



Denglin Hamming™ V2

dINNE API

DL-DG/SW-031A-02

2024-05-27

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼栋5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

Change History

Version	Change description
02	Upgrade.
01	Initial version

CONTENTS

1	Library API	3
1.1	Class Hierarchy	3
1.2	File Hierarchy	4
1.3	Full API	4

CHAPTER ONE

LIBRARY API

1.1 Class Hierarchy

- *Namespace dl*
 - *Namespace dl::nne*
 - * *Struct BuilderConfig*
 - * *Struct DynamicPluginTensorDesc*
 - * *Struct PluginField*
 - * *Struct PluginFieldCollection*
 - * *Struct PluginTensorDesc*
 - * *Class Builder*
 - * *Class Dims*
 - * *Class Dims2*
 - * *Class Dims3*
 - * *Class Dims4*
 - * *Class Engine*
 - * *Class ErrorRecorder*
 - * *Class ExecutionContext*
 - * *Class HostMemory*
 - * *Class Network*
 - * *Class OptimizationProfile*
 - * *Class Parser*
 - * *Class Plugin*
 - * *Class PluginCreator*
 - * *Class PluginExt*
 - * *Template Class PluginReg*
 - * *Class PluginRegistry*
 - * *Class PluginV2DynamicExt*
 - * *Enum BuilderFlag*

- * Enum ClusterConfig
- * Enum DataType
- * Enum ErrorCode
- * Enum Format
- * Enum GpuTarget
- * Enum ModelFormat
- * Enum NetworkDefinitionCreationFlag
- * Enum OptProfileSelector
- * Enum OptProfileType
- * Enum TensorIOMode
- * Enum WeightShareMode

1.2 File Hierarchy

- file_dlnne.h

1.3 Full API

1.3.1 Namespaces

Namespace dl

Contents

- Namespaces

Namespaces

- Namespace dl::nne

Namespace dl::nne

Contents

- Classes
- Enums
- Functions

Classes

- *Struct BuilderConfig*
- *Struct DynamicPluginTensorDesc*
- *Struct PluginField*
- *Struct PluginFieldCollection*
- *Struct PluginTensorDesc*
- *Class Builder*
- *Class Dims*
- *Class Dims2*
- *Class Dims3*
- *Class Dims4*
- *Class Engine*
- *Class ErrorRecorder*
- *Class ExecutionContext*
- *Class HostMemory*
- *Class Network*
- *Class OptimizationProfile*
- *Class Parser*
- *Class Plugin*
- *Class PluginCreator*
- *Class PluginExt*
- *Template Class PluginReg*
- *Class PluginRegistry*
- *Class PluginV2DynamicExt*

Enums

- *Enum BuilderFlag*
- *Enum ClusterConfig*
- *Enum DataType*
- *Enum ErrorCode*
- *Enum Format*
- *Enum GpuTarget*
- *Enum ModelFormat*
- *Enum NetworkDefinitionCreationFlag*
- *Enum OptProfileSelector*

- *Enum OptProfileType*
- *Enum TensorIOMode*
- *Enum WeightShareMode*

Functions

- *Function dl::nne::CreateInferBuilder*
- *Function dl::nne::CreateParser*
- *Function dl::nne::Deserialize*
- *Function dl::nne::GetPluginRegistry*

1.3.2 Classes and Structs

Struct BuilderConfig

- Defined in file_dlnne.h

Struct Documentation

struct dl::nne::BuilderConfig

BuilderConfig contains different configuration options. It can configure builder max batch size, weight share mode, and BuildModulator callback. It can also set whether to dump the dot file of the network and Relay IR.

Public Members

```
int max_batch_size = {1}
WeightShareMode ws_mode = {kSingle}
IBuildModulator *callback = {nullptr}
bool dump_dot = {false}
    DEPRECATED.
bool dump_ir = {false}
    DEPRECATED.
bool print_profiling = {false}
uint32_t flags = {}
GpuTarget target = {kCurrentGpu}
    DEPRECATED.
```


Struct DynamicPluginTensorDesc

- Defined in file_dlnne.h

Struct Documentation

struct dl::nne::DynamicPluginTensorDesc

Experimental.

Parameters

- *PluginTensorDesc*: Information required to interpret a pointer to tensor data, except that desc.dims has -1 in place of any runtime dimension.
- min: Upper bounds on tensor's dimensions.
- max: Lower bounds on tensor's dimensions.

Public Members

PluginTensorDesc desc

Dims min

Dims max

Struct PluginField

- Defined in file_dlnne.h

Struct Documentation

struct dl::nne::PluginField

Structure containing plugin attribute field names and associated data. This information can be parsed to decode necessary plugin metadata.

Public Functions

PluginField (const char *name_ = nullptr, const void *data_ = nullptr, const DataType type_ = DataType::kUNKNOWN_TYPE, int32_t length_ = 0)

Public Members

const char *name = {nullptr}
Plugin field attribute name.

const void *data = {nullptr}
Plugin field attribute data.

DataType type = {kUNKNOWN_TYPE}
Plugin field attribute type.

```
int32_t length = {0}
    Number of data entries in the plugin attribute.
```

Struct PluginFieldCollection

- Defined in file_dlnne.h

Struct Documentation

```
struct dl::nne::PluginFieldCollection
```

Parameters

- nbFields: Number of pluginField entries
- fields: Pointer to pluginField entries

Public Members

```
int nbFields
```

```
const PluginField *fields
```

Struct PluginTensorDesc

- Defined in file_dlnne.h

Struct Documentation

```
struct dl::nne::PluginTensorDesc
```

Parameters

- dims: Tensor dimensions
- type: Tensor data type
- format: Tensor format
- strides: Tensor strides

Public Members

```
Dims dims
```

```
DataType type
```

```
Format format
```

```
uint64_t *strides
```

Class Builder

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::Builder`

Build an engine from network definition.

Public Functions

Network ***CreateNetwork** () = 0

Create a network definition object.

Return A new network object

void **SetMaxBatchSize** (int *batch_size*) = 0

Set the maximum batch size.

Parameters

- *batch_size*: The maximum batch size can be used at execution time

int **GetMaxBatchSize** () **const** = 0

Get the maximum batch size.

Return The maximum batch size

Engine ***BuildEngine** (*Network* &*network*) = 0

Build *Engine*.

Return An engine from network definition

Parameters

- *network*: *Network* object

void **Reset** () = 0

Reset builder states (maxBatchSize, workspace, ...) to default.

void **Destroy** () = 0

Destroy this object.

void **SetErrorRecorder** (*ErrorRecorder* **recorder*) = 0

Set the *ErrorRecorder* for this interface.

Parameters

- *recorder*: The error recorder to register with this interface

ErrorRecorder ***GetErrorRecorder** () **const** = 0

Get the *ErrorRecorder* assigned to this interface.

Return A pointer to the *ErrorRecorder* object that has been registered.

void **SetBuilderConfig** (*BuilderConfig* builder_config) = 0
Set builder configurations except *flags*

Parameters

- builder_config: The configuration of the builder

void **SetFlag** (BuilderFlag *flag*) = 0
Set a single build mode flag.

Parameters

- flag: The flag added to the already enabled flags

void **SetFlags** (uint32_t *flags*) = 0
Set the build mode flags to turn on builder options for this network.
This function will override the previous flags.

Parameters

- flags: The flags are listed in the BuilderFlag enum.

bool **GetFlag** (BuilderFlag *flag*) = 0
Return true if the build mode flag is set.

Return true if *flag* is set, false if unset

Parameters

- flag: The flag to be queried

uint32_t **GetFlags** () = 0
Get the build mode flags for this builder config. The default value is 0.

Return The build options as a bitmask

Network ***CreateNetworkV2** (uint32_t *flags*) = 0
Create a new network definition object.

Return A new network object

Parameters

- flags: Bitset of NetworkDefinitionCreationFlags specifying network properties combined with bitwise

OptimizationProfile ***CreateOptimizationProfile** () = 0
Create a new optimization profile.

Return A new optimization object

HostMemory ***BuildSerializedNetwork** (*Network* &network) = 0
Build and serialize a network for the given *Network*. This function allows building and serialization of a network without creating an engine.

Return A pointer to a *HostMemory* object that contains a serialized engine.

Parameters

- network: *Network* definition

Protected Functions

`~Builder()`

Class Dims

- Defined in file_dlnne.h

Inheritance Relationships

Derived Types

- `public dl::nne::Dims2 (Class Dims2)`
- `public dl::nne::Dims3 (Class Dims3)`
- `public dl::nne::Dims4 (Class Dims4)`

Class Documentation

class `dl::nne::Dims`

Structure to define the dimensions of a tensor.

Subclassed by `dl::nne::Dims2`, `dl::nne::Dims3`, `dl::nne::Dims4`

Public Members

`int nbDims = {-1}`

The number of dimensions.

`int d[MAX_DIMS] = {}`

The extent of each dimension.

Public Static Attributes

`const int MAX_DIMS = 8`

The maximum number of dimensions supported for a tensor.

Class Dims2

- Defined in file_dlnne.h

Inheritance Relationships

Base Type

- `public dl::nne::Dims (Class Dims)`

Class Documentation

```
class dl::nne::Dims2 : public dl::nne::Dims
```

Public Functions

Dims2 ()

Construct an empty *Dims2* object.

Dims2 (int d0, int d1)

Construct a *Dims2* form elements.

Parameters

- d0: The first element.
- d1: The second element.

Class Dims3

- Defined in file_dlnne.h

Inheritance Relationships

Base Type

- `public dl::nne::Dims (Class Dims)`

Class Documentation

```
class dl::nne::Dims3 : public dl::nne::Dims
```

Public Functions

Dims3 ()

Construct an empty *Dims3* object.

Dims3 (int d0, int d1, int d2)

Construct a *Dims3* form elements.

Parameters

- d0: The first element.
- d1: The second element.

- d2: The third element.

Class Dims4

- Defined in file_dlnne.h

Inheritance Relationships

Base Type

- `public dl::nne::Dims (Class Dims)`

Class Documentation

```
class dl::nne::Dims4 : public dl::nne::Dims
```

Public Functions

Dims4 ()

Construct an empty *Dims4* object.

Dims4 (int *d0*, int *d1*, int *d2*, int *d3*)

Construct a *Dims4* form elements.

Parameters

- d0: The first element.
- d1: The second element.
- d2: The third element.
- d3: The fourth element.

Class Engine

- Defined in file_dlnne.h

Class Documentation

```
class dl::nne::Engine
```

An engine for executing inference on a built network, with functionally unsafe features.

Public Functions

int **GetNbBindings** () **const** = 0

Get the number of binding indices.

Return return the total bindings over all profiles

bool **BindingIsInput** (int *binding_index*) **const** = 0

Determine whether a binding is an input binding.

Return True if the index corresponds to an input binding and the index is in range.

Parameters

- *binding_index*: The binding index.

int **GetBindingIndex** (**const** char **name*) **const** = 0

Retrieve the binding index for a named tensor.

Return The binding index for the named tensor, or -1 if the name is not found.

Parameters

- *name*: The tensor name.

const char ***GetBindingName** (int *binding_index*) **const** = 0

Retrieve the name corresponding to a binding index.

Return The name corresponding to the index, or nullptr if the index is out of range.

Parameters

- *binding_index*: The binding index.

DataType **GetBindingDataType** (int *binding_index*) **const** = 0

Determine the required data type for a buffer from its binding index.

Return The type of the data in the buffer.

Parameters

- *binding_index*: The binding index.

Dims **GetBindingDimensions** (int *binding_index*) **const** = 0

Get the dimensions of a binding.

Return The dimensions of the binding if the index is in range, otherwise Dims(). Has -1 for any dimension with a dynamic value.

Parameters

- *binding_index*: The binding index.

int **GetMaxBatchSize** () **const** = 0

Get the maximum batch size which can be used for inference.

HostMemory ***Serialize** () **const** = 0

Serialize the network to a stream.

Return A *HostMemory* object that contains the serialized engine.

ExecutionContext ***CreateExecutionContext** (ClusterConfig *config* = kCluster0) = 0
Create an execution context.

See *ExecutionContext*.

ExecutionContext ***CreateExecutionContextWithoutDeviceMemory** (ClusterConfig *config* = kCluster0) = 0
Create an execution context without device memory allocated. The memory for execution context must be supplied by the application.

See *ExecutionContext*.

void **Destroy** () = 0
Destroy this object.

const char ***GetName** () const = 0
Return the name of the network associated with the engine.
The name is set during network creation and is retrieved after building or deserialization.
Return A zero delimited C-style string representing the name of the network.

void **SetErrorRecorder** (*ErrorRecorder* **recorder*) = 0
Set the *ErrorRecorder* for this interface.

Parameters

- *recorder*: The error recorder to register with this interface

ErrorRecorder ***GetErrorRecorder** () const = 0
Get the *ErrorRecorder* assigned to this interface.

Return A pointer to the *ErrorRecorder* object that has been registered.

void **Shrink** () = 0
Release the resources allocated by tensor compiler. Thus, UploadConst and CreateExecutionContext cannot be called anymore after calling this function.

size_t **GetPerClusterDeviceMemorySize** () = 0
DEPRECATED. Please use *GetDeviceMemorySize()* instead.

uint64_t **GetDeviceMemorySize** () = 0
Query the device memory size required by the execution context.

Return Total device memory size is sufficient for the max batch size

size_t **GetPerClusterConstMemorySize** () = 0
DEPRECATED. Please use *GetConstInfoEXT()* instead.

bool **SetConstMemory** (void ***mems*) = 0
DEPRECATED. Please use *SetConstMemoryEXT()* instead.

bool **UploadConst** (void ***mems*) = 0
DEPRECATED. Please use *UploadConstEXT()* instead.

bool **GetConstInfoEXT** (uint32_t *numOfConst*, uint64_t **hashKey*, uint64_t **memSize*, uint32_t **numOfConstRet*) = 0

Query all const buffer info.

Return False if numOfConstRet is null and hashKey or size is null False if numOfConst is less than *numOfConstRet Otherwise true

Parameters

- *numOfConst*: must bigger than *numOfConstRet or equal
- *hashKey*: A hashKey array to load per-const mem hashKey
- *memSize*: A size array to load per-const mem size
- *numOfConstRet*: A ptr to total const num

bool **SetConstMemoryEXT** (uint64_t *hashKey*, void **memPtr*) = 0

Set per-const const memory.

Return False if hashKey cannot be found in this engine, otherwise true.

Parameters

- *hashKey*: hash key of the const
- *memPtr*: constant memory ptr

bool **UploadConstEXT** (uint64_t *hashKey*, void **memPtr*) = 0

Upload const mem to device.

Return False if hashKey cannot be found in this engine, otherwise true.

Parameters

- *hashKey*: hash key of the const
- *memPtr*: constant memory ptr

HostMemory ***SerializeWithoutConstEXT** (uint64_t **hashKeyList*, uint32_t *size*) = 0

Serialize engine without const in hashKeyList.

Return buffer A pointer to a buffer to serialize data.

Parameters

- *hashKeyList*: hash key list of the const not serialize
- *size*: size of hashKeyList

int32_t **GetNbOptimizationProfiles** () **const** = 0

Get the number of optimization profiles define for this engine.

Return Number of optimization profiles. It is always at least 1

Dims **GetProfileDimensions** (int32_t *bindingIndex*, int32_t *profileIndex*, OptProfileSelector *select*) **const** = 0

Get the minimum/optimum/maximum dimensions for a particular binding under an oprimization profile.

Return The minimum/optimum/maximum dimensions for this binding in this profile. If the profileIndex or bindingIndex are invalid, return *Dims* with nbDims=-1

Parameters

- `bindingIndex`: The binding index, which must belong to the given profile, or must between 0 and `bindingsPerProfile - 1` as described below
- `profileIndex`: The profile index, which must be between 0 and `GetNbOptimizationProfiles() - 1`
- `select`: Whether to query the minimum, optimum, or maximum dimensions for this binding

bool **HasImplicitBatchDimension** () **const** = 0

Query whether the engine was built with an implicit batch dimension.

Return True if tensors have implicit batch dimension, false otherwise

DataType **GetTensorDataType** (const char **tensor_name*) **const** = 0

Experimental.

Determine the required data type for a buffer from its tensor name.

Return The type of the data in the buffer. `DataType::kUNKNOWN_TYPE` will be returned if (1) `tensor_name` is not the name of an input or output tensor, or (2) `tensor_name` is nullptr

Parameters

- `tensor_name`: The name of an input or output tensor.

const char ***GetIOTensorName** (int32_t *index*) **const** = 0

Experimental.

Return the name of an IO tensor.

Return The name of an IO tensor. nullptr will be returned if the index does not fall between 0 and `getNbIOTensors()-1`.

Parameters

- `index`: The value that falls between 0 and `getNbIOTensors()-1`.

int32_t **GetNbIOTensors** () **const** = 0

Experimental.

Return Return the number of input and output tensors for the network from which the engine was built.

TensorIOMode **GetTensorIOMode** (const char **tensor_name*) **const** = 0

Experimental.

Determine whether a tensor is an input or output tensor.

Return `kIOModeInput` if `tensor_name` is an input, `kIOModeOutput` if `tensor_name` is an output, or `kIOModeNone` if neither.

Parameters

- `tensor_name`: The name of an input or output tensor.

Dims **GetTensorShape** (const char **tensor_name*) **const** = 0

Experimental.

Get extent of an input or output tensor.

Return Extent of the tensor. `Dims{-1, {}}` will be returned if (1) name is not the name of an input or output tensor, or (2) name is nullptr

Parameters

- `tensor_name`: The name of an input or output tensor.

Dims **GetProfileShape** (**const** char **tensor_name*, int32_t *profile_index*, OptProfileSelector *select*) **const** = 0

Experimental.

Get the minimum / optimum / maximum dimensions for an input tensor given its name under an optimization profile.

Return The minimum / optimum / maximum dimensions for an input tensor in this profile. If the `profile_index` is invalid or provided name does not map to an input tensor, return `Dims{-1, {}}`

Parameters

- `tensor_name`: The name of an input or output tensor.
- `profile_index`: The profile index, which must be between 0 and `GetNbOptimizationProfiles()-1`.
- `select`: Whether to query the minimum, optimum, or maximum dimensions for this input tensor.

~Engine ()

Class ErrorRecorder

- Defined in `file_dlnne.h`

Class Documentation

class `dl::nne::ErrorRecorder`

Reference counted application-implemented error reporting interface for dlnne object.

Public Types

using `ErrorDesc` = **const** char*

A typedef of a c-style string for reporting error descriptions.

using `RefCount` = int32_t

A typedef of a 32bit integer for reference counting.

Public Functions

int32_t **GetNbErrors** () **const** = 0

Return the number of errors.

Return Return the number of errors detected, or 0 if there is no error.

ErrorCode **GetErrorCode** (int32_t *errorIdx*) **const** = 0

Return the ErrorCode enumeration.

Return Return the enum corresponding to `errorIdx`.

Parameters

- `errorIdx`: A 32bit integer that is indexed into the error array.

ErrorDesc **GetErrorDesc** (int32_t *errorIdx*) **const** = 0

Return the c-style string description of the error.

Return Return a string representation of the error along with a description of the error. For the error specified by the `idx` value, return the string description of the error. The error string is a c-style string that is zero delimited. In the safety context, there is a constant length requirement to remove any dynamic memory allocations and the error message may be truncated. The format of the string is "<EnumAsStr> - <Description>".

Parameters

- `errorIdx`: A 32bit integer that is indexed into the error array.

bool **HasOverflowed** () **const** = 0

Determine if the error stack has overflowed.

Return True if errors have been dropped due to overflowing the error stack.

void **Clear** () = 0

Clear the error stack on the error recorder. Remove all the tracked errors by the error recorder. This function must guarantee that after this function is called, and as long as no error occurs, the next call to `getNbErrors` will return zero.

bool **ReportError** (ErrorCode *val*, *ErrorDesc* *desc*) = 0

Report an error to the error recorder with the corresponding enum and description.

Return True if the error is determined to be fatal and processing of the current function must end.

Parameters

- `val`: The error code enum that is being reported.
- `desc`: The string description of the error.

RefCount **IncRefCount** () = 0

Increment the refcount for the current *ErrorRecorder*.

Return The current reference counted value.

RefCount **DecRefCount** () = 0

Decrement the refcount for the current *ErrorRecorder*. Decrement the reference count for the object by one and return the current value. It is undefined behavior to call `decRefCount` when `RefCount` is zero. If the *ErrorRecorder* is destroyed before the reference count hits zero, then behavior in dINNE is undefined. It is strongly recommended that the decrement is an atomic operation. dINNE guarantees that each `decRefCount` called when an object is destructed is paired with a `incRefCount` call when that object was constructed.

Return The current reference value

Protected Functions

`~ErrorRecorder()`

Class ExecutionContext

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::ExecutionContext`

Context for executing inference using an engine.

Multiple execution contexts may exist for one *Engine*, allowing the same engine to be executed with various arguments.

Public Functions

bool Execute (int *batch_size*, void ***bindings*) = 0

Synchronously execute inference on a batch.

This method requires an array of input and output buffers.

Return True if execution succeeded.

Parameters

- *batch_size*: The batch size. This is at most the value supplied when the engine was built.
- *bindings*: An array of pointers to input and output buffers.

bool ExecuteV2 (void ***bindings*) = 0

Synchronously execute inference on a batch.

This method requires an array of input and output buffers.

Return True if execution succeeded.

Parameters

- *bindings*: An array of pointers to input and output buffers.

bool Enqueue (int *batch_size*, void ***bindings*, void **stream*, void **input_consumed*) = 0

Asynchronously execute inference on a batch.

This method requires an array of input and output buffers.

Parameters

- *batch_size*: The batch size. This is at most the value supplied when the engine was built.
- *bindings*: An array of pointers to input and output buffers.
- *stream*: A cuda stream on which the inference kernels will be enqueued.
- *input_consumed*: An optional event pointer which will be signalled when the input buffers can be refilled with new data

bool EnqueueV2 (void ***bindings*, void **stream*, void **input_consumed*) = 0

Asynchronously execute inference.

This method requires an array of input and output buffers.

Parameters

- **bindings**: An array of pointers to input and output buffers.
- **stream**: A cuda stream on which the inference kernels will be enqueued.
- **input_consumed**: An optional event pointer which will be signalled when the input buffers can be refilled with new data

void **SetDebugSync** (bool *sync*) = 0

Experimental.

Set the debug sync flag.

Parameters

- **sync**: If this flag is set to true, engine will log the successful execution for each kernel during `execute()`.

bool **GetDebugSync** () **const** = 0

Experimental.

Get the debug sync flag.

const Engine *GetEngine () **const** = 0

Get the associated engine.

void **Destroy** () = 0

Destroy this object.

void **SetName** (**const** char **name*) = 0

Set the name of the execution context.

Parameters

- **name**: The name of the execution context.

const char ***GetName** () **const** = 0

Return the name of the execution context.

bool **SetBindingDimensions** (int *binding_index*, *Dims* *dimensions*) = 0

Set the dynamic dimensions of a binding.

Requires the engine to be built without an implicit batch dimension. The binding must be an input tensor, and all dimensions must be compatible with the network definition (that is, only the wildcard dimension -1 can be replaced with a new dimension > 0). Furthermore, the dimensions must be in the valid range for the currently selected optimization profile, and the corresponding engine must not be safety-certified. This method will fail unless a valid optimization profile is defined for the current execution context (`getOptimizationProfile()` must not be -1).

Return False if an error occurs (for example, index out of the range), else true

Parameters

- **binding_index**: The index of the input buffers
- **dimensions**: The dimension which will set for the corresponding index

Dims **GetBindingDimensions** (int *binding_index*) **const** = 0

Get the dynamic dimensions of a binding.

If the engine was built with an implicit batch dimension, this is the same as *Engine::GetBindingDimensions*. If *SetBindingDimensions()* has been called on this binding (or if there are no dynamic dimensions), all dimensions will be positive. If the binding is output dimension and the dimension is unknown, *Dims{}* is returned. For *Engine* with dynamic bindings, it is necessary to call *SetBindingDimensions()* before *EnqueueV2()* or *ExecuteV2()* may be called. **Note:** Some operations require binding data to infer exact output dimensions. On this condition, *GetBindingDimensions* will return the max possible dimensions before *ExecuteV2()* and *EnqueueV2()*, and after them, *GetBindingDimensions()* can return the exact value.

Return Currently selected binding dimensions. *Dims{}* if unknown.

Parameters

- *binding_index*: The specified *binding_index* of the buffers

`int32_t GetOptimizationProfile() const = 0`

Get the index of the currently selected optimization profile.

If the profile index has not been set yet (implicitly to 0 for the first execution context of the engine to be created, or explicitly for all subsequent contexts), an invalid value of -1 will be returned and all calls to *Enqueue[V2]()* or *Execute[V2]()* will fail until a valid profile index has been set.

`bool SetOptimizationProfile(int32_t profile_index) = 0`

Select an optimization profile for the current context.

The selected profile will be used in subsequent calls to *Enqueue[V2]()* or *Execute[V2]()*

Parameters

- *profile_index*: Index of the profile. It must lie between range [0, *GetEngine().GetNbOptimizationProfiles()*)

`void SetErrorRecorder(ErrorRecorder *recorder) = 0`

Set the *ErrorRecorder* for this interface.

Parameters

- *recorder*: The error recorder to register with this interface

`ErrorRecorder *GetErrorRecorder() const = 0`

Get the *ErrorRecorder* assigned to this interface.

Return A pointer to the *ErrorRecorder* object that has been registered.

`void SetDeviceMemory(void **mems) = 0`

Set the device memory for the execution context, which is allocated with cuda API.

Parameters

- *mems*: A device memory array with the same length as the number of clusters

`bool SetInputShape(const char *tensor_name, const Dims &dims) = 0`

Experimental.

Set shape of given input.

Return True on success, false if the provided name does not map to an input tensor, or if some other error occurred. Each dimension must agree with the network dimension unless the latter was -1.

Parameters

- `tensor_name`: The name of an input tensor.
- `dims`: The shape of an input tensor.

Dims `GetTensorShape (const char *tensor_name) const = 0`

Experimental.

Return the shape of the given input or output.

Return `Dims{-1, {}}` if the provided name does not map to an input or output tensor. Otherwise return the shape of the input or output tensor. An output tensor may also have -1 wildcard dimensions if its shape is currently unknown.

Parameters

- `tensor_name`: The name of an input or output tensor.

bool `SetTensorAddress (const char *tensor_name, void *data) = 0`

Experimental.

Set memory address for given input or output tensor.

Return True on success, false if error occurred. An address defaults to nullptr. Pass `data=nullptr` to reset to the default state. Return false if the provided name does not map to an input or output tensor.

Parameters

- `tensor_name`: The name of an input or output tensor.
- `data`: The pointer (void*) to the data owned by the user.

void* `GetTensorAddress (const char *tensor_name) const = 0`

Experimental.

Get memory address bound to given input or output tensor.

Return The pointer (void*) to the data of a tensor with name `tensor_name` or nullptr if the provided name does not map to an input or output tensor.

Parameters

- `tensor_name`: The name of an input or output tensor.

int32_t `InferShapes (int32_t nb_max_names, const char **tensor_names) = 0`

Experimental.

Run shape calculations.

Return 0 on success. Positive value `n` if `n` input tensors were not sufficiently specified. -1 for other errors.

Parameters

- `nb_max_names`: Maximum number of names to write to `tensor_names`. When the return value is a positive value `n` and `tensorNames != nullptr`, the names of `min(n,nbMaxNames)` insufficiently specified input tensors are written to `tensor_names`.
- `tensor_names`: Buffer in which to place names of insufficiently specified input tensors.

bool `SetInputConsumedEvent (void *event) = 0`

Experimental.

Set or reset an event which will be triggered when all input tensor buffers can be refilled

Return True on success, false if error occurred.

Parameters

- **event**: A cuda event that is triggered after all input tensors have been consumed. Passing `event==nullptr` removes whatever event was set.

void ***GetInputConsumedEvent** () **const** = 0
Experimental.

Get the event associated with consuming the input.

Return The cuda event. nullptr will be returned if the event is not set yet.

bool **EnqueueV3** (void *stream) = 0
Experimental.

Enqueue inference on a stream.

Return True if the kernels were enqueued successfully, false otherwise. Modifying or releasing memory that has been registered for the tensors before stream synchronization or the event passed to [SetInputConsumedEvent\(\)](#) has been being triggered results in undefined behavior. Input tensor can be released after the event passed to [SetInputConsumedEvent\(\)](#) is triggered whereas output tensors require stream synchronization.

Parameters

- **stream**: A cuda stream on which the inference kernels will be enqueued.

Protected Functions

~ExecutionContext ()

Class HostMemory

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::HostMemory`

Class to handle library allocated memory that is accessible to the user.

Public Functions

void ***Data** () **const** = 0
A pointer to the raw data that is owned by the library.

std::size_t **Size** () **const** = 0
The size in bytes of the data that was allocated.

DataType **Type** () **const** = 0
The type of the memory that was allocated.

void **Destroy** () = 0
Destroy the allocated memory.

Protected Functions

`~HostMemory()`

Class Network

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::Network`

A network definition for input to the builder.

Public Functions

`int GetNbInputs() const = 0`

Get the number of inputs in the network.

`int GetNbOutputs() const = 0`

Get the number of outputs in the network.

`void Destroy() = 0`

Destroy the network object.

`void SetName(const char *name) = 0`

Set the name of the network.

Parameters

- name: The name of the network.

`const char *GetName() const = 0`

Return the name associated with the network.

`bool HasImplicitBatchDimension() const = 0`

Query whether the network was created with an implicit batch dimension.

`int32_t AddOptimizationProfile(OptimizationProfile *profile) = 0`

Add an optimization profile.

Return The index of the optimization profile(starting from 0) if the input is valid, or -1 if the input is not valid

Parameters

- profile: The new optimization profile

`int32_t GetNbOptimizationProfiles() const = 0`

Get number of optimization profiles.

Return The number of optimization profiles

`bool SetConfig(const char *config) = 0`

Set config.

Return Return success if set success

Parameters

- `config`: The config set for network. The string config must be at most 4096 bytes

Protected Functions

`~Network()`

Class OptimizationProfile

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::OptimizationProfile`

Profile for dynamic input dimensions.

Public Functions

bool `SetDimensions` (**const** `char *input_name`, `OptProfileSelector select`, `OptProfileType type`, *`Dims`* `dims`) = 0

Set the minimum / optimum / maximum dimensions for a dynamic input tensor with name `input_name`.

Return false if an inconsistency was detected, and the params won't be set the the *`OptimizationProfile`*, otherwise true.

Parameters

- `input_name`: The input tensor name
- `select`: Whether to set the minimum, optimum, or maximum dimensions
- `type`: The input type
- `dims`: The minimum, optimum, or maximum dimensions for this input tensor

`Dims` `GetDimensions` (**const** `char *input_name`, `OptProfileSelector select`) **const** = 0

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

If the dimensions have not been previously set via `setDimensions()`, return an invalid *`Dims`* with `nbDims == -1`.

bool `IsValid` () **const** = 0

Check whether the optimization profile can be passed to an `IBuilderConfig` object. That is, all tensors set by this *`OptimizationProfile`* must have valid minimum / optimum / maximum dimension. This requires minimum, optimum and maximum have the same `nbDims`, and $0 \leq \min.d[i] \leq \text{opt}.d[i] \leq \max.d[i]$ when `i` is in range `[0, nbDims)`

bool `AddJson` (`char *file`) = 0

Add an json file.

Return false if add fail

Parameters

- `file`: The new json file

`~OptimizationProfile()`

Class Parser

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::Parser`

Class used for parsing models described using the tensorflow graph.

Public Functions

`bool Parse(const char *file, Network &network) = 0`
Parse a tensorflow graph file.

Parameters

- `file`: The name of the tensorflow graph.
- `network`: *Network* in which the parser will fill.

`bool ParseV2(const char *file, const char *subGraphs, Network &network) = 0`
Parse a tensorflow graph file.

Parameters

- `file`: The name of the tensorflow graph.
- `subGraphs`: Config for sub graph.
- `network`: *Network* in which the parser will fill.

`bool ParseBuffers(ModelFormat format, void **buffers, uint64_t *sizes, uint32_t numOfBuffers, Network &network) = 0`
Parse from memory buffers.

Parameters

- `format`: The type of the model.
- `buffers`: The buffers for parse.
- `sizes`: The size of buffer.
- `numOfBuffers`: The number of buffers.
- `network`: *Network* in which the parser will fill.

`void Destroy() = 0`
Destroy this object.

`bool RegisterInput(const char *inputName, Dims inputDims, Format inputFormat = kNHWC) = 0`
Register an input name of a tensorflow network with associated dimensions.

Parameters

- `inputName`: An output name of a tensorflow network.
- `inputDims`: Input dimensions.
- `inputFormat`: Input format. Optional

bool **RegisterOutput** (const char *outputName) = 0
 Register an output name of a tensorflow network.

Parameters

- `outputName`: An output name of a tensorflow network.

void **SetErrorRecorder** (ErrorRecorder *recorder) = 0
 Set the *ErrorRecorder* for this interface.

Parameters

- `recorder`: The error recorder to register with this interface

ErrorRecorder ***GetErrorRecorder** () const = 0
 Get the *ErrorRecorder* assigned to this interface.

Return A pointer to the *ErrorRecorder* object that has been registered.

bool **RegisterUserOp** (const char *library, const char *module, const char *firstOpName) = 0
 Register the plugin library name used in a tensorflow network.

Parameters

- `library`: The path and name of a tensorflow customer defined library
- `module`: The name and path of the python file
- `firstOpName`: The name of the first op defined by user. Optional

Protected Functions

~Parser ()

Class Plugin

- Defined in file_dlnne.h

Inheritance Relationships

Derived Types

- public dl::nne::PluginExt (Class *PluginExt*)
- public dl::nne::PluginV2DynamicExt (Class *PluginV2DynamicExt*)

Class Documentation

class dl::nne::Plugin

Plugin class for user-implemented layers. Plugins are a mechanism for applications to implement custom layers. When combined with IPluginCreator, it provides a mechanism to register plugins and look up the *Plugin* Registry during de-serialization.

Subclassed by *dl::nne::PluginExt*, *dl::nne::PluginV2DynamicExt*

Public Functions

~Plugin()

plugin virtual destructor

const char *GetPluginType() const = 0

Return the plugin type. Should match the plugin name returned by the corresponding plugin creator.

Return Return the plugin type

const char *GetPluginVersion() const = 0

Return the plugin version. Should match the plugin version returned by the corresponding plugin creator.

Return Return the plugin version

int GetNbOutputs() const = 0

Get the number of outputs from the layer.

Return Return the number of outputs.

Dims GetOutputDimensions(int index, const Dims *inputs, int nbInputDims) = 0

Get the dimension of an output tensor.

Return Return the dimension of an output tensor.

Parameters

- *index*: The index of the output tensor
- *inputs*: The input tensor
- *nbInputDims*: The number of the input tensors

bool SupportsFormat(const Dims *inputDims, int nbInputs, const Dims *outputDims, int nbOutputs, const DataType *inputTypes, const DataType *outputTypes, Format format) const = 0

Check format support.

Return Return true if the plugin supports the type-format combination.

Parameters

- *inputDims*: The input tensor dimensions.
- *nbInputs*: The number of inputs.
- *outputDims*: The output tensor dimensions.
- *nbOutputs*: The number of outputs.

- `inputTypes`: The data types for each input.
- `outputTypes`: The data types for each output.
- `format`: Format requested.

void **ConfigureWithFormat** (`const Dims *inputDims`, `int nbInputs`, `const Dims *outputDims`, `int nbOutputs`, `const DataType *inputTypes`, `const DataType *outputTypes`, `Format format`, `int maxBatchSize`) = 0

Configure the layer.

Parameters

- `inputDims`: The input tensor dimensions.
- `nbInputs`: The number of inputs.
- `outputDims`: The output tensor dimensions.
- `nbOutputs`: The number of outputs.
- `inputTypes`: The data types for each input.
- `outputTypes`: The data types for each output.
- `format`: The format selected for the engine.
- `maxBatchSize`: The maximum batch size.

int **Initialize** () = 0

Initialize the layer for execution.

Return Return 0 for success, else non-zero

void **Terminate** () = 0

Release resources acquired during plugin layer initialization.

size_t **GetWorkspaceSize** (`int maxBatchSize`) `const` = 0

Find the workspace size of the layer.

Return Return the workspace size which should be sufficient for any batch size up to the maximum

Parameters

- `maxBatchSize`: The max batch size of the network

int **Enqueue** (`int batch_size`, `const void *const *inputs`, `const void *const *input_stride`, `void **outputs`, `const void *const *output_stride`, `void *workspace`, `void *stream`) = 0

Execute the layer.

Return Return 0 for success, else non-zero

Parameters

- `batch_size`: The number of inputs in the batch.
- `inputs`: The memory for the input tensors.
- `input_stride`: The strides of the input tensors.
- `outputs`: The memory for the output tensors.
- `output_stride`: The strides of the output tensors.

- **workspace:** Workspace for execution.
- **stream:** The stream in which to execute the kernels.

```
void *GetGraph (int batch_size, const void *const *inputs, const void *const *input_stride,
               void **outputs, const void *const *output_stride, void *workspace, void
               *stream) = 0
```

Generate the execution graph of the layer, and the graph must be prepared before the execution context is created.

Return Return the pointer to the execution graph for success, else nullptr

Parameters

- **batch_size:** The number of inputs in the batch.
- **inputs:** The memory for the input tensors.
- **input_stride:** The strides of the input tensors.
- **outputs:** The memory for the output tensors.
- **output_stride:** The strides of the output tensors.
- **workspace:** Workspace for execution.
- **stream:** The stream in which to execute the kernels.

```
size_t GetSerializationSize () const = 0
```

Find the size of the serialization buffer required.

Return Return the size of the serialization buffer.

```
void Serialize (void *buffer) const = 0
```

Serialize the layer.

Parameters

- **buffer:** A pointer to a buffer to serialize data. Size of buffer must be equal to the value returned by `getSerializationSize`.

```
void Destroy () = 0
```

Destroy the plugin object. This will be called when the network, builder or engine is destroyed.

```
Plugin *Clone () const = 0
```

Clone the plugin object. This copies over internal plugin parameters and return a new plugin object with these parameters.

Return Return the plugin

```
void SetPluginNamespace (const char *pluginNamespace) = 0
```

Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

Parameters

- **pluginNamespace:** The plugin namespace

```
const char *GetPluginNamespace () const = 0
```

Get the namespace of the plugin creator object.

Return Return the namespace of the plugin creator object.

Class PluginCreator

- Defined in file_dlnne.h

Class Documentation

class `dl::nne::PluginCreator`

Plugin creator class for user implemented layers.

Public Functions

~PluginCreator ()

PluginCreator virtual destructor.

const char ***GetPluginName** () **const** = 0

Get the plugin name.

Return Return the plugin name

const *PluginFieldCollection* ***GetFieldNames** () = 0

Return a list of fields that needs to be passed to createPlugin.

Return Return a list of fields that needs to be passed to createPlugin.

const char ***GetPluginVersion** () **const** = 0

Get the plugin version.

Return Return the plugin version.

Plugin ***CreatePlugin** (**const** char **name*, **const** *PluginFieldCollection* **fc*) = 0

Return a plugin object. Return nullptr in case of error.

Return Return a plugin object.

Parameters

- *name*: The plugin name
- *fc*: The fields to fill in the plugin to be created

Plugin ***DeserializePlugin** (**const** char **name*, **const** void **serialData*, size_t *serialLength*) = 0

Return a plugin object. Return nullptr in case of error.

Return Return a plugin object.

Parameters

- *name*: The plugin name
- *serialData*: The fields used to deserialize the plugin
- *serialLength*: The data length of the buffer

void **SetPluginNamespace** (const char *pluginNamespace) = 0

Set the namespace of the plugin creator based on the plugin library it belongs to. This can be set while registering the plugin creator.

Parameters

- pluginNamespace: The plugin namespace

const char ***GetPluginNamespace** () const = 0

Get the namespace of the plugin creator object.

Return Return the namespace of the plugin creator object.

Class PluginExt

- Defined in file_dlnne.h

Inheritance Relationships

Base Type

- public dl::nne::Plugin (*Class Plugin*)

Class Documentation

```
class dl::nne::PluginExt : public dl::nne::Plugin
```

Public Functions

void **ConfigurePlugin** (*PluginTensorDesc* const *in, int nbInput, *PluginTensorDesc* const *out, int nbOutput, int maxBatchSize) = 0

Configure the layer. This function is called by the builder prior to initialize(). It provides an opportunity for the layer to make algorithm choices on the basis of *PluginTensorDesc* and the maximum batch size.

Parameters

- in: The input tensors' attributes that are used for the configuration.
- nbInput: Number of input tensors.
- out: The output tensors' attributes that are used for configuration.
- nbOutput: Number of output tensors.
- maxBatchSize: The maximum batch size.

int **Enqueue** (int batch_size, const void *const *inputs, void **outputs, void *workspace, void *stream) = 0

Execute the layer.

Return Return 0 for success, else non-zero

Parameters

- batch_size: The number of inputs in the batch.

- `inputs`: The memory for the input tensors.
- `outputs`: The memory for the output tensors.
- `workspace`: Workspace for execution.
- `stream`: The stream in which to execute the kernels.

void ***GetGraph** (int *batch_size*, **const** void ***const** **inputs*, void ****outputs**, void **workspace*, void **stream*) = 0

Generate the execution graph of the layer, and the graph must be prepared before the execution context is created.

Return Return the pointer to the execution graph for success, else nullptr

Parameters

- `batch_size`: The number of inputs in the batch.
- `inputs`: The memory for the input tensors.
- `outputs`: The memory for the output tensors.
- `workspace`: Workspace for execution.
- `stream`: The stream in which to execute the kernels.

Template Class PluginReg

- Defined in file_dlnne.h

Class Documentation

template<typename **T**, char ***const** **name_space*>
class dl::nne::PluginReg

Template class used to register plugin creator.

Parameters

- `T`: *Plugin* creator type
- `name_space`: *Plugin* namespace

Public Functions

PluginReg ()

~PluginReg ()

Class PluginRegistry

- Defined in file_dlnne.h

Class Documentation

class dl::nne::PluginRegistry

Single registration point for all plugins in an application. It is used to find plugin implementations during engine deserialization. Internally, the plugin registry is considered to be a singleton so all plugins in an application are part of the same global registry.

Public Functions

~PluginRegistry()

PluginCreator virtual destructor.

bool **RegisterCreator** (*PluginCreator* &creator, const char *pluginNamespace) = 0

Register a plugin creator.

Return Return false if one with the same type is already registered.

Parameters

- creator: The pluginCreator to register
- pluginNamespace:

PluginCreator *const **GetPluginCreatorList** (int *numCreators) const = 0

Return all the registered plugin creators and the number of registered plugin creators.

Return Return all the plugin creators

Parameters

- numCreators:

PluginCreator ***GetPluginCreator** (const char *pluginType, const char *pluginVersion, const char *pluginNamespace = "") = 0

Return plugin creator based on plugin type, version and namespace associated with plugin during network creation.

Return Return the creator based on plugin type, version and namespace

Parameters

- pluginType:
- pluginVersion:
- pluginNamespace:

void **SetErrorRecorder** (*ErrorRecorder* *recorder) = 0

Set the *ErrorRecorder* for this interface.

Parameters

- recorder: The error recorder to register with this interface.

ErrorRecorder *GetErrorRecorder () const = 0

Get the *ErrorRecorder* for this interface.

Return Return a pointer to the IErrorRecorder object that has been registered.

Class PluginV2DynamicExt

- Defined in file_dlnne.h

Inheritance Relationships

Base Type

- public dl::nne::Plugin (Class *Plugin*)

Class Documentation

class dl::nne::PluginV2DynamicExt : public dl::nne::Plugin
Experimental.

PluginV2DynamicExt class for user-implemented dynamic layers.

Public Functions

~PluginV2DynamicExt ()

PluginV2DynamicExt virtual destructor.

PluginV2DynamicExt *Clone () const = 0

Experimental.

Clone the plugin object. This copies over internal plugin

Return Return the *PluginV2DynamicExt*

void ConfigurePlugin (DynamicPluginTensorDesc const *in, int32_t nbInputs, DynamicPluginTensorDesc const *out, int32_t nbOutputs) = 0

Experimental.

Configure the plugin. An input shape binding is changed via setInputShapeBinding().

Parameters

- in: The input tensors' attributes that are used for the configuration.
- nbInputs: Number of input tensors.
- out: The output tensors' attributes that are used for configuration.
- nbOutputs: Number of output tensors.

int Enqueue (PluginTensorDesc const *inputDesc, PluginTensorDesc const *outputDesc, void const *const *inputs, void *const *outputs, void *workspace, void *stream) = 0

Experimental.

Execute the plugin

Return Return 0 for success, else non-zero

Parameters

- `inputDesc`: How to interpret the memory for the input tensors.
- `outputDesc`: How to interpret the memory for the output tensors.
- `inputs`: The memory for the input tensors.
- `outputs`: The memory for the output tensors.
- `workspace`: Workspace for execution.
- `stream`: The stream in which to execute.

`size_t GetWorkspaceSize (PluginTensorDesc const *inputs, int32_t nbInputs, PluginTensorDesc const *outputs, int32_t nbOutputs) const = 0`

Experimental.

Find the workspace size required by the layer.

Return The workspace size

Parameters

- `inputs`: How to interpret the memory for the input tensors.
- `nbInputs`: The number of input tensors.
- `outputs`: How to interpret the memory for the output tensors.
- `nbOutputs`: The number of output tensors.

`DataType GetOutputDataType (int32_t index, DataType const *inputTypes, int32_t nbInputs) const = 0`

Experimental.

Return the data type of the plugin output at the requested index.

Return The data type of the plugin output at the requested index

Parameters

- `index`: Index of output.
- `inputTypes`: The interpret of data types of the inputs.
- `nbInputs`: Number of inputs.

1.3.3 Enums

Enum BuilderFlag

- Defined in file `dlnne.h`

Enum Documentation

enum `nne::dl::BuilderFlag`

Builder options.

Values:

enumerator `kSpmAlloc` = 1
DEPRECATED.

enumerator `kUserConst` = 1 << 4
use user-allocated const memory instead

enumerator `kSlzIgnoreWeights` = 1 << 5
ignore weights in `Engine::Serialize()`

Enum ClusterConfig

- Defined in file `_dl_nne.h`

Enum Documentation

enum `nne::dl::ClusterConfig`

Weight share mode corresponds to cluster configuration. When the weight share mode is set to `kSingle`, `kCluster0`, `kCluster1`, `kCluster2` and `kCluster3` are valid cluster configuration. When the weight share mode is set to `kShare2`, `kCluster01`, `kCluster23`, `kCluster02`, `kCluster13`, `kCluster03`, `kCluster12` are valid. When the weight share mode is set to `kShare4`, `kCluster0123` is the only choice.

Values:

enumerator `kCluster0`

enumerator `kCluster1`

enumerator `kCluster2`

enumerator `kCluster3`

enumerator `kCluster01`

enumerator `kCluster23`

enumerator `kCluster02`

enumerator `kCluster13`

enumerator `kCluster03`

enumerator `kCluster12`

enumerator `kCluster0123`

Enum DataType

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::DataType

Values:

```

enumerator kFLOAT16 = 0
enumerator kFLOAT32 = 1
enumerator kFLOAT64 = 2
enumerator kINT8 = 3
enumerator kINT16 = 4
enumerator kINT32 = 5
enumerator kINT64 = 6
enumerator kUINT8 = 7
enumerator kUINT16 = 8
enumerator kUINT32 = 9
enumerator kUINT64 = 10
enumerator kBOOL = 11
enumerator kUNKNOWN_TYPE = 12

```

Enum ErrorCode

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::ErrorCode

Values:

```

enumerator kSUCCESS = 0
    Execution completed successfully.
enumerator kUNSPECIFIED_ERROR
    An error that does not fall into any other category. This error is included for forward compatibility.
enumerator kINTERNAL_ERROR
    A non-recoverable dINNE error occurred.
enumerator kINVALID_ARGUMENT
    An argument passed to the function is invalid in isolation. This is a violation of the API contract.
enumerator kINVALID_CONFIG
    An error occurred when comparing the state of an argument relative to other arguments.

```

enumerator kFAILED_ALLOCATION

An error occurred when performing an allocation of memory on the host or the device. A memory allocation error is normally fatal, but in the case where the application provided its own memory allocation routine, it is possible to increase the pool of available memory and resume execution.

enumerator kFAILED_INITIALIZATION

One, or more, of the components that dlnNE relies on did not initialize correctly. This is a system setup issue.

enumerator kFAILED_EXECUTION

An error occurred during execution that caused dlnNE to end prematurely, for example, an asynchronous error. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either an execution error or a memory error.

enumerator kFAILED_COMPUTATION

An error occurred during execution that caused the data to become corrupted, but execution finished. Examples of this error are NaN squashing or integer overflow. In a dynamic system, the data can be thrown away and the next frame can be processed or execution can be retried. This is either a data corruption error, an input error, or a range error.

enumerator kINVALID_STATE

dlnNE was put into a bad state by incorrect sequence of function calls. This can occur in situations where a service is optimistically executing networks for multiple different configurations without checking proper error configurations, and instead throwing away bad configurations caught by dlnNE. This is a violation of the API contract, but can be recoverable.

enumerator kUNSUPPORTED_STATE

An error occurred due to the network not being supported on the device due to constraints of the hardware or system. An example is running an unsafe layer in a safety certified context, or a resource requirement for the current network is greater than the capabilities of the target device. The network is otherwise correct, but the network and hardware combination is problematic. This can be recoverable.

Enum Format

- Defined in file_dlnne.h

Enum Documentation**enum nne::dl::Format**

Tensor formats in memory.

Values:

enumerator kLINEAR

enumerator kNCHW

enumerator kNHWC

Enum GpuTarget

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::GpuTarget

DEPRECATED. Please use [Network::SetConfig\(\)](#) with device-profile instead.

Values:

enumerator kCurrentGpu = 0

enumerator kGoldwasserL256 = 2

enumerator kGoldwasserL128 = 3

enumerator kGoldwasserUL64 = 4

enumerator kGoldwasserUL32 = 5

Enum ModelFormat

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::ModelFormat

Model type for parse If model is kRelayModel, only same version sdk can be used in.

Values:

enumerator kOnnxModel

enumerator kTensorflowModel

enumerator kCaffeModel

enumerator kRelayModel

Enum NetworkDefinitionCreationFlag

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::NetworkDefinitionCreationFlag

[Network](#) create options.

Values:

enumerator kEXPLICIT_BATCH = 1

Enum OptProfileSelector

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::OptProfileSelector

When setting or getting optimization profile parameters of dynamic dimensions, select whether the minimum, optimum or maximum values for these parameters are to set/get. The minimum and maximum specify the permitted range that is supported at runtime.

Values:

enumerator kOptProfileMin = 0

enumerator kOptProfileOpt = 1

enumerator kOptProfileMax = 2

Enum OptProfileType

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::OptProfileType

When setting or getting optimization profile parameters of dynamic dimensions, select tensor type.

Values:

enumerator kInput = 1

enumerator kDataDependOps = 2

Enum TensorIOMode

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::TensorIOMode

Experimental.

input or output mode of a tensor

Values:

enumerator kIOModeNone = 0
Tensor is not an input or output.

enumerator kIOModeInput
Tensor is input to the engine.

enumerator kIOModeOutput
Tensor is output to the engine.

Enum WeightShareMode

- Defined in file_dlnne.h

Enum Documentation

enum nne::dl::WeightShareMode

Share the weights on different clusters. kSingle means executing a network on single cluster without weight share. kShare2 means sharing network weights on two clusters between . kShare4 means sharing network weights on all four clusters.

Values:

enumerator kSingle = 1

enumerator kShare2 = 2

enumerator kShare4 = 4

1.3.4 Functions

Function dl::nne::CreateInferBuilder

- Defined in file_dlnne.h

Function Documentation

Builder *nne::dl::CreateInferBuilder()

Function dl::nne::CreateParser

- Defined in file_dlnne.h

Function Documentation

Parser *nne::dl::CreateParser()

Function dl::nne::Deserialize

- Defined in file_dlnne.h

Function Documentation

Engine `*nne::dl::Deserialize (const void *blob, std::size_t size)`

Function `dl::nne::GetPluginRegistry`

- Defined in `file_dlnne.h`

Function Documentation

PluginRegistry `*nne::dl::GetPluginRegistry ()`

1.3.5 Defines

Define `DRIVER_NNE_MAJOR`

- Defined in `file_dlnne.h`

Define Documentation

`DRIVER_NNE_MAJOR`

Define `DRIVER_NNE_MINOR`

- Defined in `file_dlnne.h`

Define Documentation

`DRIVER_NNE_MINOR`

Define `DRIVER_NNE_PATCH`

- Defined in `file_dlnne.h`

Define Documentation

`DRIVER_NNE_PATCH`

Define REGISTER_DLNNE_PLUGIN

- Defined in file_dlnne.h

Define Documentation

REGISTER_DLNNE_PLUGIN (*name*, *name_space*)