveVariable( m_material, v->get() ) );
02746   }
02747
02748   inline unsigned int MaterialObj::getVariableCount()
02749   {
02750     unsigned int result;
02751     checkError( rtMaterialGetVariableCount( m_material, &result ) );
02752     return result;
02753   }
02754
02755   inline Variable MaterialObj::getVariable(unsigned int index)
02756   {
02757     RTvariable v;
02758     checkError( rtMaterialGetVariable( m_material, index, &v) );
02759     return Variable::take( v );
02760   }
02761
02762   inline RTmaterial MaterialObj::get()
02763   {
02764     return m_material;
02765   }
02766
02767   inline void TextureSamplerObj::destroy()
02768   {
02769     checkError( rtTextureSamplerDestroy( m_texturesampler ) );
02770     m_texturesampler = 0;
02771   }
02772
02773   inline void TextureSamplerObj::validate()
02774   {
02775     checkError( rtTextureSamplerValidate( m_texturesampler ) );
02776   }
02777
02778   inline Context TextureSamplerObj::getContext()
02779   {
02780     RTcontext c;
02781     checkErrorNoGetContext( rtTextureSamplerGetContext( m_texturesampler, &c ) );

02782     return Context::take( c );
02783   }
02784
02785   inline void TextureSamplerObj::setMipLevelCount(unsigned int  num_mip_levels)
02786   {
02787     checkError( rtTextureSamplerSetMipLevelCount(m_texturesampler, num_mip_levels
      ) );
02788   }
02789
02790   inline unsigned int TextureSamplerObj::getMipLevelCount()
02791   {
02792     unsigned int result;
02793     checkError( rtTextureSamplerGetMipLevelCount( m_texturesampler, &result ) );
02794     return result;
02795   }
02796
02797   inline void TextureSamplerObj::setArraySize(unsigned int  num_textures_in_array
      )
02798   {
02799     checkError( rtTextureSamplerSetArraySize( m_texturesampler, num_textures_in_a
      rray ) );
02800   }
02801
```

```
02802   inline unsigned int TextureSamplerObj::getArraySize()
02803   {
02804     unsigned int result;
02805     checkError( rtTextureSamplerGetArraySize( m_texturesampler, &result ) );
02806     return result;
02807   }
02808
02809   inline void TextureSamplerObj::setWrapMode(unsigned int dim, RTwrapmode wrapmod
      e)
02810   {
02811     checkError( rtTextureSamplerSetWrapMode( m_texturesampler, dim, wrapmode ) );

02812   }
02813
02814   inline RTwrapmode TextureSamplerObj::getWrapMode(unsigned int dim)
02815   {
02816     RTwrapmode wrapmode;
02817     checkError( rtTextureSamplerGetWrapMode( m_texturesampler, dim, &wrapmode ) )
      ;
02818     return wrapmode;
02819   }
02820
02821   inline void TextureSamplerObj::setFilteringModes(RTfiltermode  minification, RT
      filtermode  magnification, RTfiltermode  mipmapping)
02822   {
02823     checkError( rtTextureSamplerSetFilteringModes( m_texturesampler, minification
      , magnification, mipmapping ) );
02824   }
02825
02826   inline void TextureSamplerObj::getFilteringModes(RTfiltermode& minification, RT
      filtermode& magnification, RTfiltermode& mipmapping)
02827   {
02828     checkError( rtTextureSamplerGetFilteringModes( m_texturesampler, &minificatio
      n, &magnification, &mipmapping ) );
02829   }
02830
02831   inline void TextureSamplerObj::setMaxAnisotropy(float value)
02832   {
02833     checkError( rtTextureSamplerSetMaxAnisotropy(m_texturesampler, value ) );
02834   }
02835
02836   inline float TextureSamplerObj::getMaxAnisotropy()
02837   {
02838     float result;
02839     checkError( rtTextureSamplerGetMaxAnisotropy( m_texturesampler, &result) );
02840     return result;
02841   }
02842
02843   inline void TextureSamplerObj::setReadMode(RTtexturereadmode  readmode)
02844   {
02845     checkError( rtTextureSamplerSetReadMode( m_texturesampler, readmode ) );
02846   }
02847
02848   inline RTtexturereadmode TextureSamplerObj::getReadMode()
02849   {
02850     RTtexturereadmode result;
02851     checkError( rtTextureSamplerGetReadMode( m_texturesampler, &result) );
02852     return result;
02853   }
02854
02855   inline void TextureSamplerObj::setIndexingMode(RTtextureindexmode  indexmode)
02856   {
02857     checkError( rtTextureSamplerSetIndexingMode( m_texturesampler, indexmode ) );

02858   }
02859
02860   inline RTtextureindexmode TextureSamplerObj::getIndexingMode()
```

```
02861   {
02862     RTtextureindexmode result;
02863     checkError( rtTextureSamplerGetIndexingMode( m_texturesampler, &result ) );
02864     return result;
02865   }
02866
02867   inline void TextureSamplerObj::setBuffer(unsigned int texture_array_idx, unsign
      ed int mip_level, Buffer buffer)
02868   {
02869     checkError( rtTextureSamplerSetBuffer( m_texturesampler, texture_array_idx, m
      ip_level, buffer->get() ) );
02870   }
02871
02872   inline Buffer TextureSamplerObj::getBuffer(unsigned int texture_array_idx, unsi
      gned int mip_level)
02873   {
02874     RTbuffer result;
02875     checkError( rtTextureSamplerGetBuffer(m_texturesampler, texture_array_idx, mi
      p_level, &result ) );
02876     return Buffer::take(result);
02877   }
02878
02879   inline RTtexturesampler TextureSamplerObj::get()
02880   {
02881     return m_texturesampler;
02882   }
02883
02884   inline void TextureSamplerObj::registerGLTexture()
02885   {
02886     checkError( rtTextureSamplerGLRegister( m_texturesampler ) );
02887   }
02888
02889   inline void TextureSamplerObj::unregisterGLTexture()
02890   {
02891     checkError( rtTextureSamplerGLUnregister( m_texturesampler ) );
02892   }
02893
02894 #ifdef _WIN32
02895
02896   inline void TextureSamplerObj::registerD3D9Texture()
02897   {
02898     checkError( rtTextureSamplerD3D9Register( m_texturesampler ) );
02899   }
02900
02901   inline void TextureSamplerObj::registerD3D10Texture()
02902   {
02903     checkError( rtTextureSamplerD3D10Register( m_texturesampler ) );
02904   }
02905
02906   inline void TextureSamplerObj::registerD3D11Texture()
02907   {
02908     checkError( rtTextureSamplerD3D11Register( m_texturesampler ) );
02909   }
02910
02911   inline void TextureSamplerObj::unregisterD3D9Texture()
02912   {
02913     checkError( rtTextureSamplerD3D9Unregister( m_texturesampler ) );
02914   }
02915
02916   inline void TextureSamplerObj::unregisterD3D10Texture()
02917   {
02918     checkError( rtTextureSamplerD3D10Unregister( m_texturesampler ) );
02919   }
02920
02921   inline void TextureSamplerObj::unregisterD3D11Texture()
02922   {
02923     checkError( rtTextureSamplerD3D11Unregister( m_texturesampler ) );
```

```
02924   }
02925
02926 #endif
02927
02928   inline void BufferObj::destroy()
02929   {
02930     checkError( rtBufferDestroy( m_buffer ) );
02931     m_buffer = 0;
02932   }
02933
02934   inline void BufferObj::validate()
02935   {
02936     checkError( rtBufferValidate( m_buffer ) );
02937   }
02938
02939   inline Context BufferObj::getContext()
02940   {
02941     RTcontext c;
02942     checkErrorNoGetContext( rtBufferGetContext( m_buffer, &c ) );
02943     return Context::take( c );
02944   }
02945
02946   inline void BufferObj::setFormat(RTformat format)
02947   {
02948     checkError( rtBufferSetFormat( m_buffer, format ) );
02949   }
02950
02951   inline RTformat BufferObj::getFormat()
02952   {
02953     RTformat result;
02954     checkError( rtBufferGetFormat( m_buffer, &result ) );
02955     return result;
02956   }
02957
02958   inline void BufferObj::setElementSize(RTsize size_of_element)
02959   {
02960     checkError( rtBufferSetElementSize ( m_buffer, size_of_element ) );
02961   }
02962
02963   inline RTsize BufferObj::getElementSize()
02964   {
02965     RTsize result;
02966     checkError( rtBufferGetElementSize ( m_buffer, &result) );
02967     return result;
02968   }
02969
02970   inline void BufferObj::setSize(RTsize width)
02971   {
02972     checkError( rtBufferSetSize1D( m_buffer, width ) );
02973   }
02974
02975   inline void BufferObj::getSize(RTsize& width)
02976   {
02977     checkError( rtBufferGetSize1D( m_buffer, &width ) );
02978   }
02979
02980   inline void BufferObj::setSize(RTsize width, RTsize height)
02981   {
02982     checkError( rtBufferSetSize2D( m_buffer, width, height ) );
02983   }
02984
02985   inline void BufferObj::getSize(RTsize& width, RTsize& height)
02986   {
02987     checkError( rtBufferGetSize2D( m_buffer, &width, &height ) );
02988   }
02989
02990   inline void BufferObj::setSize(RTsize width, RTsize height, RTsize depth)
```

```
02991   {
02992     checkError( rtBufferSetSize3D( m_buffer, width, height, depth ) );
02993   }
02994
02995   inline void BufferObj::getSize(RTsize& width, RTsize& height, RTsize& depth)
02996   {
02997     checkError( rtBufferGetSize3D( m_buffer, &width, &height, &depth ) );
02998   }
02999
03000   inline void BufferObj::setSize(unsigned int dimensionality, const RTsize* dims)

03001   {
03002     checkError( rtBufferSetSizev( m_buffer, dimensionality, dims ) );
03003   }
03004
03005   inline void BufferObj::getSize(unsigned int dimensionality, RTsize* dims)
03006   {
03007     checkError( rtBufferGetSizev( m_buffer, dimensionality, dims ) );
03008   }
03009
03010   inline unsigned int BufferObj::getDimensionality()
03011   {
03012     unsigned int result;
03013     checkError( rtBufferGetDimensionality( m_buffer, &result ) );
03014     return result;
03015   }
03016
03017   inline unsigned int BufferObj::getGLBOId()
03018   {
03019     unsigned int result;
03020     checkError( rtBufferGetGLBOId( m_buffer, &result ) );
03021     return result;
03022   }
03023
03024   inline void BufferObj::registerGLBuffer()
03025   {
03026     checkError( rtBufferGLRegister( m_buffer ) );
03027   }
03028
03029   inline void BufferObj::unregisterGLBuffer()
03030   {
03031     checkError( rtBufferGLUnregister( m_buffer ) );
03032   }
03033
03034 #ifdef _WIN32
03035
03036   inline void BufferObj::registerD3D9Buffer()
03037   {
03038     checkError( rtBufferD3D9Register( m_buffer ) );
03039   }
03040
03041   inline void BufferObj::registerD3D10Buffer()
03042   {
03043     checkError( rtBufferD3D10Register( m_buffer ) );
03044   }
03045
03046   inline void BufferObj::registerD3D11Buffer()
03047   {
03048     checkError( rtBufferD3D11Register( m_buffer ) );
03049   }
03050
03051   inline void BufferObj::unregisterD3D9Buffer()
03052   {
03053     checkError( rtBufferD3D9Unregister( m_buffer ) );
03054   }
03055
03056   inline void BufferObj::unregisterD3D10Buffer()
```

```
03057   {
03058     checkError( rtBufferD3D10Unregister( m_buffer ) );
03059   }
03060
03061   inline void BufferObj::unregisterD3D11Buffer()
03062   {
03063     checkError( rtBufferD3D11Unregister( m_buffer ) );
03064   }
03065
03066   inline IDirect3DResource9* BufferObj::getD3D9Resource()
03067   {
03068     IDirect3DResource9* result = NULL;
03069     checkError( rtBufferGetD3D9Resource( m_buffer, &result ) );
03070     return result;
03071   }
03072
03073   inline ID3D10Resource* BufferObj::getD3D10Resource()
03074   {
03075     ID3D10Resource* result = NULL;
03076     checkError( rtBufferGetD3D10Resource( m_buffer, &result ) );
03077     return result;
03078   }
03079
03080   inline ID3D11Resource* BufferObj::getD3D11Resource()
03081   {
03082     ID3D11Resource* result = NULL;
03083     checkError( rtBufferGetD3D11Resource( m_buffer, &result ) );
03084     return result;
03085   }
03086
03087 #endif
03088
03089   inline void* BufferObj::map()
03090   {
03091     void* result;
03092     checkError( rtBufferMap( m_buffer, &result ) );
03093     return result;
03094   }
03095
03096   inline void BufferObj::unmap()
03097   {
03098     checkError( rtBufferUnmap( m_buffer ) );
03099   }
03100
03101
03102   inline RTbuffer BufferObj::get()
03103   {
03104     return m_buffer;
03105   }
03106
03107   inline Context VariableObj::getContext()
03108   {
03109     RTcontext c;
03110     checkErrorNoGetContext( rtVariableGetContext( m_variable, &c ) );
03111     return Context::take( c );
03112   }
03113
03114   inline void VariableObj::setUint(unsigned int u1)
03115   {
03116     checkError( rtVariableSet1ui( m_variable, u1 ) );
03117   }
03118
03119   inline void VariableObj::setUint(unsigned int u1, unsigned int u2)
03120   {
03121     checkError( rtVariableSet2ui( m_variable, u1, u2 ) );
03122   }
03123
```

```
03124   inline void VariableObj::setUint(unsigned int u1, unsigned int u2, unsigned int
        u3)
03125   {
03126     checkError( rtVariableSet3ui( m_variable, u1, u2, u3 ) );
03127   }
03128
03129   inline void VariableObj::setUint(unsigned int u1, unsigned int u2, unsigned int
        u3, unsigned int u4)
03130   {
03131     checkError( rtVariableSet4ui( m_variable, u1, u2, u3, u4 ) );
03132   }
03133
03134   inline void VariableObj::set1uiv(const unsigned int* u)
03135   {
03136     checkError( rtVariableSet1uiv( m_variable, u ) );
03137   }
03138
03139   inline void VariableObj::set2uiv(const unsigned int* u)
03140   {
03141     checkError( rtVariableSet2uiv( m_variable, u ) );
03142   }
03143
03144   inline void VariableObj::set3uiv(const unsigned int* u)
03145   {
03146     checkError( rtVariableSet3uiv( m_variable, u ) );
03147   }
03148
03149   inline void VariableObj::set4uiv(const unsigned int* u)
03150   {
03151     checkError( rtVariableSet4uiv( m_variable, u ) );
03152   }
03153
03154   inline void VariableObj::setMatrix2x2fv(bool transpose, const float* m)
03155   {
03156     checkError( rtVariableSetMatrix2x2fv( m_variable, (int)transpose, m ) );
03157   }
03158
03159   inline void VariableObj::setMatrix2x3fv(bool transpose, const float* m)
03160   {
03161     checkError( rtVariableSetMatrix2x3fv( m_variable, (int)transpose, m ) );
03162   }
03163
03164   inline void VariableObj::setMatrix2x4fv(bool transpose, const float* m)
03165   {
03166     checkError( rtVariableSetMatrix2x4fv( m_variable, (int)transpose, m ) );
03167   }
03168
03169   inline void VariableObj::setMatrix3x2fv(bool transpose, const float* m)
03170   {
03171     checkError( rtVariableSetMatrix3x2fv( m_variable, (int)transpose, m ) );
03172   }
03173
03174   inline void VariableObj::setMatrix3x3fv(bool transpose, const float* m)
03175   {
03176     checkError( rtVariableSetMatrix3x3fv( m_variable, (int)transpose, m ) );
03177   }
03178
03179   inline void VariableObj::setMatrix3x4fv(bool transpose, const float* m)
03180   {
03181     checkError( rtVariableSetMatrix3x4fv( m_variable, (int)transpose, m ) );
03182   }
03183
03184   inline void VariableObj::setMatrix4x2fv(bool transpose, const float* m)
03185   {
03186     checkError( rtVariableSetMatrix4x2fv( m_variable, (int)transpose, m ) );
03187   }
03188
```

```
03189   inline void VariableObj::setMatrix4x3fv(bool transpose, const float* m)
03190   {
03191     checkError( rtVariableSetMatrix4x3fv( m_variable, (int)transpose, m ) );
03192   }
03193
03194   inline void VariableObj::setMatrix4x4fv(bool transpose, const float* m)
03195   {
03196     checkError( rtVariableSetMatrix4x4fv( m_variable, (int)transpose, m ) );
03197   }
03198
03199   inline void VariableObj::setFloat(float f1)
03200   {
03201     checkError( rtVariableSet1f( m_variable, f1 ) );
03202   }
03203
03204   inline void VariableObj::setFloat(optix::float2 f)
03205   {
03206     checkError( rtVariableSet2fv( m_variable, &f.x ) );
03207   }
03208
03209   inline void VariableObj::setFloat(float f1, float f2)
03210   {
03211     checkError( rtVariableSet2f( m_variable, f1, f2 ) );
03212   }
03213
03214   inline void VariableObj::setFloat(optix::float3 f)
03215   {
03216     checkError( rtVariableSet3fv( m_variable, &f.x ) );
03217   }
03218
03219   inline void VariableObj::setFloat(float f1, float f2, float f3)
03220   {
03221     checkError( rtVariableSet3f( m_variable, f1, f2, f3 ) );
03222   }
03223
03224   inline void VariableObj::setFloat(optix::float4 f)
03225   {
03226     checkError( rtVariableSet4fv( m_variable, &f.x ) );
03227   }
03228
03229   inline void VariableObj::setFloat(float f1, float f2, float f3, float f4)
03230   {
03231     checkError( rtVariableSet4f( m_variable, f1, f2, f3, f4 ) );
03232   }
03233
03234   inline void VariableObj::set1fv(const float* f)
03235   {
03236     checkError( rtVariableSet1fv( m_variable, f ) );
03237   }
03238
03239   inline void VariableObj::set2fv(const float* f)
03240   {
03241     checkError( rtVariableSet2fv( m_variable, f ) );
03242   }
03243
03244   inline void VariableObj::set3fv(const float* f)
03245   {
03246     checkError( rtVariableSet3fv( m_variable, f ) );
03247   }
03248
03249   inline void VariableObj::set4fv(const float* f)
03250   {
03251     checkError( rtVariableSet4fv( m_variable, f ) );
03252   }
03253
03255   inline void VariableObj::setInt(int i1)
03256   {
```

```
03257     checkError( rtVariableSet1i( m_variable, i1 ) );
03258   }
03259
03260   inline void VariableObj::setInt(optix::int2 i)
03261   {
03262     checkError( rtVariableSet2iv( m_variable, &i.x ) );
03263   }
03264
03265   inline void VariableObj::setInt(int i1, int i2)
03266   {
03267     checkError( rtVariableSet2i( m_variable, i1, i2 ) );
03268   }
03269
03270   inline void VariableObj::setInt(optix::int3 i)
03271   {
03272     checkError( rtVariableSet3iv( m_variable, &i.x ) );
03273   }
03274
03275   inline void VariableObj::setInt(int i1, int i2, int i3)
03276   {
03277     checkError( rtVariableSet3i( m_variable, i1, i2, i3 ) );
03278   }
03279
03280   inline void VariableObj::setInt(optix::int4 i)
03281   {
03282     checkError( rtVariableSet4iv( m_variable, &i.x ) );
03283   }
03284
03285   inline void VariableObj::setInt(int i1, int i2, int i3, int i4)
03286   {
03287     checkError( rtVariableSet4i( m_variable, i1, i2, i3, i4 ) );
03288   }
03289
03290   inline void VariableObj::set1iv( const int* i )
03291   {
03292     checkError( rtVariableSet1iv( m_variable, i ) );
03293   }
03294
03295   inline void VariableObj::set2iv( const int* i )
03296   {
03297     checkError( rtVariableSet2iv( m_variable, i ) );
03298   }
03299
03300   inline void VariableObj::set3iv( const int* i )
03301   {
03302     checkError( rtVariableSet3iv( m_variable, i ) );
03303   }
03304
03305   inline void VariableObj::set4iv( const int* i )
03306   {
03307     checkError( rtVariableSet4iv( m_variable, i ) );
03308   }
03309
03310   inline float VariableObj::getFloat()
03311   {
03312     float f;
03313     checkError( rtVariableGet1f( m_variable, &f ) );
03314     return f;
03315   }
03316
03317   inline unsigned int VariableObj::getUint()
03318   {
03319     unsigned int i;
03320     checkError( rtVariableGet1ui( m_variable, &i ) );
03321     return i;
03322   }
03323
```

```
03324   inline int VariableObj::getInt()
03325   {
03326     int i;
03327     checkError( rtVariableGet1i( m_variable, &i ) );
03328     return i;
03329   }
03330
03331   inline void VariableObj::setBuffer(Buffer buffer)
03332   {
03333     checkError( rtVariableSetObject( m_variable, buffer->get() ) );
03334   }
03335
03336   inline void VariableObj::set(Buffer buffer)
03337   {
03338     checkError( rtVariableSetObject( m_variable, buffer->get() ) );
03339   }
03340
03341   inline void VariableObj::setUserData(RTsize size, const void* ptr)
03342   {
03343     checkError( rtVariableSetUserData( m_variable, size, ptr ) );
03344   }
03345
03346   inline void VariableObj::getUserData(RTsize size,       void* ptr)
03347   {
03348     checkError( rtVariableGetUserData( m_variable, size, ptr ) );
03349   }
03350
03351   inline void VariableObj::setTextureSampler(TextureSampler texturesampler)
03352   {
03353     checkError( rtVariableSetObject( m_variable, texturesampler->get() ) );
03354   }
03355
03356   inline void VariableObj::set(TextureSampler texturesampler)
03357   {
03358     checkError( rtVariableSetObject( m_variable, texturesampler->get() ) );
03359   }
03360
03361   inline void VariableObj::set(GeometryGroup group)
03362   {
03363     checkError( rtVariableSetObject( m_variable, group->get() ) );
03364   }
03365
03366   inline void VariableObj::set(Group group)
03367   {
03368     checkError( rtVariableSetObject( m_variable, group->get() ) );
03369   }
03370
03371   inline void VariableObj::set(Selector sel)
03372   {
03373     checkError( rtVariableSetObject( m_variable, sel->get() ) );
03374   }
03375
03376   inline void VariableObj::set(Transform tran)
03377   {
03378     checkError( rtVariableSetObject( m_variable, tran->get() ) );
03379   }
03380
03381   inline Buffer VariableObj::getBuffer()
03382   {
03383     RTobject temp;
03384     checkError( rtVariableGetObject( m_variable, &temp ) );
03385     RTbuffer buffer = reinterpret_cast<RTbuffer>(temp);
03386     return Buffer::take(buffer);
03387   }
03388
03389   inline std::string VariableObj::getName()
03390   {
```

```
03391     const char* name;
03392     checkError( rtVariableGetName( m_variable, &name ) );
03393     return std::string(name);
03394   }
03395
03396   inline std::string VariableObj::getAnnotation()
03397   {
03398     const char* annotation;
03399     checkError( rtVariableGetAnnotation( m_variable, &annotation ) );
03400     return std::string(annotation);
03401   }
03402
03403   inline RTobjecttype VariableObj::getType()
03404   {
03405     RTobjecttype type;
03406     checkError( rtVariableGetType( m_variable, &type ) );
03407     return type;
03408   }
03409
03410   inline RTvariable VariableObj::get()
03411   {
03412     return m_variable;
03413   }
03414
03415   inline RTsize VariableObj::getSize()
03416   {
03417     RTsize size;
03418     checkError( rtVariableGetSize( m_variable, &size ) );
03419     return size;
03420   }
03421
03422   inline optix::TextureSampler VariableObj::getTextureSampler()
03423   {
03424     RTobject temp;
03425     checkError( rtVariableGetObject( m_variable, &temp ) );
03426     RTtexturesampler sampler = reinterpret_cast<RTtexturesampler>(temp);
03427     return TextureSampler::take(sampler);
03428   }
03429
03431 }
03432
03433 #endif /* __optixu_optixpp_namespace_h__ */
03434
03436
```

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 (RTobjecttype 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, RTobject child, unsigned int *index)
- RTresult rtuSelectorAddChild (RTselector selector, RTobject child, unsigned int *index)
- RTresult rtuGeometryGroupAddChild (RTgeometrygroup geometrygroup, RTgeometryinstance child, unsigned int *index)
- RTresult rtuTransformSetChild (RTtransform transform, RTobject child)
- RTresult rtuGroupRemoveChild (RTgroup group, RTobject child)
- RTresult rtuSelectorRemoveChild (RTselector selector, RTobject 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, RTobject child, unsigned int *index)
- RTU_INLINE RTresult rtuSelectorGetChildIndex (RTselector selector, RTobject 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:

context Context

usePTX32InHost64 Use 32bit PTX bounding box and intersection programs in 64bit application. Takes effect only with 64bit host.

mesh Output geometry

num_verts Vertex count

verts Vertices (num_verts∗float∗3) [v1_x, v1_y, v1_z, v2.x, ...]

num_tris Triangle count

indices Vertex indices (num_tris∗unsigned∗3) [tri1_index1, tr1_index2, ...]

mat_indices Indices of materials (num_tris∗unsigned) [tri1_mat_index, tri2_mat_index, ...]

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:

context Context

usePTX32InHost64 Use 32bit PTX bounding box and intersection programs in 64bit application. Takes effect only with 64bit host.

mesh Output geometry

num_verts Vertex count

verts Vertices (num_verts∗float∗3) [v1_x, v1_y, v1_z, v2.x, ...]

num_tris Triangle count

indices Vertex indices (num_tris∗unsigned∗3) [tri1_index1, tr1_index2, ...]

mat_indices Indices of materials (num_tris∗unsigned) [tri1_mat_index, tri2_mat_index, ...]

norms Normals (num_norms∗float∗3) [v1_x, v1_y, v1_z, v2.x, ...]

norm_indices Indices of vertex normals (num_tris∗unsigned∗3) [tri1_norm_index1, tri1_norm_index2 ...]

tex_coords Texture uv coords (num_tex_coords∗float∗2) [t1_u, t1_v, t2_u ...]

tex_indices Indices of texture uv (num_tris∗unsigned∗3) [tri1_tex_index1, tri1_tex_index2 ...]

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*, RTobject *child*, unsigned int * *index*)

Definition at line 353 of file optixu.h.

3.3.2.18 RTU_INLINE RTresult rtuSelectorRemoveChild (RTselector *selector*, RTobject *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*, RTobject *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 prop
      rietary
00006  * rights in and to this software, related documentation and any modifications th
      ereto.
00007  * Any use, reproduction, disclosure or distribution of this software and related

00008  * documentation without an express license agreement from NVIDIA Corporation is
      strictly
00009  * prohibited.
00010  *
00011  * TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THIS SOFTWARE IS PROVIDED *
      AS IS*
00012  * AND NVIDIA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EITHER EXPRESS OR IMPLIE
      D,
00013  * INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNE
      SS FOR A
00014  * PARTICULAR PURPOSE.  IN NO EVENT SHALL NVIDIA OR ITS SUPPLIERS BE LIABLE FOR A
      NY
00015  * SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING,
       WITHOUT
00016  * LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS
      OF
00017  * BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF O
      R
00018  * INABILITY TO USE THIS SOFTWARE, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBI
      LITY OF
00019  * SUCH DAMAGES
00020  */
00021
00022 #ifndef __optix_optixu_h__
00023 #define __optix_optixu_h__
00024
00025 #include <stddef.h>
00026 #include "../optix.h"
00027
00028 #ifdef __cplusplus
00029 #  define RTU_INLINE inline
00030 #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( RTobjecttype type, char* buffer, RTsize bufferSi
      ze );
00046
00047  /*
00048   * Return the size of a given RTformat.  RT_FORMAT_USER and RT_FORMAT_UNKNOWN re
      turn 0.
00049   * Returns RT_ERROR_INVALID_VALUE if the format isn't recognized, RT_SUCCESS oth
      erwise.
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 stri
      ng
00061   * errorSize                    [out] size required to hold error string
00062   */
00063   RTresult RTAPI rtuCUDACompileString( const char* source, const char** preproces
      sorArguments, 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 stri
      ng
00073   * errorSize                    [out] size required to hold error string
00074   */
00075   RTresult RTAPI rtuCUDACompileFile( const char* filename, const char** preproces
      sorArguments, 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, unsigne
      d int* index );
00101  RTresult rtuGroupAddChild        ( RTgroup group, RTselector      child, unsigne
      d int* index );
00102  RTresult rtuGroupAddChild        ( RTgroup group, RTtransform     child, unsigne
      d int* index );
00103  RTresult rtuGroupAddChild        ( RTgroup group, RTgeometrygroup child, unsigne
      d int* index );
```

```
00104  RTresult rtuSelectorAddChild     ( RTselector selector, RTgroup         child, u
       nsigned int* index );
00105  RTresult rtuSelectorAddChild     ( RTselector selector, RTselector      child, u
       nsigned int* index );
00106  RTresult rtuSelectorAddChild     ( RTselector selector, RTtransform     child, u
       nsigned int* index );
00107  RTresult rtuSelectorAddChild     ( RTselector selector, RTgeometrygroup child, u
       nsigned int* index );
00108 #endif
00109  RTresult rtuGeometryGroupAddChild( RTgeometrygroup geometrygroup, RTgeometryinst
       ance 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 on
       e.
00127   */
00128  RTresult rtuGroupRemoveChild        ( RTgroup group, RTobject child );
00129  RTresult rtuSelectorRemoveChild     ( RTselector selector, RTobject child );
00130  RTresult rtuGeometryGroupRemoveChild( RTgeometrygroup geometrygroup, RTgeometryi
       nstance 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 delete
       d
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, uns
       igned int index );
00139  RTU_INLINE RTresult rtuGeometryGroupRemoveChildByIndex( RTgeometrygroup geometry
       group, unsigned int index );
00140
00141  /*
00142   * Use a linear search to find the child in the child array, and return its inde
       x.
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 in
      t* index )
00181  {
00182    RTU_GROUP_ADD_CHILD( group, child, index );
00183  }
00184
00185  RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTobject child, un
      signed 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, unsi
      gned int* index )
00208  {
00209    RTU_GROUP_ADD_CHILD( group, child, index );
00210  }
00211
```

```
00212  RTU_INLINE RTresult rtuSelectorAddChild( RTselector selector, RTgroup child, uns
       igned 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 ch
       ild, 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 chil
       d )
00254  {
00255    RTU_CHECK_ERROR( rtTransformSetChild( transform, child ) );
00256    return RT_SUCCESS;
00257  }
00258
00259  RTU_INLINE RTresult rtuTransformSetChild( RTtransform transform, RTtransform chi
       ld )
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, RTg
      eometryinstance 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 inde
      x )
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 geometry
```

```
      group, 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, RTobject child, unsigne
      d int* index)
00341  {
00342    unsigned int count;
00343    RTobject 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, RTobject chil
      d, unsigned int* index )
00354  {
00355    unsigned int count;
00356    RTobject 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

A simple API for performing raytracing queries using OptiX or the CPU. `#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 113 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_INITOPTION_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 86 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 104 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 93 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.

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 46 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 55 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 67 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:

→ ***traversal*** Return pointer for traverse state handle

query_type Ray query type

ray_format Ray format

tri_format Triangle format

outputs OR'ed mask of requested RTUoutputs

options Bit vector of or'ed RTUinitoptions.

context 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:

traversal 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:

traversal Traversal state handle

→ ***data*** 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:

traversal Traversal state handle

→ ***data_size*** 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:

traversal Traversal state handle. Can be NULL.

code Error code from last error

→ ***return_string*** 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:

traversal Traversal state handle

which Output type to be specified

→ ***output*** 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:

traversal Traversal state handle

num_rays Number of rays to be traced

rays 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:

traversal Traversal state handle

→ ***results*** 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:

traversal 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:

traversal Traversal state handle

data Acceleration data

data_size 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:

traversal Traversal state handle

num_verts Vertex count

verts Vertices [v1_x, v1_y, v1_z, v2.x, ...]

num_tris Triangle count

indices Indices [tri1_index1, tr1_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:

traversal Traversal state handle

option The option to be set

value 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:

traversal Traversal state handle

num_tris Triangle count

tris Triangles [tri1_v1.x, tri1_v1.y, tr1_v1.z, tri1_v2.x, ...]

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:

traversal 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
00046   typedef enum {
00047     RTU_QUERY_TYPE_ANY_HIT = 0,
00048     RTU_QUERY_TYPE_CLOSEST_HIT,
00049     RTU_QUERY_TYPE_COUNT
00050   } RTUquerytype;
00051
00055   typedef enum  {
00056     RTU_RAYFORMAT_ORIGIN_DIRECTION_TMIN_TMAX_INTERLEAVED = 0,
00057     RTU_RAYFORMAT_ORIGIN_DIRECTION_INTERLEAVED,
00058     RTU_RAYFORMAT_COUNT
00059   } RTUrayformat;
00060
00067   typedef enum  {
00068     RTU_TRIFORMAT_MESH= 0,
00069     RTU_TRIFORMAT_TRIANGLE_SOUP,
00070     RTU_TRIFORMAT_COUNT
00071   } RTUtriformat;
00072
00086   typedef enum  {
00087     RTU_INITOPTION_NONE          = 0,
00088     RTU_INITOPTION_GPU_ONLY      = 1 << 0,
00089     RTU_INITOPTION_CPU_ONLY      = 1 << 1,
00090     RTU_INITOPTION_CULL_BACKFACE = 1 << 2
00091   } RTUinitoptions;
00092
00093   typedef enum  {
00094     RTU_OUTPUT_NONE        = 0,
00095     RTU_OUTPUT_NORMAL      = 1 << 0, /*< float3 [x, y, z]                      */

00096     RTU_OUTPUT_BARYCENTRIC = 1 << 1, /*< float2 [alpha, beta] (gamma implicit) */

00097     RTU_OUTPUT_BACKFACING  = 1 << 2  /*< char   [1 | 0]                       */

00098   } RTUoutput;
00099
00104   typedef enum  {
00105     RTU_OPTION_INT_NUM_THREADS=0
00106   } RTUoption;
00107
00108
00113   typedef struct RTUtraversal_api*  RTUtraversal;
00114
```

```
00115
00130   RTresult RTAPI rtuTraversalCreate( RTUtraversal* traversal,
00131                                      RTUquerytype  query_type,
00132                                      RTUrayformat  ray_format,
00133                                      RTUtriformat  tri_format,
00134                                      unsigned int  outputs,
00135                                      unsigned int  options,
00136                                      RTcontext     context );
00137
00147   RTresult RTAPI rtuTraversalGetErrorString( RTUtraversal traversal,
00148                                              RTresult code,
00149                                              const char** return_string);
00158   RTresult RTAPI rtuTraversalSetOption( RTUtraversal traversal,
00159                                         RTUoption    option,
00160                                         void*        value );
00161
00177   RTresult RTAPI rtuTraversalSetMesh( RTUtraversal    traversal,
00178                                       unsigned int    num_verts,
00179                                       const float*    verts,
00180                                       unsigned int    num_tris,
00181                                       const unsigned* indices );
00182
00197   RTresult RTAPI rtuTraversalSetTriangles( RTUtraversal traversal,
00198                                            unsigned int num_tris,
00199                                            const float* tris );
00200
00209   RTresult RTAPI rtuTraversalSetAccelData( RTUtraversal traversal,
00210                                            const void*  data,
00211                                            RTsize       data_size );
00212
00220   RTresult RTAPI rtuTraversalGetAccelDataSize( RTUtraversal traversal,
00221                                                RTsize*      data_size );
00222
00231   RTresult RTAPI rtuTraversalGetAccelData( RTUtraversal traversal,
00232                                            void*        data );
00233
00246   RTresult RTAPI rtuTraversalMapRays( RTUtraversal traversal,
00247                                       unsigned int num_rays,
00248                                       float** rays );
00249
00253   RTresult RTAPI rtuTraversalUnmapRays( RTUtraversal traversal );
00254
00262   RTresult RTAPI rtuTraversalPreprocess( RTUtraversal traversal );
00263
00270   RTresult RTAPI rtuTraversalTraverse( RTUtraversal traversal );
00271
00282   RTresult RTAPI rtuTraversalMapResults( RTUtraversal         traversal,
00283                                          RTUtraversalresult** results );
00284
00288   RTresult RTAPI rtuTraversalUnmapResults( RTUtraversal         traversal );
00289
00306   RTresult RTAPI rtuTraversalMapOutput( RTUtraversal traversal,
00307                                         RTUoutput    which,
00308                                         void**       output );
00312   RTresult RTAPI rtuTraversalUnmapOutput( RTUtraversal traversal,
00313                                           RTUoutput    which );
00321   RTresult RTAPI rtuTraversalDestroy( RTUtraversal traversal );
00322
00323 #ifdef __cplusplus
00324 } /* extern "C" */
00325 #endif
00326
00327 #endif /* _optixu_optux_traversal.h */
00328
```

Index