AETK

Generated by Doxygen 1.10.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 AE Namespace Reference

Namespace for various pre-defined STL containers using a custom allocator.

Classes

- class App
- class ColorProperty
- class CompItem

  A class for representing a composition in After Effects.
- class Item

  Represents an After Effects item.
- class Layer

  Represents a layer in After Effects.
- class LayerIDProperty
- class MarkerProperty
- class MaskIDProperty
- class MaskProperty
- class OneDProperty
- class Project

  A class representing an After Effects Project.
- class PropertyBase
- class PropertyGroup
- class Scoped_Error_Reporter

  A class that reports errors caught within its scope.
- class Scoped_Quiet_Guard
- class Scoped_Undo_Guard
- class TextDocumentProperty
- class ThreeDProperty
- class TwoDProperty

Typedefs

- template<typename T >
  using vector = std::vector<T, AEAllocator<T>>

  Alias for std::vector using AEAllocator.
- template<typename Key , typename Value , typename Comparator = std::less<Key>>
  using map

  Alias for std::map using AEAllocator.
- template<typename T >
  using list = std::list<T, AEAllocator<T>>

  Alias for std::list using AEAllocator.
- template<typename Key , typename Hash = std::hash<Key>, typename KeyEqual = std::equal_to<Key>>
  using unordered_map

  Alias for std::unordered_map using AEAllocator.

Functions

- template<typename T , typename... Args>
  std::unique_ptr< T > **make_unique** (Args &&...args)
- template<typename T , typename... Args>
  std::shared_ptr< T > make_shared (Args &&...args)

  Custom make_shared function that uses AEAllocator to create the object.

5.1.1 Detailed Description

Namespace for various pre-defined STL containers using a custom allocator.

5.1.2 Typedef Documentation

5.1.2.1 list

```
template<typename T >
using AE::list = std::list<T, AEAllocator<T>>
```

Alias for std::list using AEAllocator.

Template Parameters

T	The type of the elements in the list.

5.1.2.2 map

```
template<typename Key , typename Value , typename Comparator = std::less<Key>>
using AE::map
```

Initial value:

```
std::map<Key, Value, Comparator, AEAllocator<std::pair<const Key, Value»>
```

Alias for std::map using AEAllocator.

Template Parameters

Key	The type of the keys in the map.
Value	The type of the values in the map.
Comparator	The type of the comparator used to compare keys.

5.1.2.3 unordered_map

```
template<typename Key , typename Hash = std::hash<Key>, typename KeyEqual = std::equal_to<↩
Key>>
using AE::unordered_map
```

Initial value:

```
    std::unordered_map<Key, AEAllocator<std::pair<const Key, Hash»>
```

Alias for std::unordered_map using AEAllocator.

Template Parameters

Key	The type of the keys in the unordered map.
Hash	The type of the hash function used to hash keys.
KeyEqual	The type of the key equality function used to compare keys for equality.

5.1.2.4 vector

```
template<typename T >
using AE::vector = std::vector<T, AEAllocator<T>>
```

Alias for std::vector using AEAllocator.

Template Parameters

T	The type of the elements in the vector.

5.1.3 Function Documentation

5.1.3.1 make_shared()

```
template<typename T , typename...  Args>
std::shared_ptr< T > AE::make_shared (
            Args &&...  args )
```

Custom make_shared function that uses AEAllocator to create the object.

Template Parameters

T	The type of the object to be created.
Args	The types of the arguments to be passed to the constructor of the object.

Parameters

args	The arguments to be passed to the constructor of the object.

Returns

std::shared_ptr<T> A shared pointer to the created object.

This function creates an object of type T using the AEAllocator<T> allocator and returns a std::shared_ptr<T> to manage the object.

Chapter 6

Class Documentation

6.1 AddKeyframesInfoDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_AddKeyframesInfoH *addKeyframesInfo)

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

- AEGP/Core/Base/AEGeneral.hpp

6.2 AEAllocator< T > Class Template Reference

AEAllocator is a custom allocator that uses the After Effects' memory suites to allocate and deallocate memory.

`#include <AEAllocator.hpp>`

Public Types

- using **value_type** = T
- using **pointer** = T*
- using **size_type** = size_t

Public Member Functions

- template<typename U >
  AEAllocator (const AEAllocator< U > &) noexcept
- pointer **allocate** (size_type n)
- void **deallocate** (pointer p, size_type n)

6.2.1 Detailed Description

template<typename T>
class AEAllocator< T >

AEAllocator is a custom allocator that uses the After Effects' memory suites to allocate and deallocate memory.

Doing so allows the user to use the standard library containers with the custom allocator, and the memory will be allocated and deallocated using AE's memory, potentially improving performance and reducing memory fragmentation.

Parameters

T	The type of the elements to be allocated. \Usage The AEAllocator can be used with the standard library containers, such as std::vector, std::list, and std::map. \Example The following example demonstrates how to use the AEAllocator with a std::vector. #include <AEMemory.h>
	`template <typename T>`
	`using vector = std::vector<T, AEAllocator<T>>;`

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

- AEGP/Memory/AEAllocator.hpp

6.3 AEException Class Reference

A custom exception class derived from std::exception.

`#include <Exception.hpp>`

Public Member Functions

- AEException (const std::string &message)

  Constructs an AEException object with the given error message.
- AEException (const char *message)

  Constructs an AEException object with the given error message.
- virtual const char * what () const throw ()

  Returns the error message associated with the exception.

6.3.1 Detailed Description

A custom exception class derived from std::exception.

This class represents an exception that can be thrown in the project. It provides a way to encapsulate and propagate error messages.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AEException() [1/2]

```
AEException::AEException (
            const std::string & message )  [inline]
```

Constructs an AEException object with the given error message.

Parameters

message	The error message associated with the exception.

6.3.2.2 AEException() [2/2]

```
AEException::AEException (
            const char * message )  [inline]
```

Constructs an AEException object with the given error message.

Parameters

message	The error message associated with the exception.

6.3.3 Member Function Documentation

6.3.3.1 what()

```
virtual const char * AEException::what ( ) const throw ( )   [inline], [virtual]
```

Returns the error message associated with the exception.

Returns

const char* The error message.

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

- AEGP/Exception/Exception.hpp

6.4 AE::App Class Reference

Public Member Functions

- void Alert (std::any data) const
  Alert the user with a message.
- void * GetWindow () const
  Get the main window (HWND) for AE.
- std::string UserPluginPath ()
  Get the current project file path.
- std::string AllPluginPath ()
  Get the path to the folder containing plugins.
- std::string AppPath ()
  Get the path to the AE application.
- ColorVal BrushColor (bool useForeground)
  Get the Color of the Paint Palette.
- void SetBrushColor (ColorVal color, bool useForeground)
  Set the color of the Paint Palette.
- ColorVal CharColor ()
  Get the current color of the Character Palette, based on which is in the front.
- void CharColor (ColorVal color, bool useFill)
  Set the color of the Character Palette.

6.4.1 Member Function Documentation

6.4.1.1 Alert()

```
void AE::App::Alert (
            std::any data ) const  [inline]
```

Alert the user with a message.

Parameters

data	The message to display

6.4.1.2 AllPluginPath()

```
std::string AE::App::AllPluginPath ( )  [inline]
```

Get the path to the folder containing plugins.

Returns

Folder path as a string

6.4.1.3 AppPath()

```
std::string AE::App::AppPath ( )  [inline]
```

Get the path to the AE application.

Returns

Application path as a string

6.4.1.4 BrushColor()

```
ColorVal AE::App::BrushColor (
            bool useForeground )  [inline]
```

Get the Color of the Paint Palette.

Parameters

useForeground	• If true, get the foreground color, else get the background color

Returns

ColorVal

6.4.1.5 CharColor() [1/2]

```
ColorVal AE::App::CharColor ( )  [inline]
```

Get the current color of the Character Palette, based on which is in the front.

Returns

ColorVal

6.4.1.6 CharColor() [2/2]

```
void AE::App::CharColor (
            ColorVal color,
            bool useFill )  [inline]
```

Set the color of the Character Palette.

Parameters

color	The color to set
useFill	• If true, set the fill color, else set the stroke color

6.4.1.7 GetWindow()

```
void * AE::App::GetWindow ( ) const  [inline]
```

Get the main window (HWND) for AE.

Returns

The main window as a void pointer, cast to platform specific type

6.4.1.8 SetBrushColor()

```
void AE::App::SetBrushColor (
            ColorVal color,
            bool useForeground )  [inline]
```

Set the color of the Paint Palette.

Parameters

color	The color to set
useForeground	• If true, set the foreground color, else set the background color

6.4.1.9 UserPluginPath()

```
std::string AE::App::UserPluginPath ( )  [inline]
```

Get the current project file path.

Returns

The file path as a string

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

- AEGP/Core/App.hpp

6.5 Collection< T > Class Template Reference

A class to represent a collection of items.

```
#include <Collection.hpp>
```

Public Member Functions

- **Collection** ()=default
  Default constructor.
- Collection (std::vector< T > collection)
  Constructor that initializes the collection with the given items.
- ~**Collection** ()=default
  Destructor.
- std::vector< T > GetCollection ()
  Get the collection of items.
- void SetCollection (std::vector< T > collection)
  Set the collection of items.
- auto begin ()
  Get an iterator pointing to the beginning of the collection.
- auto end ()
  Get an iterator pointing to the end of the collection.
- virtual size_t size ()
  Get the size of the collection.
- virtual T operator[ ] (size_t index)
  Get the item at the specified index.

- virtual void append (T item)

  Append an item to the end of the collection.
- virtual void extend (std::vector< T > items)

  Extend the collection by appending multiple items.
- virtual void insert (size_t index, T item)

  Insert an item at the specified index.
- virtual void remove (T item)

  Remove the first occurrence of an item from the collection.
- virtual void pop (size_t index)

  Remove the item at the specified index.
- virtual void **clear** ()

  Clear the collection, removing all items.
- virtual size_t index (T item)

  Get the index of the first occurrence of an item in the collection.
- virtual bool contains (T item)

  Check if the collection contains a specific item.
- virtual std::vector< T > copy ()

  Create a copy of the collection.
- virtual std::vector< T > slice (int start, int end)

  Get a slice of the collection from the specified start index to the specified end index.
- virtual std::vector< T > slice (int start)

  Get a slice of the collection from the specified start index to the end of the collection.
- virtual std::vector< T > slice ()

  Get a slice of the entire collection.
- virtual void **reverse** ()

  Reverse the order of the items in the collection.
- virtual void **sort** ()

  Sort the items in the collection in ascending order.
- virtual void sort (std::function< bool(T, T)> compare)

  Sort the items in the collection using a custom comparison function.

Protected Attributes

- std::vector< T > m_collection

6.5.1 Detailed Description

template<typename T>
class Collection< T >

A class to represent a collection of items.

This class provides various methods to manipulate and perform operations on a collection of items. It supports functionalities such as appending, extending, inserting, removing, popping, clearing, reversing, sorting, and more.

Template Parameters

T	The type of items in the collection.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Collection()

```
template<typename T >
Collection< T >::Collection (
            std::vector< T > collection )  [inline]
```

Constructor that initializes the collection with the given items.

Parameters

collection	The initial collection of items.

6.5.3 Member Function Documentation

6.5.3.1 append()

```
template<typename T >
virtual void Collection< T >::append (
            T item )  [inline], [virtual]
```

Append an item to the end of the collection.

Parameters

item	The item to be appended.

6.5.3.2 begin()

```
template<typename T >
auto Collection< T >::begin ( )  [inline]
```

Get an iterator pointing to the beginning of the collection.

Returns

auto An iterator pointing to the beginning of the collection.

6.5.3.3 contains()

```
template<typename T >
virtual bool Collection< T >::contains (
            T item )  [inline], [virtual]
```

Check if the collection contains a specific item.

Parameters

item	The item to search for.

Returns

bool True if the item is found, false otherwise.

6.5.3.4 copy()

```
template<typename T >
virtual std::vector< T > Collection< T >::copy ( )  [inline], [virtual]
```

Create a copy of the collection.

Returns

std::vector<T> A copy of the collection.

6.5.3.5 end()

```
template<typename T >
auto Collection< T >::end ( )  [inline]
```

Get an iterator pointing to the end of the collection.

Returns

auto An iterator pointing to the end of the collection.

6.5.3.6 extend()

```
template<typename T >
virtual void Collection< T >::extend (
            std::vector< T > items )  [inline], [virtual]
```

Extend the collection by appending multiple items.

Parameters

items	The items to be appended.

6.5.3.7 GetCollection()

```
template<typename T >
std::vector< T > Collection< T >::GetCollection ( )  [inline]
```

Get the collection of items.

Returns

std::vector<T> The collection of items.

6.5.3.8 index()

```
template<typename T >
virtual size_t Collection< T >::index (
            T item )  [inline], [virtual]
```

Get the index of the first occurrence of an item in the collection.

Parameters

item	The item to search for.

Returns

size_t The index of the item, or -1 if not found.

6.5.3.9 insert()

```
template<typename T >
virtual void Collection< T >::insert (
            size_t index,
            T item )  [inline], [virtual]
```

Insert an item at the specified index.

Parameters

index	The index at which the item should be inserted.
item	The item to be inserted.

6.5.3.10 operator[]()

```
template<typename T >
virtual T Collection< T >::operator[] (
            size_t index )  [inline], [virtual]
```

Get the item at the specified index.

Parameters

index	The index of the item.

Returns

T The item at the specified index.

6.5.3.11 pop()

```
template<typename T >
virtual void Collection< T >::pop (
            size_t index )  [inline], [virtual]
```

Remove the item at the specified index.

Parameters

index	The index of the item to be removed.

6.5.3.12 remove()

```
template<typename T >
virtual void Collection< T >::remove (
            T item )  [inline], [virtual]
```

Remove the first occurrence of an item from the collection.

Parameters

item	The item to be removed.

6.5.3.13 SetCollection()

```
template<typename T >
void Collection< T >::SetCollection (
            std::vector< T > collection )  [inline]
```

Set the collection of items.

Parameters

collection	The new collection of items.

6.5.3.14 size()

```
template<typename T >
virtual size_t Collection< T >::size ( )  [inline], [virtual]
```

Get the size of the collection.

Returns

size_t The size of the collection.

6.5.3.15 slice() [1/3]

```
template<typename T >
virtual std::vector< T > Collection< T >::slice ( )  [inline], [virtual]
```

Get a slice of the entire collection.

Returns

std::vector<T> A slice of the collection.

6.5.3.16 slice() [2/3]

```
template<typename T >
virtual std::vector< T > Collection< T >::slice (
            int start )  [inline], [virtual]
```

Get a slice of the collection from the specified start index to the end of the collection.

Parameters

start	The start index of the slice.

Returns

std::vector<T> A slice of the collection.

6.5.3.17 slice() [3/3]

```
template<typename T >
virtual std::vector< T > Collection< T >::slice (
            int start,
            int end )  [inline], [virtual]
```

Get a slice of the collection from the specified start index to the specified end index.

Parameters

start	The start index of the slice.
end	The end index of the slice.

Returns

std::vector<T> A slice of the collection.

6.5.3.18 sort()

```
template<typename T >
virtual void Collection< T >::sort (
            std::function< bool(T, T)> compare )  [inline], [virtual]
```

Sort the items in the collection using a custom comparison function.

Parameters

compare	The comparison function to be used for sorting.

6.5.4 Member Data Documentation

6.5.4.1 m_collection

```
template<typename T >
std::vector<T> Collection< T >::m_collection  [protected]
```

The collection of items.

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

- AEGP/Core/Base/Collection.hpp

6.6 CollectionDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_Collection2H *collection)

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

- AEGP/Core/Base/AEGeneral.hpp

6.7 CollectionSuite2 Class Reference

Public Member Functions

- **CollectionSuite2** (const CollectionSuite2 &)=delete
- CollectionSuite2 & **operator=** (const CollectionSuite2 &)=delete
- **CollectionSuite2** (CollectionSuite2 &&)=delete
- CollectionSuite2 & **operator=** (CollectionSuite2 &&)=delete
- Collection2Ptr **newCollection** ()
- void **disposeCollection** (Collection2Ptr collectionH)
- A_long **getCollectionNumItems** (Collection2Ptr collectionH)
- AEGP_CollectionItemV2 **getCollectionItemByIndex** (Collection2Ptr collectionH, A_long indexL)
- void **collectionPushBack** (Collection2Ptr collectionH, const AEGP_CollectionItemV2 &itemP)
- void **collectionErase** (Collection2Ptr collectionH, A_long index_firstL, A_long index_lastL)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.8 AE::ColorProperty Class Reference

Public Member Functions

- **ColorProperty** (StreamRefPtr stream)
- AEGP_ColorVal **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (AEGP_ColorVal value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.9 Command Class Reference

Abstract base class for creating commands within the plugin.

```
#include <Plugin.hpp>
```

Public Member Functions

- Command (std::string name, AE_MenuID menuID, int after_item=INSERT_SORTED)
- virtual ~Command ()=default
- virtual void execute ()=0
- virtual void updateMenu ()=0
- std::string getName () const
- int getCommand () const
- void **insertCommand** (AE_MenuID menuID, int after_item=INSERT_SORTED)
- void **setCommandName** (std::string name)
- void **enableCommand** (bool enable)
- void **checkCommand** (bool check)

6.9.1 Detailed Description

Abstract base class for creating commands within the plugin.

This class defines the structure for commands that can be executed by the plugin. Each command is associated with a specific action or behavior.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Command()

```
Command::Command (
            std::string name,
            AE_MenuID menuID,
            int after_item = INSERT_SORTED )  [inline]
```

Constructs a Command with a name, menu ID, and insertion order.

Parameters

name	The name of the command.
menuID	The ID of the menu where the command will be inserted.
after_item	Specifies the order of the command within the menu. Defaults to INSERT_SORTED.

6.9.2.2 ~Command()

```
virtual Command::~Command ( )  [virtual], [default]
```

Virtual destructor for cleanup.

6.9.3 Member Function Documentation

6.9.3.1 execute()

```
virtual void Command::execute ( )  [pure virtual]
```

Executes the command's action. Must be implemented by derived classes. This is where you'll execute the command's action or behavior. You'll do whatever logic you'd like here– AE related or not.

6.9.3.2 getCommand()

```
int Command::getCommand ( ) const  [inline]
```

Retrieves the command's unique identifier.

Returns

The command's identifier.

6.9.3.3 getName()

```
std::string Command::getName ( ) const  [inline]
```

Retrieves the name of the command.

Returns

The command's name.

6.9.3.4 updateMenu()

```
virtual void Command::updateMenu ( )  [pure virtual]
```

Updates the state or appearance of the menu item associated with this command. Must be implemented by derived classes.

This is used in the updateMenuHook to update the state of the menu item. Use the helper functions to enable, disable, or check the menu item.

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

- AEGP/Template/Plugin.hpp

6.10 CommandSuite1 Class Reference

Public Member Functions

- **CommandSuite1** (const CommandSuite1 &)=delete
- CommandSuite1 & **operator=** (const CommandSuite1 &)=delete
- **CommandSuite1** (CommandSuite1 &&)=delete
- CommandSuite1 & **operator=** (CommandSuite1 &&)=delete
- AEGP_Command **getUniqueCommand** ()
- void **insertMenuCommand** (AEGP_Command command, const std::string &nameZ, AE_MenuID menu_id, A_long after_itemL)
- void **removeMenuCommand** (AEGP_Command command)
- void **setMenuCommandName** (AEGP_Command command, const std::string &nameZ)
- void **enableCommand** (AEGP_Command command)
- void **disableCommand** (AEGP_Command command)
- void **checkMarkMenuCommand** (AEGP_Command command, A_Boolean checkB)
- void **doCommand** (AEGP_Command command)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.11 AE::CompItem Class Reference

A class for representing a composition in After Effects.

`#include <Comp.hpp>`

Public Member Functions

- **CompItem** ()
  Default constructor for CompItem.
- CompItem (ItemPtr item)
  Constructor for CompItem that takes an ItemPtr.
- CompItem (CompPtr comp)
  Constructor for CompItem that takes a CompPtr.
- CompItem (CompItem const &comp)
  Copy constructor for CompItem.
- std::vector< Layer > layers () const
  Get the layers of the composition.
- std::shared_ptr< Layer > layer (int index) const
  Get the layer at the given index.
- std::shared_ptr< Layer > layer (std::string name) const
  Get the layer with the given name.
- std::shared_ptr< Layer > addLayer (std::shared_ptr< Item > itemToAdd)
  Add a layer to the composition.
- void removeLayer (std::shared_ptr< Layer > itemToRemove)
  Remove a layer from the composition.
- void removeLayer (int index)
  Remove a layer from the composition based on its index.
- void removeLayer (std::string name)
  Remove a layer from the composition based on its name.
- void showLayerNames (bool show)
  Show or hide layer names in the composition.

Public Member Functions inherited from AE::Item

- **Item** ()
  Default constructor.
- Item (ItemPtr item)
  Constructor that takes an existing item pointer.
- virtual ~**Item** ()
  Destructor.
- void Select (bool deselectOthers=true)
  Selects the item.
- void **Deselect** ()
  Deselects the item.
- bool IsSelected () const
  Checks if the item is selected.
- std::string Name () const
  Returns the name of the item.

- std::tuple< int, int > Dimensions () const

  Returns the dimensions of the item.
- int Width () const

  Returns the width of the item.
- int Height () const

  Returns the height of the item.
- void **Delete** ()

  Deletes the item.
- bool Missing () const

  Checks if the item is missing.
- bool HasProxy () const

  Checks if the item has a proxy.
- bool UsingProxy () const

  Checks if the item is using a proxy.
- bool MissingProxy () const

  Checks if the item is missing a proxy.
- bool HasVideo () const

  Checks if the item has video.
- bool HasAudio () const

  Checks if the item has audio.
- bool Still () const

  Checks if the item is a still image.
- bool ActiveAudio () const

  Checks if the item has active audio.
- double Duration () const

  Returns the duration of the item.
- double Time () const

  Returns the current time of the item.

Additional Inherited Members

Static Public Member Functions inherited from AE::Item

- static Item ActiveItem ()

  Returns the active item.

6.11.1 Detailed Description

A class for representing a composition in After Effects.

This class represents a composition in After Effects. It inherits from the base class Item.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 CompItem() [1/3]

```
AE::CompItem::CompItem (
            ItemPtr item )  [inline]
```

Constructor for CompItem that takes an ItemPtr.

Parameters

item	A pointer to an Item object

6.11.2.2 CompItem() [2/3]

```
AE::CompItem::CompItem (
            CompPtr comp )  [inline]
```

Constructor for CompItem that takes a CompPtr.

Parameters

comp	A pointer to a Comp object

6.11.2.3 CompItem() [3/3]

```
AE::CompItem::CompItem (
            CompItem const & comp )  [inline]
```

Copy constructor for CompItem.

Parameters

comp	A reference to a CompItem object to be copied

6.11.3 Member Function Documentation

6.11.3.1 addLayer()

```
std::shared_ptr< Layer > AE::CompItem::addLayer (
            std::shared_ptr< Item > itemToAdd )
```

Add a layer to the composition.

Parameters

itemToAdd	A shared pointer to the Item object to be added as a layer

Returns

std::shared_ptr<Layer> A shared pointer to the newly added Layer object

6.11.3.2 layer() [1/2]

```
std::shared_ptr< Layer > AE::CompItem::layer (
```

```
            int index ) const
```

Get the layer at the given index.

Parameters

index	The index of the layer to retrieve

Returns

std::shared_ptr<Layer> A shared pointer to the Layer object at the given index

6.11.3.3 layer() [2/2]

```
std::shared_ptr< Layer > AE::CompItem::layer (
            std::string name ) const
```

Get the layer with the given name.

Parameters

name	The name of the layer to retrieve

Returns

std::shared_ptr<Layer> A shared pointer to the Layer object with the given name

6.11.3.4 layers()

```
std::vector< Layer > AE::CompItem::layers ( ) const
```

Get the layers of the composition.

Returns

std::vector<Layer> A vector of Layer objects representing the layers in the composition

6.11.3.5 removeLayer() [1/3]

```
void AE::CompItem::removeLayer (
            int index )
```

Remove a layer from the composition based on its index.

Parameters

index	The index of the layer to be removed

6.11.3.6 removeLayer() [2/3]

```
void AE::CompItem::removeLayer (
            std::shared_ptr< Layer > itemToRemove )
```

Remove a layer from the composition.

Parameters

itemToRemove	A shared pointer to the Layer object to be removed

6.11.3.7 removeLayer() [3/3]

```
void AE::CompItem::removeLayer (
            std::string name )
```

Remove a layer from the composition based on its name.

Parameters

name	The name of the layer to be removed

6.11.3.8 showLayerNames()

```
void AE::CompItem::showLayerNames (
            bool show )
```

Show or hide layer names in the composition.

Parameters

show	A boolean value indicating whether to show or hide layer names

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

- AEGP/Core/Comp.hpp

6.12 CompSuite11 Class Reference

Public Member Functions

- **CompSuite11** (const CompSuite11 &)=delete
- CompSuite11 & **operator=** (const CompSuite11 &)=delete
- **CompSuite11** (CompSuite11 &&)=delete
- CompSuite11 & **operator=** (CompSuite11 &&)=delete
- CompPtr **GetCompFromItem** (ItemPtr item)

- ItemPtr **GetItemFromComp** (CompPtr comp)
- std::tuple< short, short > **GetCompDownsampleFactor** (CompPtr comp)
- void **SetCompDownsampleFactor** (CompPtr comp, const std::tuple< short, short > &factor)
- ColorVal **GetCompBGColor** (CompPtr comp)
- void **SetCompBGColor** (CompPtr comp, const ColorVal &color)
- AE_CompFlag **GetCompFlags** (CompPtr comp)
- bool **GetShowLayerNameOrSourceName** (CompPtr comp)
- void **SetShowLayerNameOrSourceName** (CompPtr comp, bool showLayerName)
- bool **GetShowBlendModes** (CompPtr comp)
- void **SetShowBlendModes** (CompPtr comp, bool showBlendModes)
- double **GetCompFramerate** (CompPtr comp)
- void **SetCompFrameRate** (CompPtr comp, double fps)
- std::tuple< A_Ratio, A_Ratio > **GetCompShutterAnglePhase** (CompPtr comp)
- std::tuple< A_Time, A_Time > **GetCompShutterFrameRange** (CompPtr comp, A_Time compTime)
- int **GetCompSuggestedMotionBlurSamples** (CompPtr comp)
- void **SetCompSuggestedMotionBlurSamples** (CompPtr comp, int samples)
- int **GetCompMotionBlurAdaptiveSampleLimit** (CompPtr comp)
- void **SetCompMotionBlurAdaptiveSampleLimit** (CompPtr comp, int samples)
- A_Time **GetCompWorkAreaStart** (CompPtr comp)
- A_Time **GetCompWorkAreaDuration** (CompPtr comp)
- void **SetCompWorkAreaStartAndDuration** (CompPtr comp, A_Time workAreaStart, A_Time workArea↩ Duration)
- LayerPtr **CreateSolidInComp** (CompPtr comp, const std::string &name, int width, int height, const ColorVal &color, A_Time duration)
- LayerPtr **CreateCameraInComp** (CompPtr comp, const std::string &name, A_FloatPoint centerPoint)
- LayerPtr **CreateLightInComp** (CompPtr comp, const std::string &name, A_FloatPoint centerPoint)
- CompPtr **CreateComp** (ItemPtr parentFolder, const std::string &name, int width, int height, const A_Ratio &pixelAspectRatio, A_Time duration, const A_Ratio &framerate)
- Collection2Ptr **GetNewCollectionFromCompSelection** (AEGP_PluginID pluginId, CompPtr comp)
- A_Time **GetCompDisplayStartTime** (CompPtr comp)
- void **SetCompDisplayStartTime** (CompPtr comp, A_Time startTime)
- void **SetCompDuration** (CompPtr comp, A_Time duration)
- CompPtr **DuplicateComp** (CompPtr comp)
- A_Time **GetCompFrameDuration** (CompPtr comp)
- CompPtr **GetMostRecentlyUsedComp** ()
- LayerPtr **CreateVectorLayerInComp** (CompPtr comp)
- StreamRefPtr **GetNewCompMarkerStream** (CompPtr parentComp)
- bool **GetCompDisplayDropFrame** (CompPtr comp)
- void **SetCompDisplayDropFrame** (CompPtr comp, bool dropFrame)
- void **ReorderCompSelection** (CompPtr comp, int index)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.13 ImportOptions::Config Struct Reference

Represents the configuration options for importing assets.

`#include <Import.hpp>`

Public Attributes

- std::optional< double > **frameRate**

  The frame rate of the imported assets.
- std::optional< int > **width**

  The width of the imported assets.
- std::optional< int > **height**

  The height of the imported assets.
- std::optional< std::string > **name**

  The name of the imported assets.
- std::optional< double > **duration**

  The duration of the imported assets.

6.13.1 Detailed Description

Represents the configuration options for importing assets.

The Config struct defines additional configuration parameters for importing assets. It includes options such as frame rate, width, height, name, and duration.

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

- AEGP/Core/Base/Import.hpp

6.14 DynamicStreamSuite4 Class Reference

Public Member Functions

- **DynamicStreamSuite4** (const DynamicStreamSuite4 &)=delete
- DynamicStreamSuite4 & **operator=** (const DynamicStreamSuite4 &)=delete
- **DynamicStreamSuite4** (DynamicStreamSuite4 &&)=delete
- DynamicStreamSuite4 & **operator=** (DynamicStreamSuite4 &&)=delete
- StreamRefPtr **GetNewStreamRefForLayer** (LayerPtr layer)
- StreamRefPtr **GetNewStreamRefForMask** (MaskRefPtr mask)
- A_long **GetStreamDepth** (StreamRefPtr stream)
- AE_StreamGroupingType **GetStreamGroupingType** (StreamRefPtr stream)
- A_long **GetNumStreamsInGroup** (StreamRefPtr stream)
- AE_DynStreamFlag **GetDynamicStreamFlags** (StreamRefPtr stream)
- void **SetDynamicStreamFlag** (StreamRefPtr stream, AE_DynStreamFlag oneFlag, bool undoable, bool set)
- StreamRefPtr **GetNewStreamRefByIndex** (StreamRefPtr parentGroup, A_long index)
- StreamRefPtr **GetNewStreamRefByMatchname** (StreamRefPtr parentGroup, const std::string &match↩ Name)
- void **DeleteStream** (StreamRefPtr stream)
- void **ReorderStream** (StreamRefPtr stream, A_long newIndex)
- A_long **DuplicateStream** (StreamRefPtr stream)
- void **SetStreamName** (StreamRefPtr stream, const std::string &newName)
- bool **CanAddStream** (StreamRefPtr parentGroup, const std::string &matchName)
- StreamRefPtr **AddStream** (StreamRefPtr parentGroup, const std::string &matchName)
- std::string **GetMatchname** (StreamRefPtr stream)
- StreamRefPtr **GetNewParentStreamRef** (StreamRefPtr stream)

- bool **GetStreamIsModified** (StreamRefPtr stream)
- bool **IsSeparationLeader** (StreamRefPtr stream)
- bool **AreDimensionsSeparated** (StreamRefPtr leaderStream)
- void **SetDimensionsSeparated** (StreamRefPtr leaderStream, bool separated)
- StreamRefPtr **GetSeparationFollower** (A_long dimension, StreamRefPtr leaderStream)
- bool **IsSeparationFollower** (StreamRefPtr stream)
- StreamRefPtr **GetSeparationLeader** (StreamRefPtr followerStream)
- A_short **GetSeparationDimension** (StreamRefPtr stream)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.15 EffectDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_EffectRefH *effect)

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

- AEGP/Core/Base/AEGeneral.hpp

6.16 EffectSuite4 Class Reference

Public Member Functions

- **EffectSuite4** (const EffectSuite4 &)=delete
- EffectSuite4 & **operator=** (const EffectSuite4 &)=delete
- **EffectSuite4** (EffectSuite4 &&)=delete
- EffectSuite4 & **operator=** (EffectSuite4 &&)=delete
- A_long **getLayerNumEffects** (LayerPtr layer)
- EffectRefPtr **getLayerEffectByIndex** (LayerPtr layer, AEGP_EffectIndex layer_effect_index)
- AEGP_InstalledEffectKey **getInstalledKeyFromLayerEffect** (EffectRefPtr effect_ref)
- std::tuple< PF_ParamType, PF_ParamDefUnion > **getEffectParamUnionByIndex** (EffectRefPtr effect_ref, PF_ParamIndex param_index)
- AE_EffectFlags **getEffectFlags** (EffectRefPtr effect_ref)
- void **setEffectFlags** (EffectRefPtr effect_ref, AE_EffectFlags effect_flags_set_mask, AE_EffectFlags effect_flags)
- void **reorderEffect** (EffectRefPtr effect_ref, A_long effect_index)
- void **effectCallGeneric** (EffectRefPtr effect_ref, const A_Time *timePT, PF_Cmd effect_cmd, void *effect_extraPV)
- void **disposeEffect** (EffectRefPtr effect_ref)
- EffectRefPtr **applyEffect** (LayerPtr layer, AEGP_InstalledEffectKey installed_effect_key)
- void **deleteLayerEffect** (EffectRefPtr effect_ref)
- A_long **getNumInstalledEffects** ()
- AEGP_InstalledEffectKey **getNextInstalledEffect** (AEGP_InstalledEffectKey installed_effect_key)
- std::string **getEffectName** (AEGP_InstalledEffectKey installed_effect_key)

- std::string **getEffectMatchName** (AEGP_InstalledEffectKey installed_effect_key)
- std::string **getEffectCategory** (AEGP_InstalledEffectKey installed_effect_key)
- EffectRefPtr **duplicateEffect** (EffectRefPtr original_effect_ref)
- A_u_long **numEffectMask** (EffectRefPtr effect_ref)
- AEGP_MaskIDVal **getEffectMaskID** (EffectRefPtr effect_ref, A_u_long mask_indexL)
- StreamRefPtr **addEffectMask** (EffectRefPtr effect_ref, AEGP_MaskIDVal id_val)
- void **removeEffectMask** (EffectRefPtr effect_ref, AEGP_MaskIDVal id_val)
- StreamRefPtr **setEffectMask** (EffectRefPtr effect_ref, A_u_long mask_indexL, AEGP_MaskIDVal id_val)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.17 FootageDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_FootageH *footage)

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

- AEGP/Core/Base/AEGeneral.hpp

6.18 FootageSuite5 Class Reference

Public Member Functions

- **FootageSuite5** (const FootageSuite5 &)=delete
- FootageSuite5 & **operator=** (const FootageSuite5 &)=delete
- **FootageSuite5** (FootageSuite5 &&)=delete
- FootageSuite5 & **operator=** (FootageSuite5 &&)=delete
- FootagePtr **getMainFootageFromItem** (ItemPtr itemH)
- FootagePtr **getProxyFootageFromItem** (ItemPtr itemH)
- std::tuple< A_long, A_long > **getFootageNumFiles** (FootagePtr footageH)
- std::string **getFootagePath** (FootagePtr footageH, A_long frame_numL, A_long file_indexL)
- AEGP_FootageSignature **getFootageSignature** (FootagePtr footageH)
- FootagePtr newFootage (std::string pathZ, AEGP_FootageLayerKey layer_infoP0, AEGP_FileSequence↩ ImportOptions *sequence_optionsP0, AE_InterpretationStyle interp_style)
- ItemPtr **addFootageToProject** (FootagePtr footageH, ItemPtr folderH)
- void **setItemProxyFootage** (FootagePtr footageH, ItemPtr itemH)
- void **replaceItemMainFootage** (FootagePtr footageH, ItemPtr itemH)
- void **disposeFootage** (FootagePtr footageH)
- AEGP_FootageInterp **getFootageInterpretation** (ItemPtr itemH, bool proxyB)
- void **setFootageInterpretation** (ItemPtr itemH, bool proxyB, const AEGP_FootageInterp *interpP)
- AEGP_FootageLayerKey **getFootageLayerKey** (FootagePtr footageH)
- FootagePtr **newPlaceholderFootage** (std::string nameZ, A_long width, A_long height, A_Time durationPT)
- FootagePtr **newPlaceholderFootageWithPath** (std::string pathZ, AE_Platform path_platform, AEIO_File↩ Type file_type, A_long widthL, A_long heightL, A_Time durationPT)
- FootagePtr **newSolidFootage** (std::string nameZ, A_long width, A_long height, ColorVal colorP)
- ColorVal **getSolidFootageColor** (ItemPtr itemH, bool proxyB)
- void **setSolidFootageColor** (ItemPtr itemH, bool proxyB, ColorVal colorP)
- void **setSolidFootageDimensions** (ItemPtr itemH, bool proxyB, A_long widthL, A_long heightL)
- AEGP_SoundDataFormat **getFootageSoundDataFormat** (FootagePtr footageH)
- AEGP_FileSequenceImportOptions **getFootageSequenceImportOptions** (FootagePtr footageH)

6.18.1 Member Function Documentation

6.18.1.1 newFootage()

```
FootagePtr FootageSuite5::newFootage (
            std::string pathZ,
            AEGP_FootageLayerKey layer_infoP0,
            AEGP_FileSequenceImportOptions * sequence_optionsP0,
            AE_InterpretationStyle interp_style )
```

Parameters

pathZ	
layer_infoP0	
sequence_optionsP0	
interp_style	

Returns

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.19 FrameReceiptDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_FrameReceiptH *frameReceipt)

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

- AEGP/Core/Base/AEGeneral.hpp

6.20 Image Class Reference

Public Member Functions

- **Image** (WorldPtr world)
- UniformImage **data** ()
- void **saveImage** (const std::string &filename, const std::string &format)

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

- AEGP/Util/Image.hpp

6.21 ImportOptions Class Reference

Represents the options for importing assets into After Effects.

```
#include <Import.hpp>
```

Classes

- struct Config

  Represents the configuration options for importing assets.

Public Member Functions

- **ImportOptions** ()

  Default constructor for ImportOptions.
- ~**ImportOptions** ()

  Destructor for ImportOptions.
- bool importAssets (const std::variant< std::string, std::vector< std::string > > &files, const Config &config={})

  Imports assets into After Effects with the specified configuration options.

6.21.1 Detailed Description

Represents the options for importing assets into After Effects.

The ImportOptions class provides a set of configuration options for importing assets into After Effects. It supports importing assets as still images, image sequences, compositions, or individual layers.

6.21.2 Member Function Documentation

6.21.2.1 importAssets()

```
bool ImportOptions::importAssets (
            const std::variant< std::string, std::vector< std::string > > & files,
            const Config & config = {} )
```

Imports assets into After Effects with the specified configuration options.

The importAssets function imports assets into After Effects based on the provided files and configuration options. It supports importing assets as still images, image sequences, compositions, or individual layers.

Parameters

files	The files to import. It can be a single file or a vector of files.
config	The configuration options for importing the assets. Defaults to an empty configuration.

Returns

bool True if the assets were imported successfully, false otherwise.

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

- AEGP/Core/Base/Import.hpp

6.22 AE::Item Class Reference

Represents an After Effects item.

`#include <Item.hpp>`

Public Member Functions

- **Item** ()
  Default constructor.
- Item (ItemPtr item)
  Constructor that takes an existing item pointer.
- virtual ~**Item** ()
  Destructor.
- void Select (bool deselectOthers=true)
  Selects the item.
- void **Deselect** ()
  Deselects the item.
- bool IsSelected () const
  Checks if the item is selected.
- std::string Name () const
  Returns the name of the item.
- std::tuple< int, int > Dimensions () const
  Returns the dimensions of the item.
- int Width () const
  Returns the width of the item.
- int Height () const
  Returns the height of the item.
- void **Delete** ()
  Deletes the item.
- bool Missing () const
  Checks if the item is missing.
- bool HasProxy () const
  Checks if the item has a proxy.
- bool UsingProxy () const
  Checks if the item is using a proxy.
- bool MissingProxy () const
  Checks if the item is missing a proxy.
- bool HasVideo () const
  Checks if the item has video.
- bool HasAudio () const

Checks if the item has audio.

- bool Still () const

  Checks if the item is a still image.

- bool ActiveAudio () const

  Checks if the item has active audio.

- double Duration () const

  Returns the duration of the item.

- double Time () const

  Returns the current time of the item.

Static Public Member Functions

- static Item ActiveItem ()

  Returns the active item.

6.22.1 Detailed Description

Represents an After Effects item.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Item()

```
AE::Item::Item (
            ItemPtr item )  [inline]
```

Constructor that takes an existing item pointer.

Parameters

item	The item pointer.

6.22.3 Member Function Documentation

6.22.3.1 ActiveAudio()

```
bool AE::Item::ActiveAudio ( ) const
```

Checks if the item has active audio.

Returns

True if the item has active audio, false otherwise.

6.22.3.2 ActiveItem()

```
static Item AE::Item::ActiveItem ( )  [static]
```

Returns the active item.

Returns

The active item.

6.22.3.3 Dimensions()

```
std::tuple< int, int > AE::Item::Dimensions ( ) const
```

Returns the dimensions of the item.

Returns

A tuple containing the width and height of the item.

6.22.3.4 Duration()

```
double AE::Item::Duration ( ) const
```

Returns the duration of the item.

Returns

The duration of the item.

6.22.3.5 HasAudio()

```
bool AE::Item::HasAudio ( ) const
```

Checks if the item has audio.

Returns

True if the item has audio, false otherwise.

6.22.3.6 HasProxy()

```
bool AE::Item::HasProxy ( ) const
```

Checks if the item has a proxy.

Returns

True if the item has a proxy, false otherwise.

6.22.3.7 HasVideo()

```
bool AE::Item::HasVideo ( ) const
```

Checks if the item has video.

Returns

True if the item has video, false otherwise.

6.22.3.8 Height()

```
int AE::Item::Height ( ) const
```

Returns the height of the item.

Returns

The height of the item.

6.22.3.9 IsSelected()

```
bool AE::Item::IsSelected ( ) const
```

Checks if the item is selected.

Returns

True if the item is selected, false otherwise.

6.22.3.10 Missing()

```
bool AE::Item::Missing ( ) const
```

Checks if the item is missing.

Returns

True if the item is missing, false otherwise.

6.22.3.11 MissingProxy()

```
bool AE::Item::MissingProxy ( ) const
```

Checks if the item is missing a proxy.

Returns

True if the item is missing a proxy, false otherwise.

6.22.3.12 Name()

```
std::string AE::Item::Name ( ) const
```

Returns the name of the item.

Returns

The name of the item.

6.22.3.13 Select()

```
void AE::Item::Select (
            bool deselectOthers = true )
```

Selects the item.

Parameters

deselectOthers	Flag indicating whether to deselect other items.

6.22.3.14 Still()

```
bool AE::Item::Still ( ) const
```

Checks if the item is a still image.

Returns

True if the item is a still image, false otherwise.

6.22.3.15 Time()

```
double AE::Item::Time ( ) const
```

Returns the current time of the item.

Returns

The current time of the item.

6.22.3.16 UsingProxy()

```
bool AE::Item::UsingProxy ( ) const
```

Checks if the item is using a proxy.

Returns

True if the item is using a proxy, false otherwise.

6.22.3.17 Width()

```
int AE::Item::Width ( ) const
```

Returns the width of the item.

Returns

The width of the item.

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

- AEGP/Core/Base/Item.hpp

6.23 ItemSuite9 Class Reference

AE Item Suite.

```
#include <AEGeneral.hpp>
```

Public Member Functions

- **ItemSuite9** (const ItemSuite9 &)=delete
- ItemSuite9 & **operator=** (const ItemSuite9 &)=delete
- **ItemSuite9** (ItemSuite9 &&)=delete
- ItemSuite9 & **operator=** (ItemSuite9 &&)=delete
- ItemPtr **GetFirstProjItem** (ProjectPtr project)
- ItemPtr **GetNextProjItem** (ProjectPtr project, ItemPtr item)
- ItemPtr **GetActiveItem** ()
- bool **IsItemSelected** (ItemPtr item)
- void **SelectItem** (ItemPtr item, bool select, bool deselectOthers)
- AE_ItemType **GetItemType** (ItemPtr item)
- std::string **GetTypeName** (AE_ItemType itemType)
- std::string **GetItemName** (ItemPtr item)
- void **SetItemName** (ItemPtr item, const std::string &name)
- A_long **GetItemID** (ItemPtr item)
- AE_ItemFlag **GetItemFlags** (ItemPtr item)
- void **SetItemUseProxy** (ItemPtr item, bool useProxy)
- ItemPtr **GetItemParentFolder** (ItemPtr item)
- void **SetItemParentFolder** (ItemPtr item, ItemPtr parentFolder)
- A_Time **GetItemDuration** (ItemPtr item)
- A_Time **GetItemCurrentTime** (ItemPtr item)
- std::tuple< A_long, A_long > **GetItemDimensions** (ItemPtr item)
- A_Ratio **GetItemPixelAspectRatio** (ItemPtr item)
- void **DeleteItem** (ItemPtr item)
- ItemPtr **CreateNewFolder** (const std::string &name, ItemPtr parentFolder)
- void **SetItemCurrentTime** (ItemPtr item, A_Time newTime)
- std::string **GetItemComment** (ItemPtr item)
- void **SetItemComment** (ItemPtr item, const std::string &comment)
- AE_Label **GetItemLabel** (ItemPtr item)
- void **SetItemLabel** (ItemPtr item, AE_Label label)
- ItemViewPtr **GetItemMRUView** (ItemPtr item)

6.23.1 Detailed Description

AE Item Suite.

The Item Suite provides access to the After Effects project items.

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.24 ItemViewSuite1 Class Reference

Public Member Functions

- A_Time **GetItemViewPlaybackTime** (ItemViewPtr itemView, bool &isCurrentlyPreviewing)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.25 KeyframeSuite5 Class Reference

Public Member Functions

- **KeyframeSuite5** (const KeyframeSuite5 &)=delete
- KeyframeSuite5 & **operator=** (const KeyframeSuite5 &)=delete
- **KeyframeSuite5** (KeyframeSuite5 &&)=delete
- KeyframeSuite5 & **operator=** (KeyframeSuite5 &&)=delete
- A_long **GetStreamNumKFs** (StreamRefPtr stream)
- A_Time **GetKeyframeTime** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AE_LTimeMode time↩ Mode)
- AEGP_KeyframeIndex **InsertKeyframe** (StreamRefPtr stream, AE_LTimeMode timeMode, const A_Time &time)
- void **DeleteKeyframe** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex)
- AEGP_StreamValue2 **GetNewKeyframeValue** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex)
- void **SetKeyframeValue** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AEGP_StreamValue2 value)
- A_short **GetStreamValueDimensionality** (StreamRefPtr stream)
- A_short **GetStreamTemporalDimensionality** (StreamRefPtr stream)
- std::tuple< AEGP_StreamValue2, AEGP_StreamValue2 > **GetNewKeyframeSpatialTangents** (Stream↩ RefPtr stream, AEGP_KeyframeIndex keyIndex)
- void **SetKeyframeSpatialTangents** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AEGP_↩ StreamValue2 inTan, AEGP_StreamValue2 outTan)
- std::tuple< AE_KeyframeEase, AE_KeyframeEase > **GetKeyframeTemporalEase** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, A_long dimension)
- void **SetKeyframeTemporalEase** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, A_long dimension, AE_KeyframeEase inEase, AE_KeyframeEase outEase)

- AE_KeyframeFlag **GetKeyframeFlags** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex)
- void **SetKeyframeFlag** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AE_KeyframeFlag flag, bool value)
- std::tuple< AE_KeyInterp, AE_KeyInterp > **GetKeyframeInterpolation** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex)
- void **SetKeyframeInterpolation** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AE_KeyInterp inInterp, AE_KeyInterp outInterp)
- AddKeyframesInfoPtr **StartAddKeyframes** (StreamRefPtr stream)
- AEGP_KeyframeIndex **AddKeyframes** (AddKeyframesInfoPtr akH, AE_LTimeMode timeMode, const A_Time &time)
- void **SetAddKeyframe** (AddKeyframesInfoPtr akH, AEGP_KeyframeIndex keyIndex, AEGP_StreamValue2 value)
- A_long **GetKeyframeLabelColorIndex** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex)
- void **SetKeyframeLabelColorIndex** (StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, A_long keyLabel)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.26 AE::Layer Class Reference

Represents a layer in After Effects.

`#include <Layer.hpp>`

Public Member Functions

- **Layer** ()

  Default constructor for Layer class.
- virtual ~**Layer** ()=default

  Virtual destructor for Layer class.
- Layer (LayerPtr layer)

  Constructor for Layer class that takes a LayerPtr as input.
- LayerPtr getLayer ()

  Get the underlying LayerPtr object.
- void setLayer (LayerPtr layer)

  Set the underlying LayerPtr object.
- int index () const

  Get the index of the layer.
- void reOrder (int newIndex)

  Reorder the layer to a new index.
- std::shared_ptr< Item > source () const

  Get the source item of the layer.
- int sourceID () const

  Get the source ID of the layer.
- CompItem parentComp () const

  Get the parent composition of the layer.

- std::string name () const

  Get the name of the layer.
- std::string sourceName () const

  Get the name of the source of the layer.
- AE_LayerQual quality () const

  Get the quality of the layer.
- void setQuality (AE_LayerQual quality)

  Set the quality of the layer.
- bool videoActive () const

  Check if video is active for the layer.
- bool audioActive () const

  Check if audio is active for the layer.
- bool effectsActive () const

  Check if effects are active for the layer.
- bool motionBlur () const

  Check if motion blur is active for the layer.
- bool frameBlending () const

  Check if frame blending is active for the layer.
- bool locked () const

  Check if the layer is locked.
- bool shy () const

  Check if the layer is shy.
- bool collapsed () const

  Check if the layer is collapsed.
- bool autoOrient () const

  Check if auto-orientation is active for the layer.
- bool adjustmentLayer () const

  Check if the layer is an adjustment layer.
- bool timeRemap () const

  Check if time remapping is active for the layer.
- bool is3D () const

  Check if the layer is in 3D mode.
- bool lookAtCamera () const

  Check if the layer is set to look at the camera.
- bool lookAtPOI () const

  Check if the layer is set to look at the point of interest.
- bool solo () const

  Check if the layer is soloed.
- bool markersLocked () const

  Check if the layer's markers are locked.
- bool nullLayer () const

  Check if the layer is a null layer.
- bool hideLockedMask () const

  Check if locked masks are hidden for the layer.
- bool guideLayer () const

  Check if the layer is a guide layer.
- bool renderSeparately () const

  Check if the layer is set to render separately.
- bool environmentLayer () const

  Check if the layer is an environment layer.
- void setVideoActive (bool active)

Set the video active state for the layer.

- void setAudioActive (bool active)

  Set the audio active state for the layer.
- void setEffectsActive (bool active)

  Set the effects active state for the layer.
- void setMotionBlur (bool active)

  Set the motion blur state for the layer.
- void setFrameBlending (bool active)

  Set the frame blending state for the layer.
- void setLocked (bool active)

  Set the locked state for the layer.
- void setShy (bool active)

  Set the shy state for the layer.
- void setCollapsed (bool active)

  Set the collapsed state for the layer.
- void setAutoOrient (bool active)

  Set the auto-orientation state for the layer.
- void setAdjustmentLayer (bool active)

  Set the adjustment layer state for the layer.
- void setTimeRemap (bool active)

  Set the time remap state for the layer.
- void setIs3D (bool active)

  Set the 3D state for the layer.
- void setLookAtCamera (bool active)

  Set the look at camera state for the layer.
- void setLookAtPOI (bool active)

  Set the look at point of interest state for the layer.
- void setSolo (bool active)

  Set the solo state for the layer.
- void setMarkersLocked (bool active)

  Set the markers locked state for the layer.
- void setNullLayer (bool active)

  Set the null layer state for the layer.
- void setHideLockedMask (bool active)

  Set the hide locked mask state for the layer.
- void setGuideLayer (bool active)

  Set the guide layer state for the layer.
- void setRenderSeparately (bool active)

  Set the render separately state for the layer.
- void setEnvironmentLayer (bool active)

  Set the environment layer state for the layer.
- bool isVideoOn () const

  Check if video is on for the layer.
- bool isAudioOn () const

  Check if audio is on for the layer.
- double time () const

  Get the time of the layer.
- double inPoint () const

  Get the in point of the layer.
- double duration () const

  Get the duration of the layer.

- void setInPointAndDuration (double inPoint, double duration)

  Set the in point and duration of the layer.
- double offset () const

  Get the offset of the layer.
- void setOffset (double offset)

  Set the offset of the layer.
- double stretch () const

  Get the stretch of the layer.
- void setStretch (double stretch)

  Set the stretch of the layer.
- void **Delete** ()

  Delete the layer.
- Layer duplicate ()

  Duplicate the layer.
- AE_LayerSamplingQual samplingQuality () const

  Get the sampling quality of the layer.
- void setSamplingQuality (AE_LayerSamplingQual quality)

  Set the sampling quality of the layer.

Static Public Member Functions

- static Layer activeLayer ()

  Get the active layer.

6.26.1 Detailed Description

Represents a layer in After Effects.

This class provides functionality to manipulate and retrieve information about a layer in After Effects.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Layer()

```
AE::Layer::Layer (
            LayerPtr layer )  [inline]
```

Constructor for Layer class that takes a LayerPtr as input.

Parameters

layer	A pointer to a Layer object.

6.26.3 Member Function Documentation

6.26.3.1 activeLayer()

```
static Layer AE::Layer::activeLayer ( )  [static]
```

Get the active layer.

Returns

Layer The active layer.

6.26.3.2 adjustmentLayer()

```
bool AE::Layer::adjustmentLayer ( ) const
```

Check if the layer is an adjustment layer.

Returns

bool True if the layer is an adjustment layer, false otherwise.

6.26.3.3 audioActive()

```
bool AE::Layer::audioActive ( ) const
```

Check if audio is active for the layer.

Returns

bool True if audio is active, false otherwise.

6.26.3.4 autoOrient()

```
bool AE::Layer::autoOrient ( ) const
```

Check if auto-orientation is active for the layer.

Returns

bool True if auto-orientation is active, false otherwise.

6.26.3.5 collapsed()

```
bool AE::Layer::collapsed ( ) const
```

Check if the layer is collapsed.

Returns

bool True if the layer is collapsed, false otherwise.

6.26.3.6 duplicate()

```
Layer AE::Layer::duplicate ( )
```

Duplicate the layer.

Returns

Layer A duplicate of the layer.

6.26.3.7 duration()

```
double AE::Layer::duration ( ) const
```

Get the duration of the layer.

Returns

double The duration of the layer.

6.26.3.8 effectsActive()

```
bool AE::Layer::effectsActive ( ) const
```

Check if effects are active for the layer.

Returns

bool True if effects are active, false otherwise.

6.26.3.9 environmentLayer()

```
bool AE::Layer::environmentLayer ( ) const
```

Check if the layer is an environment layer.

Returns

bool True if the layer is an environment layer, false otherwise.

6.26.3.10 frameBlending()

```
bool AE::Layer::frameBlending ( ) const
```

Check if frame blending is active for the layer.

Returns

bool True if frame blending is active, false otherwise.

6.26.3.11 getLayer()

```
LayerPtr AE::Layer::getLayer ( )  [inline]
```

Get the underlying LayerPtr object.

Returns

LayerPtr A pointer to the underlying Layer object.

6.26.3.12 guideLayer()

```
bool AE::Layer::guideLayer ( ) const
```

Check if the layer is a guide layer.

Returns

bool True if the layer is a guide layer, false otherwise.

6.26.3.13 hideLockedMask()

```
bool AE::Layer::hideLockedMask ( ) const
```

Check if locked masks are hidden for the layer.

Returns

bool True if locked masks are hidden, false otherwise.

6.26.3.14 index()

```
int AE::Layer::index ( ) const
```

Get the index of the layer.

Returns

int The index of the layer.

6.26.3.15 inPoint()

```
double AE::Layer::inPoint ( ) const
```

Get the in point of the layer.

Returns

double The in point of the layer.

6.26.3.16 is3D()

```
bool AE::Layer::is3D ( ) const
```

Check if the layer is in 3D mode.

Returns

bool True if the layer is in 3D mode, false otherwise.

6.26.3.17 isAudioOn()

```
bool AE::Layer::isAudioOn ( ) const
```

Check if audio is on for the layer.

Returns

bool True if audio is on, false otherwise.

6.26.3.18 isVideoOn()

```
bool AE::Layer::isVideoOn ( ) const
```

Check if video is on for the layer.

Returns

bool True if video is on, false otherwise.

6.26.3.19 locked()

```
bool AE::Layer::locked ( ) const
```

Check if the layer is locked.

Returns

bool True if the layer is locked, false otherwise.

6.26.3.20 lookAtCamera()

```
bool AE::Layer::lookAtCamera ( ) const
```

Check if the layer is set to look at the camera.

Returns

bool True if the layer is set to look at the camera, false otherwise.

6.26.3.21 lookAtPOI()

```
bool AE::Layer::lookAtPOI ( ) const
```

Check if the layer is set to look at the point of interest.

Returns

bool True if the layer is set to look at the point of interest, false otherwise.

6.26.3.22 markersLocked()

```
bool AE::Layer::markersLocked ( ) const
```

Check if the layer's markers are locked.

Returns

bool True if the layer's markers are locked, false otherwise.

6.26.3.23 motionBlur()

```
bool AE::Layer::motionBlur ( ) const
```

Check if motion blur is active for the layer.

Returns

bool True if motion blur is active, false otherwise.

6.26.3.24 name()

```
std::string AE::Layer::name ( ) const
```

Get the name of the layer.

Returns

std::string The name of the layer.

6.26.3.25 nullLayer()

```
bool AE::Layer::nullLayer ( ) const
```

Check if the layer is a null layer.

Returns

bool True if the layer is a null layer, false otherwise.

6.26.3.26 offset()

```
double AE::Layer::offset ( ) const
```

Get the offset of the layer.

Returns

double The offset of the layer.

6.26.3.27 parentComp()

```
CompItem AE::Layer::parentComp ( ) const
```

Get the parent composition of the layer.

Returns

CompItem The parent composition of the layer.

6.26.3.28 quality()

```
AE_LayerQual AE::Layer::quality ( ) const
```

Get the quality of the layer.

Returns

AE_LayerQual The quality of the layer.

6.26.3.29 renderSeparately()

```
bool AE::Layer::renderSeparately ( ) const
```

Check if the layer is set to render separately.

Returns

bool True if the layer is set to render separately, false otherwise.

6.26.3.30 reOrder()

```
void AE::Layer::reOrder (
            int newIndex )
```

Reorder the layer to a new index.

Parameters

newIndex	The new index for the layer.

6.26.3.31 samplingQuality()

```
AE_LayerSamplingQual AE::Layer::samplingQuality ( ) const
```

Get the sampling quality of the layer.

Returns

AE_LayerSamplingQual The sampling quality of the layer.

6.26.3.32 setAdjustmentLayer()

```
void AE::Layer::setAdjustmentLayer (
            bool active )
```

Set the adjustment layer state for the layer.

Parameters

active	The adjustment layer state.

6.26.3.33 setAudioActive()

```
void AE::Layer::setAudioActive (
            bool active )
```

Set the audio active state for the layer.

Parameters

active	The audio active state.

6.26.3.34 setAutoOrient()

```
void AE::Layer::setAutoOrient (
            bool active )
```

Set the auto-orientation state for the layer.

Parameters

active	The auto-orientation state.

6.26.3.35 setCollapsed()

```
void AE::Layer::setCollapsed (
            bool active )
```

Set the collapsed state for the layer.

Parameters

active	The collapsed state.

6.26.3.36 setEffectsActive()

```
void AE::Layer::setEffectsActive (
            bool active )
```

Set the effects active state for the layer.

Parameters

active	The effects active state.

6.26.3.37 setEnvironmentLayer()

```
void AE::Layer::setEnvironmentLayer (
            bool active )
```

Set the environment layer state for the layer.

Parameters

active	The environment layer state.

6.26.3.38 setFrameBlending()

```
void AE::Layer::setFrameBlending (
            bool active )
```

Set the frame blending state for the layer.

Parameters

active	The frame blending state.

6.26.3.39 setGuideLayer()

```
void AE::Layer::setGuideLayer (
            bool active )
```

Set the guide layer state for the layer.

Parameters

active	The guide layer state.

6.26.3.40 setHideLockedMask()

```
void AE::Layer::setHideLockedMask (
            bool active )
```

Set the hide locked mask state for the layer.

Parameters

active	The hide locked mask state.

6.26.3.41 setInPointAndDuration()

```
void AE::Layer::setInPointAndDuration (
            double inPoint,
            double duration )
```

Set the in point and duration of the layer.

Parameters

inPoint	The in point of the layer.
duration	The duration of the layer.

6.26.3.42 setIs3D()

```
void AE::Layer::setIs3D (
            bool active )
```

Set the 3D state for the layer.

Parameters

active	The 3D state.

6.26.3.43 setLayer()

```
void AE::Layer::setLayer (
            LayerPtr layer )  [inline]
```

Set the underlying LayerPtr object.

Parameters

layer	A pointer to a Layer object.

6.26.3.44 setLocked()

```
void AE::Layer::setLocked (
            bool active )
```

Set the locked state for the layer.

Parameters

active	The locked state.

6.26.3.45 setLookAtCamera()

```
void AE::Layer::setLookAtCamera (
            bool active )
```

Set the look at camera state for the layer.

Parameters

active	The look at camera state.

6.26.3.46 setLookAtPOI()

```
void AE::Layer::setLookAtPOI (
            bool active )
```

Set the look at point of interest state for the layer.

Parameters

active	The look at point of interest state.

6.26.3.47 setMarkersLocked()

```
void AE::Layer::setMarkersLocked (
            bool active )
```

Set the markers locked state for the layer.

Parameters

active	The markers locked state.

6.26.3.48 setMotionBlur()

```
void AE::Layer::setMotionBlur (
            bool active )
```

Set the motion blur state for the layer.

Parameters

active	The motion blur state.

6.26.3.49 setNullLayer()

```
void AE::Layer::setNullLayer (
            bool active )
```

Set the null layer state for the layer.

Parameters

active	The null layer state.

6.26.3.50 setOffset()

```
void AE::Layer::setOffset (
            double offset )
```

Set the offset of the layer.

Parameters

offset	The offset of the layer.

6.26.3.51 setQuality()

```
void AE::Layer::setQuality (
            AE_LayerQual quality )
```

Set the quality of the layer.

Parameters

quality	The quality of the layer.

6.26.3.52 setRenderSeparately()

```
void AE::Layer::setRenderSeparately (
            bool active )
```

Set the render separately state for the layer.

Parameters

active	The render separately state.

6.26.3.53 setSamplingQuality()

```
void AE::Layer::setSamplingQuality (
            AE_LayerSamplingQual quality )
```

Set the sampling quality of the layer.

Parameters

quality	The sampling quality of the layer.

6.26.3.54 setShy()

```
void AE::Layer::setShy (
            bool active )
```

Set the shy state for the layer.

Parameters

active	The shy state.

6.26.3.55 setSolo()

```
void AE::Layer::setSolo (
            bool active )
```

Set the solo state for the layer.

Parameters

active	The solo state.

6.26.3.56 setStretch()

```
void AE::Layer::setStretch (
            double stretch )
```

Set the stretch of the layer.

Parameters

stretch	The stretch of the layer.

6.26.3.57 setTimeRemap()

```
void AE::Layer::setTimeRemap (
            bool active )
```

Set the time remap state for the layer.

Parameters

active	The time remap state.

6.26.3.58 setVideoActive()

```
void AE::Layer::setVideoActive (
            bool active )
```

Set the video active state for the layer.

Parameters

active	The video active state.

6.26.3.59 shy()

```
bool AE::Layer::shy ( ) const
```

Check if the layer is shy.

Returns

bool True if the layer is shy, false otherwise.

6.26.3.60 solo()

```
bool AE::Layer::solo ( ) const
```

Check if the layer is soloed.

Returns

bool True if the layer is soloed, false otherwise.

6.26.3.61 source()

```
std::shared_ptr< Item > AE::Layer::source ( ) const
```

Get the source item of the layer.

Returns

std::shared_ptr<Item> A shared pointer to the source item of the layer.

6.26.3.62 sourceID()

```
int AE::Layer::sourceID ( ) const
```

Get the source ID of the layer.

Returns

int The source ID of the layer.

6.26.3.63 sourceName()

```
std::string AE::Layer::sourceName ( ) const
```

Get the name of the source of the layer.

Returns

std::string The name of the source of the layer.

6.26.3.64 stretch()

```
double AE::Layer::stretch ( ) const
```

Get the stretch of the layer.

Returns

double The stretch of the layer.

6.26.3.65 time()

```
double AE::Layer::time ( ) const
```

Get the time of the layer.

Returns

double The time of the layer.

6.26.3.66 timeRemap()

```
bool AE::Layer::timeRemap ( ) const
```

Check if time remapping is active for the layer.

Returns

bool True if time remapping is active, false otherwise.

6.26.3.67 videoActive()

```
bool AE::Layer::videoActive ( ) const
```

Check if video is active for the layer.

Returns

bool True if video is active, false otherwise.

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

- AEGP/Core/Base/Layer.hpp

6.27 AE::LayerIDProperty Class Reference

Public Member Functions

- **LayerIDProperty** (StreamRefPtr stream)
- A_long **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (A_long value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.28 LayerRenderOptionsDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_LayerRenderOptionsH *layerRenderOptions)

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

- AEGP/Core/Base/AEGeneral.hpp

6.29 LayerRenderOptionsSuite2 Class Reference

Public Member Functions

- **LayerRenderOptionsSuite2** (const LayerRenderOptionsSuite2 &)=delete
- LayerRenderOptionsSuite2 & **operator=** (const LayerRenderOptionsSuite2 &)=delete
- **LayerRenderOptionsSuite2** (LayerRenderOptionsSuite2 &&)=delete
- LayerRenderOptionsSuite2 & **operator=** (LayerRenderOptionsSuite2 &&)=delete
- LayerRenderOptionsPtr **newFromLayer** (LayerPtr layer)
- LayerRenderOptionsPtr **newFromUpstreamOfEffect** (EffectRefPtr effect_ref)
- LayerRenderOptionsPtr **newFromDownstreamOfEffect** (EffectRefPtr effect_ref)
- LayerRenderOptionsPtr **duplicate** (LayerRenderOptionsPtr optionsH)
- void **dispose** (LayerRenderOptionsPtr optionsH)
- void **setTime** (LayerRenderOptionsPtr optionsH, A_Time time)
- A_Time **getTime** (LayerRenderOptionsPtr optionsH)
- void **setTimeStep** (LayerRenderOptionsPtr optionsH, A_Time time_step)
- A_Time **getTimeStep** (LayerRenderOptionsPtr optionsH)
- void **setWorldType** (LayerRenderOptionsPtr optionsH, AE_WorldType type)
- AE_WorldType **getWorldType** (LayerRenderOptionsPtr optionsH)
- void **setDownsampleFactor** (LayerRenderOptionsPtr optionsH, A_short x, A_short y)
- std::tuple< A_short, A_short > **getDownsampleFactor** (LayerRenderOptionsPtr optionsH)
- void **setMatteMode** (LayerRenderOptionsPtr optionsH, AE_MatteMode mode)
- AE_MatteMode **getMatteMode** (LayerRenderOptionsPtr optionsH)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.30 LayerSuite9 Class Reference

Public Member Functions

- **LayerSuite9** (const LayerSuite9 &)=delete
- LayerSuite9 & **operator=** (const LayerSuite9 &)=delete
- **LayerSuite9** (LayerSuite9 &&)=delete
- LayerSuite9 & **operator=** (LayerSuite9 &&)=delete
- A_long **GetCompNumLayers** (CompPtr comp)
- LayerPtr **GetCompLayerByIndex** (CompPtr comp, A_long layerIndex)
- LayerPtr **GetActiveLayer** ()
- A_long **GetLayerIndex** (LayerPtr layer)
- ItemPtr **GetLayerSourceItem** (LayerPtr layer)
- A_long **GetLayerSourceItemID** (LayerPtr layer)
- CompPtr **GetLayerParentComp** (LayerPtr layer)
- std::tuple< std::string, std::string > **GetLayerName** (LayerPtr layer)
- AE_LayerQual **GetLayerQuality** (LayerPtr layer)
- void **SetLayerQuality** (LayerPtr layer, AE_LayerQual quality)
- AE_LayerFlag **GetLayerFlags** (LayerPtr layer)
- void **SetLayerFlag** (LayerPtr layer, AE_LayerFlag singleFlag, bool value)
- bool **IsLayerVideoReallyOn** (LayerPtr layer)
- bool **IsLayerAudioReallyOn** (LayerPtr layer)

- A_Time **GetLayerCurrentTime** (LayerPtr layer, AE_LTimeMode timeMode)
- A_Time **GetLayerInPoint** (LayerPtr layer, AE_LTimeMode timeMode)
- A_Time **GetLayerDuration** (LayerPtr layer, AE_LTimeMode timeMode)
- void **SetLayerInPointAndDuration** (LayerPtr layer, AE_LTimeMode timeMode, A_Time inPoint, A_Time duration)
- A_Time **GetLayerOffset** (LayerPtr layer)
- void **SetLayerOffset** (LayerPtr layer, A_Time offset)
- A_Ratio **GetLayerStretch** (LayerPtr layer)
- void **SetLayerStretch** (LayerPtr layer, A_Ratio stretch)
- std::tuple< AE_TransferFlags, AE_TrackMatte > **GetLayerTransferMode** (LayerPtr layer)
- void **SetLayerTransferMode** (LayerPtr layer, AE_TransferFlags flags, AE_TrackMatte trackMatte)
- bool **IsAddLayerValid** (ItemPtr itemToAdd, CompPtr intoComp)
- LayerPtr **AddLayer** (ItemPtr itemToAdd, CompPtr intoComp)
- void **ReorderLayer** (LayerPtr layer, A_long layerIndex)
- FloatRect **GetLayerMaskedBounds** (LayerPtr layer, AE_LTimeMode timeMode, A_Time time)
- AE_ObjectType **GetLayerObjectType** (LayerPtr layer)
- bool **IsLayer3D** (LayerPtr layer)
- bool **IsLayer2D** (LayerPtr layer)
- bool **IsVideoActive** (LayerPtr layer, AE_LTimeMode timeMode, A_Time time)
- bool **IsLayerUsedAsTrackMatte** (LayerPtr layer, bool fillMustBeActive)
- bool **DoesLayerHaveTrackMatte** (LayerPtr layer)
- A_Time **ConvertCompToLayerTime** (LayerPtr layer, A_Time compTime)
- A_Time **ConvertLayerToCompTime** (LayerPtr layer, A_Time layerTime)
- A_long **GetLayerDancingRandValue** (LayerPtr layer, A_Time compTime)
- AEGP_LayerIDVal **GetLayerID** (LayerPtr layer)
- A_Matrix4 **GetLayerToWorldXform** (LayerPtr layer, A_Time compTime)
- A_Matrix4 **GetLayerToWorldXformFromView** (LayerPtr layer, A_Time viewTime, A_Time compTime)
- void **SetLayerName** (LayerPtr layer, const std::string &newName)
- LayerPtr **GetLayerParent** (LayerPtr layer)
- void **SetLayerParent** (LayerPtr layer, LayerPtr parentLayer)
- void **DeleteLayer** (LayerPtr layer)
- LayerPtr **DuplicateLayer** (LayerPtr origLayer)
- LayerPtr **GetLayerFromLayerID** (CompPtr parentComp, AEGP_LayerIDVal id)
- AEGP_LabelID **GetLayerLabel** (LayerPtr layer)
- void **SetLayerLabel** (LayerPtr layer, AEGP_LabelID label)
- AE_LayerSamplingQual **GetLayerSamplingQuality** (LayerPtr layer)
- void **SetLayerSamplingQuality** (LayerPtr layer, AE_LayerSamplingQual quality)
- LayerPtr **GetTrackMatteLayer** (LayerPtr layer)
- void **SetTrackMatte** (LayerPtr layer, LayerPtr trackMatteLayer, AE_TrackMatte trackMatteType)
- void **RemoveTrackMatte** (LayerPtr layer)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.31 MarkerDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_MarkerValP *marker)

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

- AEGP/Core/Base/AEGeneral.hpp

6.32 AE::MarkerProperty Class Reference

Public Member Functions

- **MarkerProperty** (StreamRefPtr stream)
- MarkerValPtr **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (MarkerValPtr value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.33 MarkerSuite3 Class Reference

Public Member Functions

- **MarkerSuite3** (const MarkerSuite3 &)=delete
- MarkerSuite3 & **operator=** (const MarkerSuite3 &)=delete
- **MarkerSuite3** (MarkerSuite3 &&)=delete
- MarkerSuite3 & **operator=** (MarkerSuite3 &&)=delete
- MarkerValPtr **getNewMarker** ()
- void **disposeMarker** (MarkerValPtr markerP)
- MarkerValPtr **duplicateMarker** (MarkerValPtr markerP)
- void **setMarkerFlag** (MarkerValPtr markerP, AEGP_MarkerFlagType flagType, bool valueB)
- bool **getMarkerFlag** (MarkerValPtr markerP, AEGP_MarkerFlagType flagType)

- std::string **getMarkerString** (MarkerValPtr markerP, AEGP_MarkerStringType strType)
- void **setMarkerString** (MarkerValPtr markerP, AEGP_MarkerStringType strType, const std::string &unicodeP, A_long lengthL)
- A_long **countCuePointParams** (MarkerValPtr markerP)
- std::tuple< std::string, std::string > **getIndCuePointParam** (MarkerValPtr markerP, A_long param_indexL)
- void **setIndCuePointParam** (MarkerValPtr markerP, A_long param_indexL, const std::string &unicodeKeyP, A_long key_lengthL, const std::string &unicodeValueP, A_long value_lengthL)
- void **insertCuePointParam** (MarkerValPtr markerP, A_long param_indexL)
- void **deleteIndCuePointParam** (MarkerValPtr markerP, A_long param_indexL)
- void **setMarkerDuration** (MarkerValPtr markerP, const A_Time &durationPT)
- A_Time **getMarkerDuration** (MarkerValPtr markerP)
- void **setMarkerLabel** (MarkerValPtr markerP, A_long value)
- A_long **getMarkerLabel** (MarkerValPtr markerP)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.34 MaskDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_MaskRefH *mask)

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

- AEGP/Core/Base/AEGeneral.hpp

6.35 AE::MaskIDProperty Class Reference

Public Member Functions

- **MaskIDProperty** (StreamRefPtr stream)
- A_long **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (A_long value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.36 MaskOutlineSuite3 Class Reference

Public Member Functions

- **MaskOutlineSuite3** (const MaskOutlineSuite3 &)=delete
- MaskOutlineSuite3 & **operator=** (const MaskOutlineSuite3 &)=delete
- **MaskOutlineSuite3** (MaskOutlineSuite3 &&)=delete
- MaskOutlineSuite3 & **operator=** (MaskOutlineSuite3 &&)=delete
- bool **isMaskOutlineOpen** (MaskOutlineValPtr mask_outlineH)
- void **setMaskOutlineOpen** (MaskOutlineValPtr mask_outlineH, bool openB)
- A_long **getMaskOutlineNumSegments** (MaskOutlineValPtr mask_outlineH)
- AEGP_MaskVertex **getMaskOutlineVertexInfo** (MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex which_pointL)
- void **setMaskOutlineVertexInfo** (MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex which_pointL, const AEGP_MaskVertex &vertexP)
- void **createVertex** (MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex insert_position)
- void **deleteVertex** (MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex index)
- A_long **getMaskOutlineNumFeathers** (MaskOutlineValPtr mask_outlineH)
- AEGP_MaskFeather **getMaskOutlineFeatherInfo** (MaskOutlineValPtr mask_outlineH, AEGP_FeatherIndex which_featherL)
- void **setMaskOutlineFeatherInfo** (MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex which_featherL, const AEGP_MaskFeather &featherP)
- AEGP_FeatherIndex **createMaskOutlineFeather** (MaskOutlineValPtr mask_outlineH, const AEGP_Mask↩ Feather &featherP0)
- void **deleteMaskOutlineFeather** (MaskOutlineValPtr mask_outlineH, AEGP_FeatherIndex index)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.37 AE::MaskProperty Class Reference

Public Member Functions

- **MaskProperty** (StreamRefPtr stream)
- MaskOutlineValPtr **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (MaskOutlineValPtr value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.38 MaskSuite6 Class Reference

Public Member Functions

- **MaskSuite6** (const MaskSuite6 &)=delete
- MaskSuite6 & **operator=** (const MaskSuite6 &)=delete
- **MaskSuite6** (MaskSuite6 &&)=delete
- MaskSuite6 & **operator=** (MaskSuite6 &&)=delete
- A_long **getLayerNumMasks** (LayerPtr aegp_layerH)
- MaskRefPtr **getLayerMaskByIndex** (LayerPtr aegp_layerH, AEGP_MaskIndex mask_indexL)
- void **disposeMask** (MaskRefPtr mask_refH)
- bool **getMaskInvert** (MaskRefPtr mask_refH)
- void **setMaskInvert** (MaskRefPtr mask_refH, bool invertB)
- AE_MaskMode **getMaskMode** (MaskRefPtr mask_refH)
- void **setMaskMode** (MaskRefPtr maskH, AE_MaskMode mode)
- AE_MaskMBlur **getMaskMotionBlurState** (MaskRefPtr mask_refH)
- void **setMaskMotionBlurState** (MaskRefPtr mask_refH, AE_MaskMBlur blur_state)
- AE_MaskFeatherFalloff **getMaskFeatherFalloff** (MaskRefPtr mask_refH)
- void **setMaskFeatherFalloff** (MaskRefPtr mask_refH, AE_MaskFeatherFalloff feather_falloffP)
- AEGP_MaskIDVal **getMaskID** (MaskRefPtr mask_refH)
- MaskRefPtr **createNewMask** (LayerPtr layerH, A_long mask_indexPL0)
- void **deleteMaskFromLayer** (MaskRefPtr mask_refH)

- ColorVal **getMaskColor** (MaskRefPtr mask_refH)
- void **setMaskColor** (MaskRefPtr mask_refH, ColorVal colorP)
- bool **getMaskLockState** (MaskRefPtr mask_refH)
- void **setMaskLockState** (MaskRefPtr mask_refH, bool lockB)
- bool **getMaskIsRotoBezier** (MaskRefPtr mask_refH)
- void **setMaskIsRotoBezier** (MaskRefPtr mask_refH, bool is_roto_bezierB)
- MaskRefPtr **duplicateMask** (MaskRefPtr orig_mask_refH)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.39 MemHandleDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_MemHandle *memHandle)

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

- AEGP/Core/Base/AEGeneral.hpp

6.40 MemorySuite1 Class Reference

AE Memory Suite.

```
#include <AEGeneral.hpp>
```

Public Member Functions

- **MemorySuite1** (const MemorySuite1 &)=delete
- MemorySuite1 & **operator=** (const MemorySuite1 &)=delete
- **MemorySuite1** (MemorySuite1 &&)=delete
- MemorySuite1 & **operator=** (MemorySuite1 &&)=delete
- MemHandlePtr **NewMemHandle** (const std::string &what, AEGP_MemSize size, AE_MemFlag flags)
- void **FreeMemHandle** (MemHandlePtr memHandle)
- void **LockMemHandle** (MemHandlePtr memHandle, void **ptrToPtr)
- void **UnlockMemHandle** (MemHandlePtr memHandle)
- AEGP_MemSize **GetMemHandleSize** (MemHandlePtr memHandle)
- void **ResizeMemHandle** (const std::string &what, AEGP_MemSize newSize, MemHandlePtr memHandle)
- void **SetMemReportingOn** (bool turnOn)
- std::tuple< A_long, A_long > **GetMemStats** ()

Static Public Member Functions

- static MemHandlePtr **createPtr** (AEGP_MemHandle memHandle)

6.40.1 Detailed Description

AE Memory Suite.

The Memory Suite provides access to the After Effects memory management.

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.41 AE::OneDProperty Class Reference

Public Member Functions

- **OneDProperty** (StreamRefPtr stream)
- double **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (double value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.42 OutputModuleSuite4 Class Reference

Public Member Functions

- **OutputModuleSuite4** (const OutputModuleSuite4 &)=delete
- OutputModuleSuite4 & **operator=** (const OutputModuleSuite4 &)=delete
- **OutputModuleSuite4** (OutputModuleSuite4 &&)=delete
- OutputModuleSuite4 & **operator=** (OutputModuleSuite4 &&)=delete
- OutputModuleRefPtr **getOutputModuleByIndex** (RQItemRefPtr rq_itemH, A_long outmod_indexL)
- AE_EmbeddingType **getEmbedOptions** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setEmbedOptions** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, AE_EmbeddingType embed_options)
- AE_PostRenderAction **getPostRenderAction** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setPostRenderAction** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, AE_PostRenderAction post_render_action)
- AE_OutputTypes **getEnabledOutputs** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setEnabledOutputs** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, AE_OutputTypes enabled_types)
- AE_VideoChannels **getOutputChannels** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setOutputChannels** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, AE_VideoChannels output_channels)
- std::tuple< bool, AE_StretchQuality, bool > **getStretchInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setStretchInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, bool is_enabledB, AE_StretchQuality stretch_quality)
- std::tuple< bool, A_Rect > **getCropInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setCropInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, bool enableB, A_Rect crop_rect)
- std::tuple< AEGP_SoundDataFormat, bool > **getSoundFormatInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setSoundFormatInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, AEGP_SoundDataFormat sound_format_info, bool audio_enabledB)
- std::string **getOutputFilePath** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)
- void **setOutputFilePath** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH, const std::string &path)
- OutputModuleRefPtr **addDefaultOutputModule** (RQItemRefPtr rq_itemH)
- std::tuple< std::string, std::string, bool, bool > **getExtraOutputModuleInfo** (RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.43 Param Class Reference

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

- Effect/Core/Base/AEffect.hpp

6.44 ParamConfig Class Reference

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

- Effect/Core/Base/AEffect.hpp

6.45 PlatformDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_PlatformWorldH *platform)

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

- AEGP/Core/Base/AEGeneral.hpp

6.46 Plugin Class Reference

Represents the plugin, managing its commands and lifecycle.

`#include <Plugin.hpp>`

Public Member Functions

- **Plugin** (struct SPBasicSuite *pica_basicP, AEGP_PluginID aegp_plugin_id, AEGP_GlobalRefcon *global↩ _refconV)
- virtual ~Plugin ()
- virtual void **onInit** ()=0

  onInit Initializes the plugin and its commands. Initializes the plugin and its commands. Here, you will add commands to the plugin's command list, and then use the utility functions to register your command hooks (if any).
- virtual void onDeath ()=0
- virtual void onIdle ()=0
- void addCommand (std::unique_ptr< Command > command)
- void **registerCommandHook** ()
- void **registerUpdateMenuHook** ()
- void **registerDeathHook** ()
- void **registerIdleHook** ()

Static Public Member Functions

- template<typename T >
  static A_Err **EntryPointFunc** (struct SPBasicSuite *pica_basicP, A_long major_versionL, A_long minor_↩ versionL, AEGP_PluginID aegp_plugin_id, AEGP_GlobalRefcon *global_refconV)
- static A_Err **CommandHook** (AEGP_GlobalRefcon global_refcon, AEGP_CommandRefcon command_↩ refcon, AEGP_Command command, AEGP_HookPriority hook_priority, A_Boolean already_handled, A_↩ Boolean *handled)
- static A_Err **UpdateMenuHook** (AEGP_GlobalRefcon global_refcon, AEGP_UpdateMenuRefcon update_↩ menu_refcon, AEGP_WindowType active_window)
- static A_Err **DeathHook** (AEGP_GlobalRefcon global_refcon, AEGP_DeathRefcon death_refcon)
- static A_Err **IdleHook** (AEGP_GlobalRefcon global_refcon, AEGP_IdleRefcon idle_refcon, A_long *max_↩ sleepPL)

Static Public Attributes

- static Plugin * **instance**

6.46.1 Detailed Description

Represents the plugin, managing its commands and lifecycle.

This class serves as the central management point for the plugin, handling initialization, command registration, and event hooks.

AE Refcons are ignored, as you can use maps to store data instead.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 ~Plugin()

```
virtual Plugin::~Plugin ( )  [inline], [virtual]
```

Virtual destructor for cleanup.

6.46.3 Member Function Documentation

6.46.3.1 addCommand()

```
void Plugin::addCommand (
            std::unique_ptr< Command > command )  [inline]
```

Adds a command to the plugin's command list.

Parameters

command	A unique pointer to the Command object.

6.46.3.2 onDeath()

```
virtual void Plugin::onDeath ( )  [pure virtual]
```

Called when the plugin is being unloaded. This will automatically clean up commands, its up to you to clean up anything else you need to.

6.46.3.3 onIdle()

```
virtual void Plugin::onIdle ( )  [pure virtual]
```

Called when the plugin is idle. This is a good place to do any background processing or updating of the UI. This is also where you would utilize the TaskManager to do background processing.

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

- AEGP/Template/Plugin.hpp

6.47 AE::Project Class Reference

A class representing an After Effects Project.

```
#include <Project.hpp>
```

Public Member Functions

- Project ()
  Default constructor.
- ~Project ()=default
  Destructor.
- std::string name ()
  Get the name of the project.
- std::string path ()
  Get the path of the project.
- AE_ProjBitDepth bitDepth ()
  Get the bit depth of the project.
- void setBitDepth (AE_ProjBitDepth depth)
  Set the bit depth of the project.
- void save ()
  Save the project.
- void saveAs (const std::string &path)
  Save the project to a new path.
- bool isDirty ()
  Check if the project is dirty.

Static Public Member Functions

- static Project open (const std::string &path)
  Open an After Effects Project.
- static Project newProject (const std::string &path="")
  Create a new After Effects Project.

6.47.1 Detailed Description

A class representing an After Effects Project.

This class represents an After Effects project and provides methods to interact with it.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 Project()

```
AE::Project::Project ( )  [inline]
```

Default constructor.

Initializes a new Project object by calling the init() function.

6.47.2.2 ~Project()

```
AE::Project::~Project ( )  [default]
```

Destructor.

Default destructor for the Project class.

6.47.3 Member Function Documentation

6.47.3.1 bitDepth()

```
AE_ProjBitDepth AE::Project::bitDepth ( )
```

Get the bit depth of the project.

Retrieves the bit depth of the current project.

Returns

AE_ProjBitDepth The bit depth of the project.

6.47.3.2 isDirty()

```
bool AE::Project::isDirty ( )
```

Check if the project is dirty.

Checks if the current project has unsaved changes.

Returns

bool True if the project is dirty, false otherwise.

6.47.3.3 name()

```
std::string AE::Project::name ( )
```

Get the name of the project.

Retrieves the name of the current project.

Returns

std::string The name of the project.

6.47.3.4 newProject()

```
static Project AE::Project::newProject (
            const std::string & path = "" )  [static]
```

Create a new After Effects Project.

Creates a new After Effects project and saves it to the specified path. If no path is provided, the project will be saved to the default location.

Parameters

path	The path to save the project to.

Returns

Project The newly created project.

6.47.3.5 open()

```
static Project AE::Project::open (
            const std::string & path )  [static]
```

Open an After Effects Project.

Opens an After Effects project from the specified path.

Parameters

path	The path to the project file.

Returns

Project The opened project.

6.47.3.6 path()

```
std::string AE::Project::path ( )
```

Get the path of the project.

Retrieves the path of the current project.

Returns

std::string The path of the project.

6.47.3.7 save()

```
void AE::Project::save ( )
```

Save the project.

Saves the current project.

6.47.3.8 saveAs()

```
void AE::Project::saveAs (
            const std::string & path )
```

Save the project to a new path.

Saves the current project to the specified path.

Parameters

path	The path to save the project to.

6.47.3.9 setBitDepth()

```
void AE::Project::setBitDepth (
            AE_ProjBitDepth depth )
```

Set the bit depth of the project.

Sets the bit depth of the current project.

Parameters

depth	The bit depth to set.

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

- AEGP/Core/Project.hpp

6.48 ProjSuite6 Class Reference

AE Project Suite.

```
#include <AEGeneral.hpp>
```

Public Member Functions

- **ProjSuite6** (const ProjSuite6 &)=delete
- ProjSuite6 & **operator=** (const ProjSuite6 &)=delete
- **ProjSuite6** (ProjSuite6 &&)=delete
- ProjSuite6 & **operator=** (ProjSuite6 &&)=delete
- int GetNumProjects ()

  ` *Get the number of projects in the current AE session*
- ProjectPtr **GetProjectByIndex** (A_long projIndex)
- std::string **GetProjectName** (ProjectPtr project)
- std::string **GetProjectPath** (ProjectPtr project)
- ItemPtr **GetProjectRootFolder** (ProjectPtr project)
- void **SaveProjectToPath** (ProjectPtr project, const std::string &path)
- TimeDisplay3 **GetProjectTimeDisplay** (ProjectPtr project)
- void **SetProjectTimeDisplay** (ProjectPtr project, TimeDisplay3 timeDisplay)
- bool **ProjectIsDirty** (ProjectPtr project)
- void **SaveProjectAs** (ProjectPtr project, const std::string &path)
- ProjectPtr **NewProject** ()
- ProjectPtr **OpenProjectFromPath** (const std::string &path)
- AE_ProjBitDepth **GetProjectBitDepth** (ProjectPtr project)
- void **SetProjectBitDepth** (ProjectPtr project, AE_ProjBitDepth bitDepth)

6.48.1 Detailed Description

AE Project Suite.

The Project Suite provides access to the After Effects project.

6.48.2 Member Function Documentation

6.48.2.1 GetNumProjects()

```
int ProjSuite6::GetNumProjects ( )
```

` Get the number of projects in the current AE session

@function GetNumProjects

Returns

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.49 AE::PropertyBase Class Reference

Public Member Functions

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Protected Attributes

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.50 AE::PropertyGroup Class Reference

Public Member Functions

- **PropertyGroup** (StreamRefPtr stream, AE_StreamType valType=AE_StreamType::NONE)
- virtual int **getNumProperties** () const =0
- virtual std::shared_ptr< PropertyBase > **getProperty** (int index) const =0
- virtual std::shared_ptr< PropertyBase > **getProperty** (const std::string &name) const =0
- template<typename EnumType >
  std::shared_ptr< PropertyBase > **getProperty** (EnumType name) const
- virtual std::shared_ptr< PropertyBase > **addProperty** (const std::string &name)=0
- template<typename EnumType >
  std::shared_ptr< PropertyBase > **addProperty** (EnumType name)
- virtual void **removeProperty** (const std::string &name)=0
- virtual void **removeProperty** (int index)=0
- template<typename EnumType >
  void **removeProperty** (EnumType name)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.51 RegisterSuite5 Class Reference

Public Member Functions

- **RegisterSuite5** (const RegisterSuite5 &)=delete
- RegisterSuite5 & **operator=** (const RegisterSuite5 &)=delete
- **RegisterSuite5** (RegisterSuite5 &&)=delete
- RegisterSuite5 & **operator=** (RegisterSuite5 &&)=delete
- void **registerCommandHook** (AEGP_HookPriority hook_priority, AEGP_Command command, AEGP_↩ CommandHook command_hook_func, AEGP_CommandRefcon refconP)
- void **registerUpdateMenuHook** (AEGP_UpdateMenuHook update_menu_hook_func, AEGP_Update↩ MenuRefcon refconP)
- void **registerDeathHook** (AEGP_DeathHook death_hook_func, AEGP_DeathRefcon refconP)
- void **registerIdleHook** (AEGP_IdleHook idle_hook_func, AEGP_IdleRefcon refconP)
- void **registerPresetLocalizationString** (const std::string &english_nameZ, const std::string &localized_↩ nameZ)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.52 RenderOptionsDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_RenderOptionsH *renderOptions)

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

- AEGP/Core/Base/AEGeneral.hpp

6.53 RenderOptionsSuite4 Class Reference

Public Member Functions

- **RenderOptionsSuite4** (const RenderOptionsSuite4 &)=delete
- RenderOptionsSuite4 & **operator=** (const RenderOptionsSuite4 &)=delete
- **RenderOptionsSuite4** (RenderOptionsSuite4 &&)=delete
- RenderOptionsSuite4 & **operator=** (RenderOptionsSuite4 &&)=delete
- RenderOptionsPtr **newFromItem** (ItemPtr itemH)
- RenderOptionsPtr **duplicate** (RenderOptionsPtr optionsH)
- void **setTime** (RenderOptionsPtr optionsH, A_Time time)
- A_Time **getTime** (RenderOptionsPtr optionsH)
- void **setTimeStep** (RenderOptionsPtr optionsH, A_Time time_step)
- A_Time **getTimeStep** (RenderOptionsPtr optionsH)
- void **setFieldRender** (RenderOptionsPtr optionsH, PF_Field field_render)
- PF_Field **getFieldRender** (RenderOptionsPtr optionsH)

- void **setWorldType** (RenderOptionsPtr optionsH, AE_WorldType type)
- AE_WorldType **getWorldType** (RenderOptionsPtr optionsH)
- void **setDownsampleFactor** (RenderOptionsPtr optionsH, A_short x, A_short y)
- std::tuple< A_short, A_short > **getDownsampleFactor** (RenderOptionsPtr optionsH)
- void **setRegionOfInterest** (RenderOptionsPtr optionsH, const A_LRect *roiP)
- A_LRect **getRegionOfInterest** (RenderOptionsPtr optionsH)
- void **setMatteMode** (RenderOptionsPtr optionsH, AE_MatteMode mode)
- AE_MatteMode **getMatteMode** (RenderOptionsPtr optionsH)
- void **setChannelOrder** (RenderOptionsPtr optionsH, AE_ChannelOrder channel_order)
- AE_ChannelOrder **getChannelOrder** (RenderOptionsPtr optionsH)
- bool **getRenderGuideLayers** (RenderOptionsPtr optionsH)
- void **setRenderGuideLayers** (RenderOptionsPtr optionsH, bool render_themB)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.54 RenderQueueItemSuite4 Class Reference

Public Member Functions

- **RenderQueueItemSuite4** (const RenderQueueItemSuite4 &)=delete
- RenderQueueItemSuite4 & **operator=** (const RenderQueueItemSuite4 &)=delete
- **RenderQueueItemSuite4** (RenderQueueItemSuite4 &&)=delete
- RenderQueueItemSuite4 & **operator=** (RenderQueueItemSuite4 &&)=delete
- A_long **getNumRQItems** ()
- RQItemRefPtr **getRQItemByIndex** (A_long rq_item_index)
- RQItemRefPtr **getNextRQItem** (RQItemRefPtr current_rq_item)
- A_long **getNumOutputModulesForRQItem** (RQItemRefPtr rq_item)
- AE_RenderItemStatus **getRenderState** (RQItemRefPtr rq_item)
- void **setRenderState** (RQItemRefPtr rq_item, AE_RenderItemStatus status)
- A_Time **getStartedTime** (RQItemRefPtr rq_item)
- A_Time **getElapsedTime** (RQItemRefPtr rq_item)
- AE_LogType **getLogType** (RQItemRefPtr rq_item)
- void **setLogType** (RQItemRefPtr rq_item, AE_LogType logtype)
- void **removeOutputModule** (RQItemRefPtr rq_item, OutputModuleRefPtr outmod)
- std::string **getComment** (RQItemRefPtr rq_item)
- void **setComment** (RQItemRefPtr rq_item, const std::string &comment)
- CompPtr **getCompFromRQItem** (RQItemRefPtr rq_item)
- void **deleteRQItem** (RQItemRefPtr rq_item)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.55 RenderQueueSuite1 Class Reference

Public Member Functions

- **RenderQueueSuite1** (const RenderQueueSuite1 &)=delete
- RenderQueueSuite1 & **operator=** (const RenderQueueSuite1 &)=delete
- **RenderQueueSuite1** (RenderQueueSuite1 &&)=delete
- RenderQueueSuite1 & **operator=** (RenderQueueSuite1 &&)=delete
- void **addCompToRenderQueue** (CompPtr comp, const std::string &path)
- void **setRenderQueueState** (AE_RenderQueueState state)
- AE_RenderQueueState **getRenderQueueState** ()

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

- AEGP/Core/Base/AEGeneral.hpp

6.56 RenderSuite5 Class Reference

Public Member Functions

- **RenderSuite5** (const RenderSuite5 &)=delete
- RenderSuite5 & **operator=** (const RenderSuite5 &)=delete
- **RenderSuite5** (RenderSuite5 &&)=delete
- RenderSuite5 & **operator=** (RenderSuite5 &&)=delete
- FrameReceiptPtr **renderAndCheckoutFrame** (RenderOptionsPtr optionsH)
- FrameReceiptPtr **renderAndCheckoutLayerFrame** (LayerRenderOptionsPtr optionsH)
- WorldPtr **getReceiptWorld** (FrameReceiptPtr receiptH)
- A_LRect **getRenderedRegion** (FrameReceiptPtr receiptH)
- bool **isRenderedFrameSufficient** (RenderOptionsPtr rendered_optionsH, RenderOptionsPtr proposed_↩ optionsH)
- TimeStampPtr **getCurrentTimestamp** ()
- bool **hasItemChangedSinceTimestamp** (ItemPtr itemH, A_Time start_timeP, A_Time durationP, Time↩ StampPtr time_stampP)
- bool **isItemWorthwhileToRender** (RenderOptionsPtr roH, TimeStampPtr time_stampP)
- void **checkinRenderedFrame** (RenderOptionsPtr roH, TimeStampPtr time_stampP, A_u_long ticks_to_↩ renderL, PlatformWorldPtr imageH)
- std::string **getReceiptGuid** (FrameReceiptPtr receiptH)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.57 AE::Scoped_Error_Reporter Class Reference

A class that reports errors caught within its scope.

`#include <Context.hpp>`

Public Member Functions

- **Scoped_Error_Reporter** ()=default

  Default constructor for the Scoped_Error_Reporter class.
- ~Scoped_Error_Reporter ()

  Destructor for the Scoped_Error_Reporter class.

6.57.1 Detailed Description

A class that reports errors caught within its scope.

The Scoped_Error_Reporter class is responsible for catching and reporting errors that occur within its scope. It provides a mechanism to re-throw exceptions and handle them appropriately. If an exception is caught, it can be reported as a standard or non-standard exception.

6.57.2 Constructor & Destructor Documentation

6.57.2.1 ~Scoped_Error_Reporter()

```
AE::Scoped_Error_Reporter::~Scoped_Error_Reporter ( )  [inline]
```

Destructor for the Scoped_Error_Reporter class.

The destructor attempts to re-throw any exception caught during the scope of this object. If an exception is caught, it is handled by reporting the error message. If no exception is caught, the destructor does nothing. If an error occurs while handling the exception, an optional catch block logs the error or takes other appropriate actions.

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

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

- AEGP/Util/Context.hpp

6.58 AE::Scoped_Quiet_Guard Class Reference

Public Member Functions

- Scoped_Quiet_Guard ()
- ~Scoped_Quiet_Guard ()

6.58.1 Constructor & Destructor Documentation

6.58.1.1 Scoped_Quiet_Guard()

```
AE::Scoped_Quiet_Guard::Scoped_Quiet_Guard ( )  [inline]
```

Constructs a Scoped_Quiet_Guard object and starts quiet mode for error messages.

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

6.58.1.2 ~Scoped_Quiet_Guard()

```
AE::Scoped_Quiet_Guard::~Scoped_Quiet_Guard ( )  [inline]
```

Destructs the Scoped_Quiet_Guard object and ends quiet mode for error messages.

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

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

- AEGP/Util/Context.hpp

6.59 AE::Scoped_Undo_Guard Class Reference

Public Member Functions

- Scoped_Undo_Guard (std::string name)
- ~Scoped_Undo_Guard ()

6.59.1 Constructor & Destructor Documentation

6.59.1.1 Scoped_Undo_Guard()

```
AE::Scoped_Undo_Guard::Scoped_Undo_Guard (
            std::string name )  [inline]
```

Constructs a Scoped_Undo_Guard object with the specified name.

Parameters

name	The name of the undo group.

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

6.59.1.2 ~Scoped_Undo_Guard()

```
AE::Scoped_Undo_Guard::~Scoped_Undo_Guard ( )  [inline]
```

Destructs the Scoped_Undo_Guard object and ends the undo group.

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

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

- AEGP/Util/Context.hpp

6.60 SoundDataDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_SoundDataH *soundData)

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

- AEGP/Core/Base/AEGeneral.hpp

6.61 SoundDataFormat Class Reference

AE Sound Data Format.

```
#include <AEGeneral.hpp>
```

Public Member Functions

- **SoundDataFormat** (AEGP_SoundDataFormat soundDataFormat)
- **SoundDataFormat** (double sampleRate, AE_SoundEncoding encoding, A_long bytesPerSample, A_long numChannels)
- AEGP_SoundDataFormat get () const

  Get the Sound Data Format object.
- void **set** (AEGP_SoundDataFormat soundDataFormat)
- double **getSampleRate** () const
- void **setSampleRate** (double sampleRate)
- AE_SoundEncoding **getEncoding** () const
- void **setEncoding** (AE_SoundEncoding encoding)
- A_long **getBytesPerSample** () const
- void **setBytesPerSample** (A_long bytesPerSample)
- A_long **getNumChannels** () const
- void **setNumChannels** (A_long numChannels)

6.61.1 Detailed Description

AE Sound Data Format.

6.61.2 Member Function Documentation

6.61.2.1 get()

```
AEGP_SoundDataFormat SoundDataFormat::get ( ) const  [inline]
```

Get the Sound Data Format object.

Returns

AEGP_SoundDataFormat

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

- AEGP/Core/Base/AEGeneral.hpp

6.62 SoundDataSuite1 Class Reference

Public Member Functions

- **SoundDataSuite1** (const SoundDataSuite1 &)=delete
- SoundDataSuite1 & **operator=** (const SoundDataSuite1 &)=delete
- **SoundDataSuite1** (SoundDataSuite1 &&)=delete
- SoundDataSuite1 & **operator=** (SoundDataSuite1 &&)=delete
- SoundDataPtr **NewSoundData** (const SoundDataFormat &soundFormat)
- SoundDataFormat **GetSoundDataFormat** (SoundDataPtr soundData)
- void **LockSoundDataSamples** (SoundDataPtr soundData, void ∗∗samples)
- void **UnlockSoundDataSamples** (SoundDataPtr soundData)
- int **GetNumSamples** (SoundDataPtr soundData)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.63 StreamRefDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_StreamRefH ∗stream)

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

- AEGP/Core/Base/AEGeneral.hpp

6.64 StreamSuite6 Class Reference

Public Member Functions

- **StreamSuite6** (const StreamSuite6 &)=delete
- StreamSuite6 & **operator=** (const StreamSuite6 &)=delete
- **StreamSuite6** (StreamSuite6 &&)=delete
- StreamSuite6 & **operator=** (StreamSuite6 &&)=delete
- bool **IsStreamLegal** (LayerPtr layer, AE_LayerStream whichStream)
- bool **CanVaryOverTime** (StreamRefPtr stream)
- AE_KeyInterpMask **GetValidInterpolations** (StreamRefPtr stream)
- StreamRefPtr **GetNewLayerStream** (LayerPtr layer, AE_LayerStream whichStream)
- A_long **GetEffectNumParamStreams** (EffectRefPtr effectRef)
- StreamRefPtr **GetNewEffectStreamByIndex** (EffectRefPtr effectRef, A_long paramIndex)
- StreamRefPtr **GetNewMaskStream** (MaskRefPtr maskRef, AE_MaskStream whichStream)
- std::string **GetStreamName** (StreamRefPtr stream, bool forceEnglish)
- std::string **GetStreamUnitsText** (StreamRefPtr stream, bool forceEnglish)
- std::tuple< AE_StreamFlag, double, double > **GetStreamProperties** (StreamRefPtr stream)
- bool **IsStreamTimevarying** (StreamRefPtr stream)
- AE_StreamType **GetStreamType** (StreamRefPtr stream)
- AEGP_StreamValue2 **GetNewStreamValue** (StreamRefPtr stream, AE_LTimeMode timeMode, A_Time time, bool preExpression)
- void **DisposeStreamValue** (AEGP_StreamValue2 value)
- void **SetStreamValue** (StreamRefPtr stream, AEGP_StreamValue2 value)
- std::tuple< AEGP_StreamVal2, AE_StreamType > **GetLayerStreamValue** (LayerPtr layer, AE_LayerStream whichStream, AE_LTimeMode timeMode, A_Time time, bool preExpression)
- StreamRefPtr **DuplicateStreamRef** (StreamRefPtr stream)
- int **GetUniqueStreamID** (StreamRefPtr stream)

Static Public Member Functions

- static StreamRefPtr **createPtr** (AEGP_StreamRefH streamRef)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.65 SuiteManager Class Reference

Singleton class managing the After Effects suite handler and plugin ID.

`#include <SuiteManager.hpp>`

Public Member Functions

- **SuiteManager** (SuiteManager const &)=delete
- void **operator=** (SuiteManager const &)=delete
- void InitializeSuiteHandler (SPBasicSuite *pica_basicP)

  Initializes the suite handler.
- AEGP_SuiteHandler & GetSuiteHandler ()

  Gets the suite handler.
- void SetPluginID (AEGP_PluginID *pluginIDPtr)

  Sets the plugin ID.
- AEGP_PluginID * GetPluginID () const

  Gets the plugin ID.
- **SuiteManager** (SuiteManager const &)=delete
- void **operator=** (SuiteManager const &)=delete
- void InitializeSuiteHandler (SPBasicSuite *pica_basicP)

  Initializes the suite handler.
- AEGP_SuiteHandler & GetSuiteHandler ()

  Gets the suite handler.
- void SetPluginID (AEGP_PluginID *pluginIDPtr)

  Sets the plugin ID.
- AEGP_PluginID * GetPluginID () const

  Gets the plugin ID.

Static Public Member Functions

- static SuiteManager & GetInstance ()

  Gets the singleton instance of SuiteManager.
- static SuiteManager & GetInstance ()

  Gets the singleton instance of SuiteManager.

6.65.1 Detailed Description

Singleton class managing the After Effects suite handler and plugin ID.

The SuiteManager class is responsible for managing the After Effects suite handler and plugin ID. It follows the Singleton pattern to ensure that only one instance of the class can exist. The class provides methods to initialize the suite handler, get the suite handler, set the plugin ID, and get the plugin ID.

6.65.2 Member Function Documentation

6.65.2.1 GetInstance() [1/2]

```
static SuiteManager & SuiteManager::GetInstance ( )  [inline], [static]
```

Gets the singleton instance of SuiteManager.

This method returns the singleton instance of the SuiteManager class.

Returns

SuiteManager& The reference to the singleton instance of SuiteManager.

6.65.2.2 GetInstance() [2/2]

```
static SuiteManager & SuiteManager::GetInstance ( )  [inline], [static]
```

Gets the singleton instance of SuiteManager.

This method returns the singleton instance of the SuiteManager class.

Returns

SuiteManager& The reference to the singleton instance of SuiteManager.

6.65.2.3 GetPluginID() [1/2]

```
AEGP_PluginID * SuiteManager::GetPluginID ( ) const  [inline]
```

Gets the plugin ID.

This method returns a constant pointer to the plugin ID.

Returns

const AEGP_PluginID* The constant pointer to the plugin ID.

6.65.2.4 GetPluginID() [2/2]

```
AEGP_PluginID * SuiteManager::GetPluginID ( ) const  [inline]
```

Gets the plugin ID.

This method returns a constant pointer to the plugin ID.

Returns

const AEGP_PluginID* The constant pointer to the plugin ID.

6.65.2.5 GetSuiteHandler() [1/2]

```
AEGP_SuiteHandler & SuiteManager::GetSuiteHandler ( )  [inline]
```

Gets the suite handler.

This method returns a reference to the suite handler.

Returns

AEGP_SuiteHandler& The reference to the suite handler.

6.65.2.6 GetSuiteHandler() [2/2]

```
AEGP_SuiteHandler & SuiteManager::GetSuiteHandler ( )  [inline]
```

Gets the suite handler.

This method returns a reference to the suite handler.

Returns

AEGP_SuiteHandler& The reference to the suite handler.

6.65.2.7 InitializeSuiteHandler() [1/2]

```
void SuiteManager::InitializeSuiteHandler (
            SPBasicSuite * pica_basicP )  [inline]
```

Initializes the suite handler.

This method initializes the suite handler with the provided SPBasicSuite pointer. It should be called before accessing any AE suites.

Parameters

pica_basicP	The SPBasicSuite pointer.

6.65.2.8 InitializeSuiteHandler() [2/2]

```
void SuiteManager::InitializeSuiteHandler (
            SPBasicSuite * pica_basicP )  [inline]
```

Initializes the suite handler.

This method initializes the suite handler with the provided SPBasicSuite pointer. It should be called before accessing any AE suites.

Parameters

pica_basicP	The SPBasicSuite pointer.

6.65.2.9 SetPluginID() [1/2]

```
void SuiteManager::SetPluginID (
            AEGP_PluginID * pluginIDPtr )  [inline]
```

Sets the plugin ID.

This method sets the plugin ID with the provided AEGP_PluginID pointer.

Parameters

pluginIDPtr	The AEGP_PluginID pointer.

6.65.2.10 SetPluginID() [2/2]

```
void SuiteManager::SetPluginID (
            AEGP_PluginID * pluginIDPtr )  [inline]
```

Sets the plugin ID.

This method sets the plugin ID with the provided AEGP_PluginID pointer.

Parameters

pluginIDPtr	The AEGP_PluginID pointer.

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

- AEGP/Core/Base/SuiteManager.hpp
- Effect/Core/Base/AEffectSuiteManager.hpp

6.66 TaskScheduler Class Reference

Manages the scheduling and execution of tasks.

```
#include <Task.hpp>
```

Public Member Functions

- void ScheduleTask (std::function< void()> task, bool callIdle=TRUE)
  Schedules a task with no return value.
- template<typename ReturnType >
  std::future< ReturnType > ScheduleTask (std::function< ReturnType()> task, bool callIdle=TRUE)
  Schedules a task with a return value.
- void **ExecuteTask** ()
  Executes the next scheduled task.

Static Public Member Functions

- static TaskScheduler & GetInstance ()
  Gets the singleton instance of the TaskScheduler.

6.66.1 Detailed Description

Manages the scheduling and execution of tasks.

6.66.2 Member Function Documentation

6.66.2.1 GetInstance()

```
static TaskScheduler & TaskScheduler::GetInstance ( )  [inline], [static]
```

Gets the singleton instance of the TaskScheduler.

Returns

TaskScheduler& The singleton instance.

6.66.2.2 ScheduleTask() [1/2]

```
template<typename ReturnType >
std::future< ReturnType > TaskScheduler::ScheduleTask (
            std::function< ReturnType()> task,
            bool callIdle = TRUE )  [inline]
```

Schedules a task with a return value.

Template Parameters

ReturnType	The return type of the task.

Parameters

task	The task to be scheduled.
callIdle	Flag indicating whether to call idle routines for quicker response.

Returns

std::future<ReturnType> A future object representing the result of the task.

6.66.2.3 ScheduleTask() [2/2]

```
void TaskScheduler::ScheduleTask (
            std::function< void()> task,
            bool callIdle = TRUE )  [inline]
```

Schedules a task with no return value.

Parameters

task	The task to be scheduled.
callIdle	Flag indicating whether to call idle routines for quicker response.

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

- AEGP/Util/Task.hpp

6.67 AE::TextDocumentProperty Class Reference

Public Member Functions

- **TextDocumentProperty** (StreamRefPtr stream)
- TextDocumentPtr **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (TextDocumentPtr value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.68 TextDocumentSuite1 Class Reference

Public Member Functions

- **TextDocumentSuite1** (const TextDocumentSuite1 &)=delete
- TextDocumentSuite1 & **operator=** (const TextDocumentSuite1 &)=delete
- **TextDocumentSuite1** (TextDocumentSuite1 &&)=delete
- TextDocumentSuite1 & **operator=** (TextDocumentSuite1 &&)=delete
- std::string **getNewText** (TextDocumentPtr text_documentH)
- void **setText** (TextDocumentPtr text_documentH, const std::string &unicodePS)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.69 TextLayerSuite1 Class Reference

Public Member Functions

- **TextLayerSuite1** (const TextLayerSuite1 &)=delete
- TextLayerSuite1 & **operator=** (const TextLayerSuite1 &)=delete
- **TextLayerSuite1** (TextLayerSuite1 &&)=delete
- TextLayerSuite1 & **operator=** (TextLayerSuite1 &&)=delete
- TextOutlinesPtr **getNewTextOutlines** (LayerPtr layer, const A_Time &layer_time)
- int **getNumTextOutlines** (TextOutlinesPtr outlines)
- PF_PathOutlinePtr **getIndexedTextOutline** (TextOutlinesPtr outlines, int path_index)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.70 TextOutlineDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_TextOutlinesH *memHandle)

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

- AEGP/Core/Base/AEGeneral.hpp

6.71 AE::ThreeDProperty Class Reference

Public Member Functions

- **ThreeDProperty** (StreamRefPtr stream)
- AEGP_ThreeDVal **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (AEGP_ThreeDVal value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.72 TimeDisplay3 Class Reference

Public Member Functions

- **TimeDisplay3** (AEGP_TimeDisplay3 timeDisplay)
- **TimeDisplay3** (AE_TimeDisplayMode displayMode, AE_SourceTimecodeDisplayMode footageDisplay↩ Mode, bool displayDropFrame, bool useFeetFrames, char timeBase, char framesPerFoot, AE_Frames↩ DisplayMode framesDisplayMode)
- AEGP_TimeDisplay3 **get** () const
- void **set** (AEGP_TimeDisplay3 timeDisplay)
- AEGP_TimeDisplayMode **getDisplayMode** () const
- void **setDisplayMode** (AEGP_TimeDisplayMode displayMode)
- AEGP_SourceTimecodeDisplayMode **getFootageDisplayMode** () const
- void **setFootageDisplayMode** (AEGP_SourceTimecodeDisplayMode footageDisplayMode)
- bool **getDisplayDropFrame** () const
- void **setDisplayDropFrame** (bool displayDropFrame)
- bool **getUseFeetFrames** () const
- void **setUseFeetFrames** (bool useFeetFrames)
- char **getTimeBase** () const
- void **setTimeBase** (char timeBase)
- char **getFramesPerFoot** () const
- void **setFramesPerFoot** (char framesPerFoot)
- AEGP_FramesDisplayMode **getFramesDisplayMode** () const
- void **setFramesDisplayMode** (AEGP_FramesDisplayMode framesDisplayMode)

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

- AEGP/Core/Base/AEGeneral.hpp

6.73 AE::TwoDProperty Class Reference

Public Member Functions

- **TwoDProperty** (StreamRefPtr stream)
- AEGP_TwoDVal **getValue** (AE_LTimeMode timeMode, double time=0.0) const
- void **setValue** (AEGP_TwoDVal value, AE_LTimeMode timeMode, double time=0.0)

Public Member Functions inherited from AE::PropertyBase

- **PropertyBase** (StreamRefPtr stream, AE_StreamType valType)
- std::string **getName** () const
- std::string **getUnits** () const
- bool **canAddProperty** (const std::string &name) const
- bool **isLegal** () const
- bool **isTimeVarying** () const
- bool **isHidden** () const
- bool **isElided** () const
- std::shared_ptr< PropertyGroup > **getParentGroup** () const
- std::string **getMatchName** () const
- void **setName** (const std::string &name)
- void **reorder** (int newIndex)
- void **deleteProperty** ()
- AE_StreamType **getValueType** () const

Additional Inherited Members

Protected Attributes inherited from AE::PropertyBase

- StreamRefPtr **streamRef**
- AE_StreamType **valueType**

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

- AEGP/Core/Base/Properties.hpp

6.74 UniformImage Struct Reference

Public Member Functions

- **UniformImage** (void *pData, int w, int h, int depth, size_t pitch)

Public Attributes

- void * **data**
- int **width**
- int **height**
- int **bitDepth**
- size_t **rowPitch**

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

- AEGP/Util/Image.hpp

6.75 UtilitySuite6 Class Reference

Public Member Functions

- **UtilitySuite6** (const UtilitySuite6 &)=delete
- UtilitySuite6 & **operator=** (const UtilitySuite6 &)=delete
- **UtilitySuite6** (UtilitySuite6 &&)=delete
- UtilitySuite6 & **operator=** (UtilitySuite6 &&)=delete
- void **reportInfo** (const std::string &info_string)
- void **reportInfoUnicode** (const std::string &info_string)
- std::tuple< A_short, A_short > **getDriverPluginInitFuncVersion** ()
- std::tuple< A_short, A_short > **getDriverImplementationVersion** ()
- void **startQuietErrors** ()
- void **endQuietErrors** (bool report_quieted_errorsB)
- std::string **getLastErrorMessage** (A_long buffer_size)
- void **startUndoGroup** (const std::string &undo_name)
- void **endUndoGroup** ()
- void * **getMainHWND** ()
- void **showHideAllFloaters** (bool include_tool_palB)
- ColorVal **getPaintPalForeColor** ()
- ColorVal **getPaintPalBackColor** ()
- void **setPaintPalForeColor** (const ColorVal &fore_color)
- void **setPaintPalBackColor** (const ColorVal &back_color)
- std::tuple< bool, ColorVal > **getCharPalFillColor** ()
- std::tuple< bool, ColorVal > **getCharPalStrokeColor** ()
- void **setCharPalFillColor** (const ColorVal &fill_color)
- void **setCharPalStrokeColor** (const ColorVal &stroke_color)
- bool **charPalIsFillColorUIFrontmost** ()
- A_Ratio **convertFpLongToHSFRatio** (A_FpLong numberF)
- A_FpLong **convertHSFRatioToFpLong** (A_Ratio ratioR)
- void **causeIdleRoutinesToBeCalled** ()
- bool **getSuppressInteractiveUI** ()
- void **writeToOSConsole** (const std::string &text)
- void **writeToDebugLog** (const std::string &subsystem, const std::string &eventType, const std::string &text)
- std::string **getPluginPath** (AE_PluginPathType path_type)

6.75.1 Detailed Description

Examples

C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp.

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

6.76 WorldDeleter Class Reference

Public Member Functions

- void **operator()** (AEGP_WorldH *world)

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

- AEGP/Core/Base/AEGeneral.hpp

6.77 WorldSuite3 Class Reference

Public Member Functions

- **WorldSuite3** (const WorldSuite3 &)=delete
- WorldSuite3 & **operator=** (const WorldSuite3 &)=delete
- **WorldSuite3** (WorldSuite3 &&)=delete
- WorldSuite3 & **operator=** (WorldSuite3 &&)=delete
- WorldPtr **newWorld** (AE_WorldType type, A_long widthL, A_long heightL)
- AE_WorldType **getType** (WorldPtr worldH)
- std::tuple< A_long, A_long > **getSize** (WorldPtr worldH)
- A_u_long **getRowBytes** (WorldPtr worldH)
- PF_Pixel8 * **getBaseAddr8** (WorldPtr worldH)
- PF_Pixel16 * **getBaseAddr16** (WorldPtr worldH)
- PF_PixelFloat * **getBaseAddr32** (WorldPtr worldH)
- PF_EffectWorld **fillOutPFEffectWorld** (WorldPtr worldH)
- void **fastBlur** (A_FpLong radiusF, PF_ModeFlags mode, PF_Quality quality, WorldPtr worldH)
- PlatformWorldPtr **newPlatformWorld** (AEGP_WorldType type, A_long widthL, A_long heightL)
- WorldPtr **newReferenceFromPlatformWorld** (PlatformWorldPtr platform_worldH)

Static Public Member Functions

- static WorldPtr **createPtr** (AEGP_WorldH ref)
- static PlatformWorldPtr **createPlatformPtr** (AEGP_PlatformWorldH ref)

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

- AEGP/Core/Base/AEGeneral.hpp
- AEGP/Core/Base/AEGeneral.cpp

Chapter 7

File Documentation

7.1 AEGP/AEGP.hpp File Reference

Include all AEGP headers.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
#include "AETK/AEGP/Core/Base/Import.hpp"
#include "AETK/AEGP/Core/Base/Item.hpp"
#include "AETK/AEGP/Core/Base/Layer.hpp"
#include "AETK/AEGP/Core/Base/Properties.hpp"
#include "AETK/AEGP/Core/Base/SuiteManager.hpp"
#include "AETK/AEGP/Core/Base/Collection.hpp"
#include "AETK/AEGP/Core/Project.hpp"
#include "AETK/AEGP/Core/App.hpp"
#include "AETK/AEGP/Core/Comp.hpp"
#include "AETK/AEGP/Core/Effects.hpp"
#include "AETK/AEGP/Memory/AEAllocator.hpp"
#include "AETK/AEGP/Memory/AEMemory.hpp"
#include "AETK/AEGP/Exception/Exception.hpp"
#include "AETK/AEGP/Util/Context.hpp"
#include "AETK/AEGP/Util/Task.hpp"
#include "AETK/AEGP/Util/Image.hpp"
#include "AETK/AEGP/Template/Plugin.hpp"
```

Macros

- #define **AEGP_HPP**
- #define **SDK_VERSION** 2023

7.1.1 Detailed Description

Include all AEGP headers.

Author

tjerf

Date

March 2024

7.2 AEGP.hpp

Go to the documentation of this file.

```
00001 /*****************************************************************//**
00002  * \file   AEGP.hpp
00003  * \brief  Include all AEGP headers
00004  *
00005  * \author tjerf
00006  * \date   March 2024
00007  *********************************************************************/
00008 #pragma once
00009
00010 #ifndef AEGP_HPP
00011 #define AEGP_HPP
00012
00013 #define SDK_VERSION 2023
00014
00015 // General Imports
00016 #include "AETK/AEGP/Core/Base/AEGeneral.hpp" //Wrapper around all AEGP Suite headers
00017 #include "AETK/AEGP/Core/Base/Import.hpp" // Utility Wrappers for importing files
00018 #include "AETK/AEGP/Core/Base/Item.hpp" // Wrapper for AEGP_ItemH
00019 #include "AETK/AEGP/Core/Base/Layer.hpp" // Wrapper for AEGP_LayerH
00020 #include "AETK/AEGP/Core/Base/Properties.hpp" // Base Wrapper for AEGP_StreamRefH
00021 #include "AETK/AEGP/Core/Base/SuiteManager.hpp" // Wrapper for AEGP_SuiteHandler and AEGP_PluginID
00022 #include "AETK/AEGP/Core/Base/Collection.hpp" // Wrapper for Collections of Items (not using
      collection suites)
00023
00024 //Mid-Level Wrapper imports
00025 #include "AETK/AEGP/Core/Project.hpp" // Wrapper for AEGP_ProjectH
00026 #include "AETK/AEGP/Core/App.hpp" //Wrapper for misc AEGP funcs related to application.
00027 #include "AETK/AEGP/Core/Comp.hpp" //Wrapper for AEGP_CompH (Derived from AEGP_ItemH)
00028 #include "AETK/AEGP/Core/Effects.hpp" //Wrapper for AEGP_EffectRefH, uses Properties.hpp
00029
00030 //Memory Related Imports
00031 #include "AETK/AEGP/Memory/AEAllocator.hpp" //custom allocator for AEGP to use with any STL containers
00032 #include "AETK/AEGP/Memory/AEMemory.hpp" //pre-defined "AE::Memory" namespace for AEGP containers.
      "AE::vector" etc.
00033
00034 //Exception Imports
00035 #include "AETK/AEGP/Exception/Exception.hpp" //Custom Exception class for AEGP
00036
00037 //Utility Imports
00038 #include "AETK/AEGP/Util/Context.hpp" //Scoped Context Managers for AEGP Calls
00039 #include "AETK/AEGP/Util/Task.hpp" // Threading utility for calling and executing AEGP with multiple
      threads.
00040 #include "AETK/AEGP/Util/Image.hpp" // Image utility for AEGP
00041
00042 //Template Imports
00043 #include "AETK/AEGP/Template/Plugin.hpp" //Template class for designing plugins
00044
00045 #endif // !AEGP_HPP
00046
```

7.3 AEGP/Core/App.hpp File Reference

A class to interact with the After Effects application Provides utility functions to interact with the application, such as Alerting the user, getting the main window, getting the current project or application path, and getting or setting the color of the paint or text palette.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AE::App

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.3.1 Detailed Description

A class to interact with the After Effects application Provides utility functions to interact with the application, such as Alerting the user, getting the main window, getting the current project or application path, and getting or setting the color of the paint or text palette.

Author

tjerf

Date

March 2024

7.4 App.hpp

Go to the documentation of this file.

```
00001 /*********************************************************************//**
00002  * \file   App.hpp
00003  * \brief  A class to interact with the After Effects application
00004  * Provides utility functions to interact with the application, such as
00005  * Alerting the user, getting the main window, getting the current project
00006  * or application path, and getting or setting the color of the paint or
00007  * text palette.
00008  *
00009  * \author tjerf
00010  * \date   March 2024
00011  *********************************************************************/
00012
00013 #pragma once
00014
00015 #ifndef APP_HPP
00016 #define APP_HPP
00017
00018 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00019
00020 namespace AE
00021 {
00022
00023 class App
00024 {
00025   public:
00026     App() = default;
00027     virtual ~App() = default;
00028
00029     /**
00030      * @brief
00031      * Alert the user with a message
00032      *
00033      * @param data
00034      * The message to display
00035      */
00036     void Alert(std::any data) const
00037     {
00038         try
00039         {
00040             if (data.has_value())
00041             {
00042                 UtilitySuite6().reportInfo(std::any_cast<std::string>(data));
00043             }
00044         }
00045         catch (const std::bad_any_cast &e)
00046         {
00047             throw e;
00048         }
00049         catch (const AEException &e)
00050         {
00051             throw AEException(e);
00052         }
00053     }
00054
00055     /**
00056      * @brief
```

```
00057      * Get the main window (HWND) for AE
00058      *
00059      * @return
00060      * The main window as a void pointer, cast to platform specific type
00061      */
00062     void *GetWindow() const
00063     {
00064         try
00065         {
00066             void *window = UtilitySuite6().getMainHWND();
00067             return window;
00068         }
00069         catch (const AEException &e)
00070         {
00071             throw e;
00072         }
00073     }
00074
00075     /**
00076      * @brief Get the current project file path
00077      *
00078      * \return
00079      * The file path as a string
00080      */
00081     std::string UserPluginPath()
00082     {
00083         try
00084         {
00085             return UtilitySuite6().getPluginPath(
00086                 AE_PluginPathType::USER_PLUGIN);
00087         }
00088         catch (const AEException &e)
00089         {
00090             throw e;
00091         }
00092     }
00093
00094     /**
00095      * @brief Get the path to the folder containing plugins.
00096      *
00097      * \return
00098      * Folder path as a string
00099      */
00100     std::string AllPluginPath()
00101     {
00102         try
00103         {
00104             return UtilitySuite6().getPluginPath(
00105                 AE_PluginPathType::ALLUSER_PLUGIN);
00106         }
00107         catch (const AEException &e)
00108         {
00109             throw e;
00110         }
00111     }
00112
00113     /**
00114      * @brief Get the path to the AE application
00115      *
00116      * \return
00117      * Application path as a string
00118      */
00119     std::string AppPath()
00120     {
00121         try
00122         {
00123             return UtilitySuite6().getPluginPath(AE_PluginPathType::APP);
00124         }
00125         catch (const AEException &e)
00126         {
00127             throw e;
00128         }
00129     }
00130
00131     /**
00132      * @brief Get the Color of the Paint Palette
00133      *
00134      * @param useForeground
00135      * - If true, get the foreground color, else get the background color
00136      *
00137      * \return
00138      * ColorVal
00139      */
00140     ColorVal BrushColor(bool useForeground)
00141     {
00142         try
00143         {
```

```
00144             if (useForeground)
00145             {
00146                 return UtilitySuite6().getPaintPalForeColor();
00147             }
00148             else
00149             {
00150                 return UtilitySuite6().getPaintPalBackColor();
00151             }
00152         }
00153         catch (const AEException &e)
00154         {
00155             throw e;
00156         }
00157     }
00158
00159     /**
00160      * @brief Set the color of the Paint Palette
00161      *
00162      * @param color
00163      * The color to set
00164      * @param useForeground
00165      * - If true, set the foreground color, else set the background color
00166      */
00167     void SetBrushColor(ColorVal color, bool useForeground)
00168     {
00169         try
00170         {
00171             if (useForeground)
00172             {
00173                 UtilitySuite6().setPaintPalForeColor(color);
00174             }
00175             else
00176             {
00177                 UtilitySuite6().setPaintPalBackColor(color);
00178             }
00179         }
00180         catch (const AEException &e)
00181         {
00182             throw e;
00183         }
00184     }
00185     // no function abailable to swap color being used in paint palette
00186     /**
00187      * @brief Get the current color of the Character Palette, based on which is
00188      * in the front.
00189      *
00190      * @return
00191      * ColorVal
00192      */
00193     ColorVal CharColor()
00194     {
00195         try
00196         {
00197             if (UtilitySuite6().charPalIsFillColorUIFrontmost())
00198             {
00199                 return std::get<1>(UtilitySuite6().getCharPalFillColor());
00200             }
00201             else
00202             {
00203                 return std::get<1>(UtilitySuite6().getCharPalStrokeColor());
00204             }
00205         }
00206         catch (const AEException &e)
00207         {
00208             throw e;
00209         }
00210     }
00211
00212     /**
00213      * @brief Set the color of the Character Palette
00214      *
00215      * @param color
00216      * The color to set
00217      * @param useFill
00218      * - If true, set the fill color, else set the stroke color
00219      */
00220     void CharColor(ColorVal color, bool useFill)
00221     {
00222         try
00223         {
00224             if (useFill)
00225             {
00226                 UtilitySuite6().setCharPalFillColor(color);
00227             }
00228             else
00229             {
00230                 UtilitySuite6().setCharPalStrokeColor(color);
```

```
00231             }
00232         }
00233         catch (const AEException &e)
00234         {
00235             throw e;
00236         }
00237     }
00238 };
00239
00240 } // namespace AE
00241
00242 #endif // APP_HPP
```

7.5 AEGP/Core/Base/AEGeneral.hpp File Reference

General functions and types for After Effects SDK, built by wrapping AE_GeneralPlug.h.

```
#include "Headers/AEConfig.h"
#include "AETK/AEGP/Core/Base/SuiteManager.hpp"
#include "AETK/AEGP/Exception/Exception.hpp"
#include "Headers/AE_GeneralPlug.h"
#include "Util/entry.h"
#include <any>
#include <functional>
#include <future>
#include <iostream>
#include <memory>
#include <mutex>
#include <optional>
#include <queue>
#include <string>
#include <tuple>
#include <unicode/unistr.h>
#include <unordered_map>
#include <variant>
#include <vector>
#include <map>
```

Classes

- class StreamRefDeleter
- class MarkerDeleter
- class WorldDeleter
- class PlatformDeleter
- class EffectDeleter
- class FootageDeleter
- class MaskDeleter
- class RenderOptionsDeleter
- class LayerRenderOptionsDeleter
- class MemHandleDeleter
- class TextOutlineDeleter
- class AddKeyframesInfoDeleter
- class CollectionDeleter
- class FrameReceiptDeleter
- class MemorySuite1

  AE Memory Suite.

- class TimeDisplay3
- class ProjSuite6

  AE Project Suite.
- class SoundDataFormat

  AE Sound Data Format.
- class ItemSuite9

  AE Item Suite.
- class ItemViewSuite1
- class SoundDataDeleter
- class SoundDataSuite1
- class CompSuite11
- class LayerSuite9
- class StreamSuite6
- class DynamicStreamSuite4
- class KeyframeSuite5
- class TextDocumentSuite1
- class MarkerSuite3
- class TextLayerSuite1
- class EffectSuite4
- class MaskSuite6
- class MaskOutlineSuite3
- class FootageSuite5
- class UtilitySuite6
- class RenderQueueSuite1
- class RenderQueueItemSuite4
- class OutputModuleSuite4
- class WorldSuite3
- class RenderOptionsSuite4
- class LayerRenderOptionsSuite2
- class RenderSuite5
- class CollectionSuite2
- class RegisterSuite5
- class CommandSuite1

Macros

- #define AE_CHECK(expr)
- #define **FOOTAGE_MAIN_FILE_INDEX** 0
- #define **INSERT_SORTED** (-2)
- #define **INSERT_BOTTOM** (-1)
- #define **INSERT_TOP** 0

Typedefs

- typedef std::shared_ptr< AEGP_ProjectH > **ProjectPtr**

  Define shared pointers for After Effects SDK types.
- typedef std::shared_ptr< AEGP_ItemH > **ItemPtr**
- typedef std::shared_ptr< AEGP_CompH > **CompPtr**
- typedef std::shared_ptr< AEGP_FootageH > **FootagePtr**
- typedef std::shared_ptr< AEGP_LayerH > **LayerPtr**
- typedef std::shared_ptr< AEGP_EffectRefH > **EffectRefPtr**
- typedef std::shared_ptr< AEGP_MaskRefH > **MaskRefPtr**

- typedef std::shared_ptr< AEGP_StreamRefH > **StreamRefPtr**
- typedef std::shared_ptr< AEGP_RenderLayerContextH > **RenderLayerContextPtr**
- typedef std::shared_ptr< AEGP_PersistentBlobH > **PersistentBlobPtr**
- typedef std::shared_ptr< AEGP_MaskOutlineValH > **MaskOutlineValPtr**
- typedef std::shared_ptr< AEGP_CollectionH > **CollectionPtr**
- typedef std::shared_ptr< AEGP_Collection2H > **Collection2Ptr**
- typedef std::shared_ptr< AEGP_SoundDataH > **SoundDataPtr**
- typedef std::shared_ptr< AEGP_AddKeyframesInfoH > **AddKeyframesInfoPtr**
- typedef std::shared_ptr< AEGP_RenderReceiptH > **RenderReceiptPtr**
- typedef std::shared_ptr< AEGP_WorldH > **WorldPtr**
- typedef std::shared_ptr< AEGP_RenderOptionsH > **RenderOptionsPtr**
- typedef std::shared_ptr< AEGP_LayerRenderOptionsH > **LayerRenderOptionsPtr**
- typedef std::shared_ptr< AEGP_FrameReceiptH > **FrameReceiptPtr**
- typedef std::shared_ptr< AEGP_RQItemRefH > **RQItemRefPtr**
- typedef std::shared_ptr< AEGP_OutputModuleRefH > **OutputModuleRefPtr**
- typedef std::shared_ptr< AEGP_TextDocumentH > **TextDocumentPtr**
- typedef std::shared_ptr< AEGP_MarkerValP > **MarkerValPtr**
- typedef std::shared_ptr< AEGP_TextOutlinesH > **TextOutlinesPtr**
- typedef std::shared_ptr< AEGP_PlatformWorldH > **PlatformWorldPtr**
- typedef std::shared_ptr< AEGP_ItemViewP > **ItemViewPtr**
- typedef std::shared_ptr< AEGP_ColorProfileP > **ColorProfilePtr**
- typedef std::shared_ptr< AEGP_ConstColorProfileP > **ConstColorProfilePtr**
- typedef std::shared_ptr< AEGP_TimeStamp > **TimeStampPtr**
- typedef std::shared_ptr< AEGP_StreamValue2 > **StreamValue2Ptr**
- typedef std::shared_ptr< AEGP_MemHandle > **MemHandlePtr**
- typedef std::tuple< double, double, double, double > **ColorVal**

  AE Color Profiles.
- typedef std::tuple< double, double, double, double > **FloatRect**
- typedef std::tuple< double, double > **AE_KeyframeEase**
- using StreamVal

Enumerations

- enum class **AE_MemFlag** { **NONE** = AEGP_MemFlag_NONE , **CLEAR** = AEGP_MemFlag_CLEAR , **QUIET** = AEGP_MemFlag_QUIET }
- enum class AE_Platform { **WIN** = AEGP_Platform_WIN , **MAC** = AEGP_Platform_MAC }

  AE Platforms.
- enum class **AE_ProjBitDepth** { **_8** = AEGP_ProjBitDepth_8 , **_16** = AEGP_ProjBitDepth_16 , **_32** = AEGP_ProjBitDepth_32 , **NUM_VALID_DEPTHS** = AEGP_ProjBitDepth_NUM_VALID_DEPTHS }
- enum class **AE_CameraType** { **NONE** = AEGP_CameraType_NONE , **PERSPECTIVE** = AEGP_CameraType_PERSPECTIVE , **ORTHOGRAPHIC** = AEGP_CameraType_ORTHOGRAPHIC , **NUM_TYPES** = AEGP_CameraType_NUM_TYPES }
- enum class **AE_TimeDisplayType** { **TIMECODE** = AEGP_TimeDisplayType_TIMECODE , **FRAMES** = AEGP_TimeDisplayType_FRAMES , **FEET_AND_FRAMES** = AEGP_TimeDisplayType_FEET_AND_FRAMES }
- enum class **AE_FilmSizeUnits** { **NONE** = AEGP_FilmSizeUnits_NONE , **HORIZONTAL** = AEGP_FilmSizeUnits_HORIZONTAL , **VERTICAL** = AEGP_FilmSizeUnits_VERTICAL , **DIAGONAL** = AEGP_FilmSizeUnits_DIAGONAL }
- enum class **AE_LightType** {
  NONE = AEGP_LightType_NONE , **PARALLEL** = AEGP_LightType_PARALLEL , **SPOT** = AEGP_LightType_SPOT , **POINT** = AEGP_LightType_POINT ,
  AMBIENT = AEGP_LightType_AMBIENT , **RESERVED1** = AEGP_LightType_RESERVED1 , **NUM_TYPES** = AEGP_LightType_NUM_TYPES }

- enum class **AE_LightFalloffType** { **NONE** = AEGP_LightFalloff_NONE , **SMOOTH** = AEGP_LightFalloff_↩ SMOOTH , **INVERSE_SQUARE_CLAMPED** = AEGP_LightFalloff_INVERSE_SQUARE_CLAMPED }
- enum class **AE_FootageDepth** {
  _1 = AEGP_FootageDepth_1 , **_2** = AEGP_FootageDepth_2 , **_4** = AEGP_FootageDepth_4 , **_8** = AEGP↩ _FootageDepth_8 ,
  _16 = AEGP_FootageDepth_16 , **_24** = AEGP_FootageDepth_24 , **_30** = AEGP_FootageDepth_30 , **_32** = AEGP_FootageDepth_32 ,
  GRAY_2 = AEGP_FootageDepth_GRAY_2 , **GRAY_4** = AEGP_FootageDepth_GRAY_4 , **GRAY_8** = AEGP_FootageDepth_GRAY_8 , **_48** = AEGP_FootageDepth_48 ,
  _64 = AEGP_FootageDepth_64 , **GRAY_16** = AEGP_FootageDepth_GRAY_16 }
- enum class **AE_FramesPerFoot** { **_35MM** = AEGP_FramesPerFoot_35MM , **_16MM** = AEGP_FramesPer↩ Foot_16MM }
- enum class **AE_TimeDisplayMode** { **TIMECODE** = AEGP_TimeDisplay_TIMECODE , **FRAMES** = AEGP↩ _TimeDisplay_FRAMES }
- enum class **AE_SourceTimecodeDisplayMode** { **ZERO** = AEGP_SourceTimecode_ZERO , **SOURCE_**↩ **TIMECODE** = AEGP_SourceTimecode_SOURCE_TIMECODE }
- enum class **AE_FramesDisplayMode** { **ZERO_BASED** = AEGP_Frames_ZERO_BASED , **ONE_BASED** = AEGP_Frames_ONE_BASED , **TIMECODE_CONVERSION** = AEGP_Frames_TIMECODE_CONVERSION }
- enum class **AE_SoundEncoding** {
  UNSIGNED_PCM = AEGP_SoundEncoding_UNSIGNED_PCM , **SIGNED_PCM** = AEGP_SoundEncoding↩ _SIGNED_PCM , **FLOAT** = AEGP_SoundEncoding_FLOAT , **END** = AEGP_SoundEncoding_END ,
  BEGIN = AEGP_SoundEncoding_BEGIN }
- enum class **AE_ItemType** {
  NONE = AEGP_ItemType_NONE , **FOLDER** = AEGP_ItemType_FOLDER , **COMP** = AEGP_ItemType_↩ COMP , **SOLID** = AEGP_ItemType_SOLID_defunct ,
  FOOTAGE = AEGP_ItemType_FOOTAGE , **NUM_TYPES** = AEGP_ItemType_NUM_TYPES1 }
- enum class **AE_ItemFlag** {
  MISSING = AEGP_ItemFlag_MISSING , **HAS_PROXY** = AEGP_ItemFlag_HAS_PROXY , **USING_PROXY** = AEGP_ItemFlag_USING_PROXY , **MISSING_PROXY** = AEGP_ItemFlag_MISSING_PROXY ,
  HAS_VIDEO = AEGP_ItemFlag_HAS_VIDEO , **HAS_AUDIO** = AEGP_ItemFlag_HAS_AUDIO , **STILL** = AEGP_ItemFlag_STILL , **HAS_ACTIVE_AUDIO** = AEGP_ItemFlag_HAS_ACTIVE_AUDIO }
- enum class **AE_Label** {
  NONE = AEGP_Label_NONE , **NO_LABEL** = AEGP_Label_NO_LABEL , **LABEL_1** = AEGP_Label_1 , **LABEL_2** = AEGP_Label_2 ,
  LABEL_3 = AEGP_Label_3 , **LABEL_4** = AEGP_Label_4 , **LABEL_5** = AEGP_Label_5 , **LABEL_6** = AEGP_Label_6 ,
  LABEL_7 = AEGP_Label_7 , **LABEL_8** = AEGP_Label_8 , **LABEL_9** = AEGP_Label_9 , **LABEL_10** = AEGP_Label_10 ,
  LABEL_11 = AEGP_Label_11 , **LABEL_12** = AEGP_Label_12 , **LABEL_13** = AEGP_Label_13 , **LABEL_14** = AEGP_Label_14 ,
  LABEL_15 = AEGP_Label_15 , **LABEL_16** = AEGP_Label_16 , **NUM_TYPES** = AEGP_Label_NUMTYPES }
- enum class **AE_PersistentType** {
  MACHINE_SPECIFIC = AEGP_PersistentType_MACHINE_SPECIFIC , **MACHINE_INDEPENDENT** = AEGP_PersistentType_MACHINE_INDEPENDENT , **MACHINE_INDEPENDENT_RENDER** = AEGP_↩ PersistentType_MACHINE_INDEPENDENT_RENDER , **MACHINE_INDEPENDENT_OUTPUT** = AEGP_↩ PersistentType_MACHINE_INDEPENDENT_OUTPUT ,
  MACHINE_INDEPENDENT_COMPOSITION , **MACHINE_SPECIFIC_TEXT** = AEGP_PersistentType↩ _MACHINE_SPECIFIC_TEXT , **MACHINE_SPECIFIC_PAINT** = AEGP_PersistentType_MACHINE_↩ SPECIFIC_PAINT , **MACHINE_SPECIFIC_EFFECTS** = AEGP_PersistentType_MACHINE_SPECIFIC_↩ EFFECTS ,
  MACHINE_SPECIFIC_EXPRESSION_SNIPPETS , **MACHINE_SPECIFIC_SCRIPT_SNIPPETS** , **NUM_**↩ **TYPES** = AEGP_PersistentType_NUMTYPES }
- enum class **AE_CompFlag** {
  SHOW_ALL_SHY = AEGP_CompFlag_SHOW_ALL_SHY , **ENABLE_MOTION_BLUR** , **ENABLE_TIME**↩ **_FILTER** , **GRID_TO_FRAMES** = AEGP_CompFlag_GRID_TO_FRAMES ,

GRID_TO_FIELDS = AEGP_CompFlag_GRID_TO_FIELDS , **USE_LOCAL_DSF** = AEGP_CompFlag_↩ USE_LOCAL_DSF , **DRAFT_3D** = AEGP_CompFlag_DRAFT_3D , **SHOW_GRAPH** = AEGP_CompFlag↩ _SHOW_GRAPH }

- enum class **AE_TransferFlags** { **PRESERVE_ALPHA** = AEGP_TransferFlag_PRESERVE_ALPHA , **RANDOMIZE_DISSOLVE** = AEGP_TransferFlag_RANDOMIZE_DISSOLVE }
- enum class **AE_TrackMatte** {
  NO_TRACK_MATTE = AEGP_TrackMatte_NO_TRACK_MATTE , **ALPHA** = AEGP_TrackMatte_ALPHA , **NOT_ALPHA** = AEGP_TrackMatte_NOT_ALPHA , **LUMA** = AEGP_TrackMatte_LUMA , **NOT_LUMA** = AEGP_TrackMatte_NOT_LUMA }
- enum class **AE_LayerQual** { **NONE** = AEGP_LayerQual_NONE , **WIREFRAME** = AEGP_LayerQual_↩ WIREFRAME , **DRAFT** = AEGP_LayerQual_DRAFT , **BEST** = AEGP_LayerQual_BEST }
- enum class **AE_LayerSamplingQual** { **BILINEAR** = AEGP_LayerSamplingQual_BILINEAR , **BICUBIC** = AEGP_LayerSamplingQual_BICUBIC }
- enum class **AE_LayerFlag** {
  NONE = AEGP_LayerFlag_NONE , **VIDEO_ACTIVE** = AEGP_LayerFlag_VIDEO_ACTIVE , **AUDIO_ACTIVE** = AEGP_LayerFlag_AUDIO_ACTIVE , **EFFECTS_ACTIVE** = AEGP_LayerFlag_EFFECTS_ACTIVE , **MOTION_BLUR** = AEGP_LayerFlag_MOTION_BLUR , **FRAME_BLENDING** = AEGP_LayerFlag_FRAME↩ _BLENDING , **LOCKED** = AEGP_LayerFlag_LOCKED , **SHY** = AEGP_LayerFlag_SHY , **COLLAPSE** = AEGP_LayerFlag_COLLAPSE , **AUTO_ORIENT_ROTATION** = AEGP_LayerFlag_AUTO↩ _ORIENT_ROTATION , **ADJUSTMENT_LAYER** = AEGP_LayerFlag_ADJUSTMENT_LAYER , **TIME_↩ REMAPPING** = AEGP_LayerFlag_TIME_REMAPPING , **LAYER_IS_3D** = AEGP_LayerFlag_LAYER_IS_3D , **LOOK_AT_CAMERA** = AEGP_LayerFlag_LOOK_AT↩ _CAMERA , **LOOK_AT_POI** = AEGP_LayerFlag_LOOK_AT_POI , **SOLO** = AEGP_LayerFlag_SOLO , **MARKERS_LOCKED** = AEGP_LayerFlag_MARKERS_LOCKED , **NULL_LAYER** = AEGP_LayerFlag_↩ NULL_LAYER , **HIDE_LOCKED_MASKS** = AEGP_LayerFlag_HIDE_LOCKED_MASKS , **GUIDE_LAYER** = AEGP_LayerFlag_GUIDE_LAYER , **ADVANCED_FRAME_BLENDING** = AEGP_LayerFlag_ADVANCED_FRAME_BLENDING , **SUBLAYERS↩ _RENDER_SEPARATELY** = AEGP_LayerFlag_SUBLAYERS_RENDER_SEPARATELY , **ENVIRONMENT↩ _LAYER** = AEGP_LayerFlag_ENVIRONMENT_LAYER }
- enum class **AE_ObjectType** {
  NONE = AEGP_ObjectType_NONE , **AV** = AEGP_ObjectType_AV , **LIGHT** = AEGP_ObjectType_LIGHT , **CAMERA** = AEGP_ObjectType_CAMERA , **TEXT** = AEGP_ObjectType_TEXT , **VECTOR** = AEGP_ObjectType_VECTOR , **RESERVED1** = AEGP_↩ ObjectType_RESERVED1 , **RESERVED2** = AEGP_ObjectType_RESERVED2 , **RESERVED3** = AEGP_ObjectType_RESERVED3 , **RESERVED4** = AEGP_ObjectType_RESERVED4 , **RESERVED5** = AEGP_ObjectType_RESERVED5 , **NUM_TYPES** = AEGP_ObjectType_NUM_TYPES }
- enum class **AE_LTimeMode** { **LayerTime** = AEGP_LTimeMode_LayerTime , **CompTime** = AEGP_LTime↩ Mode_CompTime }
- enum class **AE_LayerStream** {
  ANCHORPOINT = AEGP_LayerStream_ANCHORPOINT , **POSITION** = AEGP_LayerStream_POSITION , **SCALE** = AEGP_LayerStream_SCALE , **ROTATION** = AEGP_LayerStream_ROTATION , **ROTATE_Z** = AEGP_LayerStream_ROTATE_Z , **OPACITY** = AEGP_LayerStream_OPACITY , **AUDIO** = AEGP_LayerStream_AUDIO , **MARKER** = AEGP_LayerStream_MARKER , **TIME_REMAP** = AEGP_LayerStream_TIME_REMAP , **ROTATE_X** = AEGP_LayerStream_ROTATE_X , **ROTATE_Y** = AEGP_LayerStream_ROTATE_Y , **ORIENTATION** = AEGP_LayerStream_ORIENTATION , **ZOOM** = AEGP_LayerStream_ZOOM , **DEPTH_OF_FIELD** = AEGP_LayerStream_DEPTH_OF_FIELD , **FOCUS_DISTANCE** = AEGP_LayerStream_FOCUS_DISTANCE , **APERTURE** = AEGP_LayerStream_↩ APERTURE , **BLUR_LEVEL** = AEGP_LayerStream_BLUR_LEVEL , **IRIS_SHAPE** = AEGP_LayerStream_IRIS_SHAPE , **IRIS_ROTATION** = AEGP_LayerStream_IRIS_ROTATION , **IRIS_ROUNDNESS** = AEGP_LayerStream_↩ IRIS_ROUNDNESS , **IRIS_ASPECT_RATIO** = AEGP_LayerStream_IRIS_ASPECT_RATIO , **IRIS_DIFFRACTION_FRINGE** = AEGP_LayerStream_IRIS_DIFFRACTION_FRINGE , **IRIS_HIGHLIGHT_GAIN** = AEGP_LayerStream_↩ IRIS_HIGHLIGHT_GAIN , **IRIS_HIGHLIGHT_THRESHOLD** = AEGP_LayerStream_IRIS_HIGHLIGHT_↩ THRESHOLD , **IRIS_HIGHLIGHT_SATURATION** = AEGP_LayerStream_IRIS_HIGHLIGHT_SATURATION , **INTENSITY** = AEGP_LayerStream_INTENSITY , **COLOR** = AEGP_LayerStream_COLOR , **CONE_ANGLE** = AEGP_↩ LayerStream_CONE_ANGLE ,

CONE_FEATHER = AEGP_LayerStream_CONE_FEATHER , **SHADOW_DARKNESS** = AEGP_Layer↩ Stream_SHADOW_DARKNESS , **SHADOW_DIFFUSION** = AEGP_LayerStream_SHADOW_DIFFUSION , **LIGHT_FALLOFF_TYPE** = AEGP_LayerStream_LIGHT_FALLOFF_TYPE ,
LIGHT_FALLOFF_START = AEGP_LayerStream_LIGHT_FALLOFF_START , **LIGHT_FALLOFF_↩ DISTANCE** = AEGP_LayerStream_LIGHT_FALLOFF_DISTANCE , **ACCEPTS_SHADOWS** = AEGP_↩ LayerStream_ACCEPTS_SHADOWS , **ACCEPTS_LIGHTS** = AEGP_LayerStream_ACCEPTS_LIGHTS ,
AMBIENT_COEFF = AEGP_LayerStream_AMBIENT_COEFF , **DIFFUSE_COEFF** = AEGP_Layer↩ Stream_DIFFUSE_COEFF , **SPECULAR_INTENSITY** = AEGP_LayerStream_SPECULAR_INTENSITY , **SPECULAR_SHININESS** = AEGP_LayerStream_SPECULAR_SHININESS ,
CASTS_SHADOWS = AEGP_LayerStream_CASTS_SHADOWS , **LIGHT_TRANSMISSION** = AEGP↩ _LayerStream_LIGHT_TRANSMISSION , **METAL** = AEGP_LayerStream_METAL , **REFLECTION_↩ INTENSITY** = AEGP_LayerStream_REFLECTION_INTENSITY ,
REFLECTION_SHARPNESS = AEGP_LayerStream_REFLECTION_SHARPNESS , **REFLECTION_↩ ROLLOFF** = AEGP_LayerStream_REFLECTION_ROLLOFF , **TRANSPARENCY_COEFF** = AEGP↩ _LayerStream_TRANSPARENCY_COEFF , **TRANSPARENCY_ROLLOFF** = AEGP_LayerStream_↩ TRANSPARENCY_ROLLOFF ,
INDEX_OF_REFRACTION = AEGP_LayerStream_INDEX_OF_REFRACTION , **EXTRUSION_BEVEL↩ _STYLE** = AEGP_LayerStream_EXTRUSION_BEVEL_STYLE , **EXTRUSION_BEVEL_DIRECTION** = AEGP_LayerStream_EXTRUSION_BEVEL_DIRECTION , **EXTRUSION_BEVEL_DEPTH** = AEGP_Layer↩ Stream_EXTRUSION_BEVEL_DEPTH ,
EXTRUSION_HOLE_BEVEL_DEPTH = AEGP_LayerStream_EXTRUSION_HOLE_BEVEL_DEPTH , **EXTRUSION_DEPTH** = AEGP_LayerStream_EXTRUSION_DEPTH , **PLANE_CURVATURE** = AEGP↩ _LayerStream_PLANE_CURVATURE , **PLANE_SUBDIVISION** = AEGP_LayerStream_PLANE_↩ SUBDIVISION }

- enum class **AE_MaskStream** { **OUTLINE** = AEGP_MaskStream_OUTLINE , **OPACITY** = AEGP_Mask↩ Stream_OPACITY , **FEATHER** = AEGP_MaskStream_FEATHER , **EXPANSION** = AEGP_MaskStream_↩ EXPANSION }
- enum class **AE_StreamFlag** { **NONE** = AEGP_StreamFlag_NONE , **HAS_MIN** = AEGP_StreamFlag_HAS↩ _MIN , **HAS_MAX** = AEGP_StreamFlag_HAS_MAX , **IS_SPATIAL** = AEGP_StreamFlag_IS_SPATIAL }
- enum class **AE_KeyInterp** { **NONE** = AEGP_KeyInterp_NONE , **LINEAR** = AEGP_KeyInterp_LINEAR , **BEZIER** = AEGP_KeyInterp_BEZIER , **HOLD** = AEGP_KeyInterp_HOLD }
- enum class **AE_KeyInterpMask** {
  NONE = AEGP_KeyInterpMask_NONE , **LINEAR** = AEGP_KeyInterpMask_LINEAR , **BEZIER** = AEGP_↩ KeyInterpMask_BEZIER , **HOLD** = AEGP_KeyInterpMask_HOLD ,
  CUSTOM = AEGP_KeyInterpMask_CUSTOM , **ANY** = AEGP_KeyInterpMask_ANY }
- enum class **AE_StreamType** {
  NONE = AEGP_StreamType_NO_DATA , **ThreeD_SPATIAL** = AEGP_StreamType_ThreeD_SPATIAL , **ThreeD** = AEGP_StreamType_ThreeD , **TwoD_SPATIAL** = AEGP_StreamType_TwoD_SPATIAL ,
  TwoD = AEGP_StreamType_TwoD , **OneD** = AEGP_StreamType_OneD , **COLOR** = AEGP_StreamType_↩ COLOR , **ARB** = AEGP_StreamType_ARB ,
  MARKER = AEGP_StreamType_MARKER , **LAYER_ID** = AEGP_StreamType_LAYER_ID , **MASK_ID** = AEGP_StreamType_MASK_ID , **MASK** = AEGP_StreamType_MASK ,
  TEXT_DOCUMENT = AEGP_StreamType_TEXT_DOCUMENT }
- enum class **AE_StreamGroupingType** { **NONE** = AEGP_StreamGroupingType_NONE , **LEAF** = AEGP↩ _StreamGroupingType_LEAF , **NAMED_GROUP** = AEGP_StreamGroupingType_NAMED_GROUP , **INDEXED_GROUP** = AEGP_StreamGroupingType_INDEXED_GROUP }
- enum class **AE_DynStreamFlag** {
  ACTIVE_EYEBALL = AEGP_DynStreamFlag_ACTIVE_EYEBALL , **HIDDEN** = AEGP_DynStreamFlag_↩ HIDDEN , **DISABLED** = AEGP_DynStreamFlag_DISABLED , **ELIDED** = AEGP_DynStreamFlag_ELIDED ,
  SHOWN_WHEN_EMPTY = AEGP_DynStreamFlag_SHOWN_WHEN_EMPTY , **SKIP_REVEAL_WHEN_↩ UNHIDDEN** = AEGP_DynStreamFlag_SKIP_REVEAL_WHEN_UNHIDDEN }
- enum class **AE_KeyframeFlag** {
  NONE = AEGP_KeyframeFlag_NONE , **TEMPORAL_CONTINUOUS** = AEGP_KeyframeFlag_↩ TEMPORAL_CONTINUOUS , **TEMPORAL_AUTOBEZIER** = AEGP_KeyframeFlag_TEMPORAL_↩ AUTOBEZIER , **SPATIAL_CONTINUOUS** = AEGP_KeyframeFlag_SPATIAL_CONTINUOUS ,
  SPATIAL_AUTOBEZIER = AEGP_KeyframeFlag_SPATIAL_AUTOBEZIER , **ROVING** = AEGP_Keyframe↩ Flag_ROVING }

- enum class **AE_EffectFlags** {
  NONE = AEGP_EffectFlags_NONE , **ACTIVE** = AEGP_EffectFlags_ACTIVE , **AUDIO_ONLY** = AEGP_↩
  EffectFlags_AUDIO_ONLY , **AUDIO_TOO** = AEGP_EffectFlags_AUDIO_TOO ,
  MISSING = AEGP_EffectFlags_MISSING }
- enum class **AE_MaskMBlur** { **SAME_AS_LAYER** = AEGP_MaskMBlur_SAME_AS_LAYER , **OFF** = AEGP↩
  _MaskMBlur_OFF , **ON** = AEGP_MaskMBlur_ON }
- enum class **AE_MaskFeatherFalloff** { **SMOOTH** = AEGP_MaskFeatherFalloff_SMOOTH , **LINEAR** = AEGP_MaskFeatherFalloff_LINEAR }
- enum class **AE_MaskFeatherInterp** { **NORMAL** = AEGP_MaskFeatherInterp_NORMAL , **HOLD_CW** = AEGP_MaskFeatherInterp_HOLD_CW }
- enum class **AE_MaskFeatherType** { **OUTER** = AEGP_MaskFeatherType_OUTER , **INNER** = AEGP_Mask↩
  FeatherType_INNER }
- enum class **AE_MaskMode** {
  NONE = PF_MaskMode_NONE , **ADD** = PF_MaskMode_ADD , **SUBTRACT** = PF_MaskMode_SUBTRACT , **INTERSECT** = PF_MaskMode_INTERSECT ,
  LIGHTEN = PF_MaskMode_LIGHTEN , **DARKEN** = PF_MaskMode_DARKEN , **DIFF** = PF_MaskMode_↩
  DIFFERENCE , **ACCUM** = PF_MaskMode_ACCUM }
- enum class **AE_AlphaFlags** { **PREMUL** = AEGP_AlphaPremul , **INVERTED** = AEGP_AlphaInverted , **ALPHA_IGNORE** = AEGP_AlphaIgnore }
- enum class **AE_PulldownPhase** {
  NO_PULLDOWN = AEGP_PulldownPhase_NO_PULLDOWN , **WSSWW** = AEGP_PulldownPhase_↩
  WSSWW , **SSWWW** = AEGP_PulldownPhase_SSWWW , **SWWWS** = AEGP_PulldownPhase_SWWWS ,
  WWWSS = AEGP_PulldownPhase_WWWSS , **WWSSW** = AEGP_PulldownPhase_WWSSW , **WWWSW** = AEGP_PulldownPhase_WWWSW , **WWSWW** = AEGP_PulldownPhase_WWSWW ,
  WSWWW = AEGP_PulldownPhase_WSWWW , **SWWWW** = AEGP_PulldownPhase_SWWWW , **WWWWS** = AEGP_PulldownPhase_WWWWS }
- enum class **AE_LayerDrawStyle** { **LAYER_BOUNDS** = AEGP_LayerDrawStyle_LAYER_BOUNDS , **DOCUMENT_BOUNDS** = AEGP_LayerDrawStyle_DOCUMENT_BOUNDS }
- enum class **AE_InterpretationStyle** { **NO_DIALOG_GUESS** = AEGP_InterpretationStyle_NO_DIALOG_↩
  GUESS , **DIALOG_OK** = AEGP_InterpretationStyle_DIALOG_OK , **NO_DIALOG_NO_GUESS** = AEGP_↩
  InterpretationStyle_NO_DIALOG_NO_GUESS }
- enum class **AE_PluginPathType** { **PLUGIN** = AEGP_GetPathTypes_PLUGIN , **USER_PLUGIN** = AEGP_↩
  GetPathTypes_USER_PLUGIN , **ALLUSER_PLUGIN** = AEGP_GetPathTypes_ALLUSER_PLUGIN , **APP** = AEGP_GetPathTypes_APP }
- enum class **AE_RenderQueueState** { **STOPPED** = AEGP_RenderQueueState_STOPPED , **PAUSED** = AEGP_RenderQueueState_PAUSED , **RENDERING** = AEGP_RenderQueueState_RENDERING }
- enum class **AE_RenderItemStatus** {
  NONE = AEGP_RenderItemStatus_NONE , **WILL_CONTINUE** = AEGP_RenderItemStatus_WILL_↩
  CONTINUE , **NEEDS_OUTPUT** = AEGP_RenderItemStatus_NEEDS_OUTPUT , **UNQUEUED** = AEGP_↩
  RenderItemStatus_UNQUEUED ,
  QUEUED = AEGP_RenderItemStatus_QUEUED , **RENDERING** = AEGP_RenderItemStatus_RENDERING , **USER_STOPPED** = AEGP_RenderItemStatus_USER_STOPPED , **ERR_STOPPED** = AEGP_Render↩
  ItemStatus_ERR_STOPPED ,
  DONE = AEGP_RenderItemStatus_DONE }
- enum class **AE_LogType** { **NONE** = AEGP_LogType_NONE , **ERRORS_ONLY** = AEGP_LogType_↩
  ERRORS_ONLY , **PLUS_SETTINGS** = AEGP_LogType_PLUS_SETTINGS , **PER_FRAME_INFO** = AEGP↩
  _LogType_PER_FRAME_INFO }
- enum class **AE_EmbeddingType** { **NONE** = AEGP_Embedding_NONE , **NOTHING** = AEGP_Embedding_↩
  NOTHING , **LINK** = AEGP_Embedding_LINK , **LINK_AND_COPY** = AEGP_Embedding_LINK_AND_COPY }
- enum class **AE_PostRenderAction** { **NONE** = AEGP_PostRenderOptions_NONE , **IMPORT** = AEGP_Post↩
  RenderOptions_IMPORT , **IMPORT_AND_REPLACE_USAGE** = AEGP_PostRenderOptions_IMPORT_↩
  AND_REPLACE_USAGE , **SET_PROXY** = AEGP_PostRenderOptions_SET_PROXY }
- enum class **AE_OutputTypes** { **NONE** = AEGP_OutputType_NONE , **VIDEO** = AEGP_OutputType_VIDEO , **AUDIO** = AEGP_OutputType_AUDIO }
- enum class **AE_VideoChannels** { **NONE** = AEGP_VideoChannels_NONE , **RGB** = AEGP_VideoChannels↩
  _RGB , **RGBA** = AEGP_VideoChannels_RGBA , **ALPHA** = AEGP_VideoChannels_ALPHA }

- enum class **AE_StretchQuality** { **NONE** = AEGP_StretchQual_NONE , **LOW** = AEGP_StretchQual_LOW , **HIGH** = AEGP_StretchQual_HIGH }
- enum class **AE_OutputColorType** { **STRAIGHT** = AEGP_OutputColorType_STRAIGHT , **PREMUL** = AEGP_OutputColorType_PREMUL }
- enum class **AE_WorldType** { **NONE** = AEGP_WorldType_NONE , **W8** = AEGP_WorldType_8 , **W16** = AEGP_WorldType_16 , **W32** = AEGP_WorldType_32 }
- enum class **AE_MatteMode** { **STRAIGHT** = AEGP_MatteMode_STRAIGHT , **PREMUL_BLACK** = AEGP↩_MatteMode_PREMUL_BLACK , **PREMUL_BG_COLOR** = AEGP_MatteMode_PREMUL_BG_COLOR }
- enum class **AE_ChannelOrder** { **ARGB** = AEGP_ChannelOrder_ARGB , **BGRA** = AEGP_ChannelOrder↩_BGRA }
- enum class **AE_ItemQuality** { **DRAFT** = AEGP_ItemQuality_DRAFT , **BEST** = AEGP_ItemQuality_BEST }
- enum class **AE_CollectionItemType** {
  NONE = AEGP_CollectionItemType_NONE , **LAYER** = AEGP_CollectionItemType_LAYER , **MASK** = AEGP_CollectionItemType_MASK , **EFFECT** = AEGP_CollectionItemType_EFFECT ,
  STREAM = AEGP_CollectionItemType_STREAM , **KEYFRAME** = AEGP_CollectionItemType_KEYFRAME ,
  MASK_VERTEX = AEGP_CollectionItemType_MASK_VERTEX , **STREAMREF** = AEGP_CollectionItem↩Type_STREAMREF }
- enum class **AE_StreamCollectionItemType** { **NONE** = AEGP_StreamCollectionItemType_NONE , **LAYER** = AEGP_StreamCollectionItemType_LAYER , **MASK** = AEGP_StreamCollectionItemType_MASK , **EFFECT** = AEGP_StreamCollectionItemType_EFFECT }
- enum class **AE_WindowType** {
  NONE = AEGP_WindType_NONE , **PROJECT** = AEGP_WindType_PROJECT , **COMP** = AEGP_Wind↩Type_COMP , **TIME_LAYOUT** = AEGP_WindType_TIME_LAYOUT ,
  LAYER = AEGP_WindType_LAYER , **FOOTAGE** = AEGP_WindType_FOOTAGE , **RENDER_QUEUE** = AEGP_WindType_RENDER_QUEUE , **QT** = AEGP_WindType_QT ,
  DIALOG = AEGP_WindType_DIALOG , **FLOWCHART** = AEGP_WindType_FLOWCHART , **EFFECT** = AEGP_WindType_EFFECT , **OTHER** = AEGP_WindType_OTHER }
- enum class **AE_MenuID** {
  NONE = AEGP_Menu_NONE , **APPLE** = AEGP_Menu_APPLE , **FILE** = AEGP_Menu_FILE , **EDIT** = AEGP_Menu_EDIT ,
  COMPOSITION = AEGP_Menu_COMPOSITION , **LAYER** = AEGP_Menu_LAYER , **EFFECT** = AEGP_↩Menu_EFFECT , **WINDOW** = AEGP_Menu_WINDOW ,
  FLOATERS = AEGP_Menu_FLOATERS , **KF_ASSIST** = AEGP_Menu_KF_ASSIST , **IMPORT** = AEGP_↩Menu_IMPORT , **SAVE_FRAME_AS** = AEGP_Menu_SAVE_FRAME_AS ,
  PREFS = AEGP_Menu_PREFS , **EXPORT** = AEGP_Menu_EXPORT , **ANIMATION** = AEGP_Menu_↩ANIMATION , **PURGE** = AEGP_Menu_PURGE ,
  NEW = AEGP_Menu_NEW }

Functions

- double **TimeToSeconds** (const A_Time &time)
- double **ToFrames** (const A_Time &time, const A_Time &frameRate)
- double **ToFrames** (double time, double frameRate)
- A_Time **SecondsToTime** (double seconds)
- A_Time **FramesToTime** (double frames, const A_Time &frameRate)
- A_Time **FramesToTime** (double frames, double frameRate)
- std::string **GetErrorMessage** (int errorCode)
- std::string **ConvertUTF16ToUTF8** (const A_UTF16Char ∗utf16String)
- std::vector< UChar > **ConvertUTF8ToUTF16UnSafe** (const std::string &utf8String)
- std::vector< A_UTF16Char > **ConvertUTF8ToUTF16** (const std::string &utf8String)
- std::string **memHandleToString** (AEGP_MemHandle memHandle)
- ColorVal toColorVal (const AEGP_ColorVal &color)

  Convert AE ColorVal to ColorVal.
- AEGP_ColorVal toAEGP_ColorVal (const ColorVal &color)

  Convert ColorVal to AE ColorVal.

- AEGP_DownsampleFactor toAEGP_DownsampleFactor (const std::tuple< A_short, A_short > &factor)
  AE Footage Suite.
- std::tuple< short, short > toDownsampleFactor (const AEGP_DownsampleFactor &factor)
  Convert AEGP_DownsampleFactor to tuple.
- StreamRefPtr **toStreamRefPtr** (AEGP_StreamRefH streamRef)
- A_FloatRect **toA_FloatRect** (const std::tuple< double, double, double, double > &rect)
- std::tuple< double, double, double, double > **toFloatRect** (const A_FloatRect &rect)
- AEGP_TwoDVal **toAEGP_TwoDVal** (const std::tuple< double, double > &val)
- std::tuple< double, double > **toTwoDVal** (const AEGP_TwoDVal &val)
- AEGP_ThreeDVal **toAEGP_ThreeDVal** (const std::tuple< double, double, double > &val)
- std::tuple< double, double, double > **toThreeDVal** (const AEGP_ThreeDVal &val)
- AEGP_KeyframeEase **toAEGP_KeyframeEase** (const std::tuple< double, double > &val)
- std::tuple< double, double > **toKeyframeEase** (const AEGP_KeyframeEase &val)
- StreamVal **CreateStream** (AEGP_StreamValue2 val)
- AEGP_MaskFeather **createAEGP_MaskFeather** ()
- std::tuple< A_long, PF_FpLong, PF_FpLong, PF_FpShort, PF_FpShort, AEGP_MaskFeatherInterp, AEGP_MaskFeatherType > **getAEGP_MaskFeatherInfo** (const AEGP_MaskFeather &feather)
- AEGP_AlphaLabel **createAEGP_AlphaLabel** ()
- std::tuple< AEGP_AlphaFlags, A_u_char, A_u_char, A_u_char > **getAEGP_AlphaLabelInfo** (const AEGP_AlphaLabel &label)
- AEGP_LoopBehavior **createAEGP_LoopBehavior** ()
- std::tuple< A_long, A_long > **getAEGP_LoopBehaviorInfo** (const AEGP_LoopBehavior &behavior)
- AEGP_FootageLayerKey **createAEGP_FootageLayerKey** ()
- std::tuple< A_long, A_long, std::string, AEGP_LayerDrawStyle > **getAEGP_FootageLayerKeyInfo** (const AEGP_FootageLayerKey &key)
- AEGP_FileSequenceImportOptions **createAEGP_FileSequenceImportOptions** ()
- std::tuple< bool, bool, A_long, A_long > **getAEGP_FileSequenceImportOptionsInfo** (const AEGP_File↩ SequenceImportOptions &options)

7.5.1 Detailed Description

General functions and types for After Effects SDK, built by wrapping AE_GeneralPlug.h.

AEGeneral.hpp

Author

tjerf

Date

March 2024

7.5.2 Macro Definition Documentation

7.5.2.1 AE_CHECK

```
#define AE_CHECK(
                expr )
```

Value:

```
    do                                                                  \
    {                                                                   \
        A_Err err = (expr);                                             \
        if (err != A_Err_NONE)                                          \
        {                                                               \
            std::string errorMessage = GetErrorMessage(err);            \
            throw AEException(errorMessage.c_str());                    \
        }                                                               \
    } while (0)
```

7.5.3 Typedef Documentation

7.5.3.1 StreamVal

```
using StreamVal
```

Initial value:

```
    std::variant<AEGP_OneDVal, AEGP_TwoDVal, AEGP_ThreeDVal, AEGP_ColorVal,
                 MarkerValPtr, A_long, MaskOutlineValPtr, TextDocumentPtr>
```

7.5.4 Function Documentation

7.5.4.1 toAEGP_ColorVal()

```
AEGP_ColorVal toAEGP_ColorVal (
             const ColorVal & color )  [inline]
```

Convert ColorVal to AE ColorVal.

Parameters

color	

Returns

AEGP_ColorVal

7.5.4.2 toAEGP_DownsampleFactor()

```
AEGP_DownsampleFactor toAEGP_DownsampleFactor (
             const std::tuple< A_short, A_short > & factor )  [inline]
```

AE Footage Suite.

The Footage Suite provides access to the After Effects project footage.

7.5.4.3 toColorVal()

```
ColorVal toColorVal (
             const AEGP_ColorVal & color )  [inline]
```

Convert AE ColorVal to ColorVal.

Parameters

color	

Returns

ColorVal

7.5.4.4 toDownsampleFactor()

```
std::tuple< short, short > toDownsampleFactor (
            const AEGP_DownsampleFactor & factor )  [inline]
```

Convert AEGP_DownsampleFactor to tuple.

Parameters

factor	

Returns

std::tuple<A_short, A_short>

7.6 AEGeneral.hpp

Go to the documentation of this file.

```
00001 /*********************************************************************/ /**
00002                                                                        * \file
00003                                                                        *AEGeneral.hpp
00004                                                                        * \brief
00005                                                                        *General
00006                                                                        *functions
00007                                                                        *and types
00008                                                                        *for After
00009                                                                        *Effects
00010                                                                        *SDK, built
00011                                                                        *by
00012                                                                        *wrapping
00013                                                                        * AE_GeneralPlug.h
00014                                                                        *
00015                                                                        * \author
00016                                                                        *tjerf
00017                                                                        * \date
00018                                                                        *March 2024
00019
     **********************************************************************/
00020 #ifndef AE_MAIN_HPP
00021 #define AE_MAIN_HPP
00022
00023 #include "Headers/AEConfig.h"
00024
00025
00026 #ifdef AE_OS_WIN
00027 #include <windows.h>
00028 #endif
00029
00030 #include "AETK/AEGP/Core/Base/SuiteManager.hpp"
00031 #include "AETK/AEGP/Exception/Exception.hpp"
00032 #include "Headers/AE_GeneralPlug.h"
00033 #include "Util/entry.h"
00034
00035 #include <any>
00036 #include <functional>
00037 #include <future>
00038 #include <iostream>
00039 #include <memory>
00040 #include <mutex>
00041 #include <optional>
00042 #include <queue>
00043 #include <string>
00044 #include <tuple>
00045 #include <unicode/unistr.h>
```

```
00046 #include <unordered_map>
00047 #include <variant>
00048 #include <vector>
00049 #include <map>
00050
00051 double TimeToSeconds(
00052     const A_Time &time); // Will find active comp and convert using frame rate
00053 double ToFrames(const A_Time &time, const A_Time &frameRate);
00054 double ToFrames(double time, double frameRate);
00055 A_Time SecondsToTime(double seconds);
00056 A_Time FramesToTime(double frames, const A_Time &frameRate);
00057 A_Time FramesToTime(double frames, double frameRate);
00058
00059 inline std::string GetErrorMessage(int errorCode)
00060 {
00061     switch (errorCode)
00062     {
00063     case A_Err_NONE:
00064         return "No error occurred.";
00065     case A_Err_GENERIC:
00066         return "A generic error occurred.";
00067     case A_Err_STRUCT:
00068         return "Structural error, possibly in plugin or project configuration.";
00069     case A_Err_PARAMETER:
00070         return "Parameter error: One or more parameters are invalid.";
00071     case A_Err_ALLOC:
00072         return "Allocation error: Failed to allocate necessary resources.";
00073     case A_Err_WRONG_THREAD:
00074         return "Wrong thread error: Attempted to execute a thread-specific "
00075                "operation on the wrong thread.";
00076     case A_Err_CONST_PROJECT_MODIFICATION:
00077         return "Constant project modification error: Attempted to modify a "
00078                "read-only project.";
00079     case A_Err_MISSING_SUITE:
00080         return "Missing suite error: Failed to acquire a required suite.";
00081     case A_Err_NOT_IN_CACHE_OR_COMPUTE_PENDING:
00082         return "Data not in cache or compute pending: Requested data is not "
00083                "available and is either being computed or must be "
00084                "re-requested.";
00085     case A_Err_PROJECT_LOAD_FATAL:
00086         return "Fatal project load error: Unable to load the project due to a "
00087                "critical error.";
00088     case A_Err_EFFECT_APPLY_FATAL:
00089         return "Fatal effect application error: Applying the effect resulted "
00090                "in "
00091                "a critical error.";
00092     default:
00093         // Handle reserved errors generically, as specifics are unknown
00094         if (errorCode >= A_Err_RESERVED_7 && errorCode <= A_Err_RESERVED_21)
00095         {
00096             return "Reserved error: An unspecified error occurred. Please "
00097                    "consult "
00098                    "the documentation or contact support.";
00099         }
00100         return "Unknown error: An unrecognized error occurred. Please consult "
00101                "the documentation or contact support.";
00102     }
00103 }
00104 // Check for error and throw exception
00105 #define AE_CHECK(expr)                                                  \
00106     do                                                                  \
00107     {                                                                   \
00108         A_Err err = (expr);                                             \
00109         if (err != A_Err_NONE)                                          \
00110         {                                                               \
00111             std::string errorMessage = GetErrorMessage(err);            \
00112             throw AEException(errorMessage.c_str());                    \
00113         }                                                               \
00114     } while (0)
00115
00116 /**
00117  * @brief Define shared pointers for After Effects SDK types
00118  *
00119  */
00120 typedef std::shared_ptr<AEGP_ProjectH> ProjectPtr;
00121 typedef std::shared_ptr<AEGP_ItemH> ItemPtr;
00122 typedef std::shared_ptr<AEGP_CompH> CompPtr;
00123 typedef std::shared_ptr<AEGP_FootageH> FootagePtr;
00124 typedef std::shared_ptr<AEGP_LayerH> LayerPtr;
00125 typedef std::shared_ptr<AEGP_EffectRefH> EffectRefPtr;
00126 typedef std::shared_ptr<AEGP_MaskRefH> MaskRefPtr;
00127 typedef std::shared_ptr<AEGP_StreamRefH> StreamRefPtr;
00128 typedef std::shared_ptr<AEGP_RenderLayerContextH> RenderLayerContextPtr;
00129 typedef std::shared_ptr<AEGP_PersistentBlobH> PersistentBlobPtr;
00130 typedef std::shared_ptr<AEGP_MaskOutlineValH> MaskOutlineValPtr;
00131 typedef std::shared_ptr<AEGP_CollectionH> CollectionPtr;
00132 typedef std::shared_ptr<AEGP_Collection2H> Collection2Ptr;
```

```
00133 typedef std::shared_ptr<AEGP_SoundDataH> SoundDataPtr;
00134 typedef std::shared_ptr<AEGP_AddKeyframesInfoH> AddKeyframesInfoPtr;
00135 typedef std::shared_ptr<AEGP_RenderReceiptH> RenderReceiptPtr;
00136 typedef std::shared_ptr<AEGP_WorldH> WorldPtr;
00137 typedef std::shared_ptr<AEGP_RenderOptionsH> RenderOptionsPtr;
00138 typedef std::shared_ptr<AEGP_LayerRenderOptionsH> LayerRenderOptionsPtr;
00139 typedef std::shared_ptr<AEGP_FrameReceiptH> FrameReceiptPtr;
00140 typedef std::shared_ptr<AEGP_RQItemRefH> RQItemRefPtr;
00141 typedef std::shared_ptr<AEGP_OutputModuleRefH> OutputModuleRefPtr;
00142 typedef std::shared_ptr<AEGP_TextDocumentH> TextDocumentPtr;
00143 typedef std::shared_ptr<AEGP_MarkerValP> MarkerValPtr;
00144 typedef std::shared_ptr<AEGP_TextOutlinesH> TextOutlinesPtr;
00145 typedef std::shared_ptr<AEGP_PlatformWorldH> PlatformWorldPtr;
00146 typedef std::shared_ptr<AEGP_ItemViewP> ItemViewPtr;
00147 typedef std::shared_ptr<AEGP_ColorProfileP> ColorProfilePtr;
00148 typedef std::shared_ptr<AEGP_ConstColorProfileP> ConstColorProfilePtr;
00149 typedef std::shared_ptr<AEGP_TimeStamp> TimeStampPtr;
00150
00151 typedef std::shared_ptr<AEGP_MarkerValP> MarkerValPtr;
00152 typedef std::shared_ptr<AEGP_StreamValue2> StreamValue2Ptr;
00153 typedef std::shared_ptr<AEGP_MemHandle> MemHandlePtr;
00154 typedef std::shared_ptr<AEGP_TimeStamp> TimeStampPtr;
00155
00156 class StreamRefDeleter
00157 {
00158   public:
00159     void operator()(AEGP_StreamRefH *stream)
00160     {
00161         if (stream && *stream)
00162         {
00163             SuiteManager::GetInstance()
00164                 .GetSuiteHandler()
00165                 .StreamSuite2()
00166                 ->AEGP_DisposeStream(*stream);
00167         }
00168     }
00169 };
00170
00171 class MarkerDeleter
00172 {
00173   public:
00174     void operator()(AEGP_MarkerValP *marker)
00175     {
00176         if (marker && *marker)
00177         {
00178             SuiteManager::GetInstance()
00179                 .GetSuiteHandler()
00180                 .MarkerSuite3()
00181                 ->AEGP_DisposeMarker(*marker);
00182         }
00183     }
00184 };
00185
00186 class WorldDeleter
00187 {
00188   public:
00189     void operator()(AEGP_WorldH *world)
00190     {
00191         if (world && *world)
00192         {
00193             SuiteManager::GetInstance()
00194                 .GetSuiteHandler()
00195                 .WorldSuite3()
00196                 ->AEGP_Dispose(*world);
00197         }
00198     }
00199 };
00200
00201 class PlatformDeleter
00202 {
00203   public:
00204     void operator()(AEGP_PlatformWorldH *platform)
00205     {
00206         if (platform && *platform)
00207         {
00208             SuiteManager::GetInstance()
00209                 .GetSuiteHandler()
00210                 .WorldSuite3()
00211                 ->AEGP_DisposePlatformWorld(*platform);
00212         }
00213     }
00214 };
00215
00216 class EffectDeleter
00217 {
00218   public:
00219     void operator()(AEGP_EffectRefH *effect)
```

```
00220     {
00221         if (effect && *effect)
00222         {
00223             SuiteManager::GetInstance()
00224                 .GetSuiteHandler()
00225                 .EffectSuite4()
00226                 ->AEGP_DisposeEffect(*effect);
00227         }
00228     }
00229 };
00230
00231 class FootageDeleter
00232 {
00233   public:
00234     void operator()(AEGP_FootageH *footage)
00235     {
00236         if (footage && *footage)
00237         {
00238             SuiteManager::GetInstance()
00239                 .GetSuiteHandler()
00240                 .FootageSuite5()
00241                 ->AEGP_DisposeFootage(*footage);
00242         }
00243     }
00244 };
00245
00246 class MaskDeleter
00247 {
00248   public:
00249     void operator()(AEGP_MaskRefH *mask)
00250     {
00251         if (mask && *mask)
00252         {
00253             SuiteManager::GetInstance()
00254                 .GetSuiteHandler()
00255                 .MaskSuite6()
00256                 ->AEGP_DisposeMask(*mask);
00257         }
00258     }
00259 };
00260
00261 class RenderOptionsDeleter
00262 {
00263   public:
00264     void operator()(AEGP_RenderOptionsH *renderOptions)
00265     {
00266         if (renderOptions && *renderOptions)
00267         {
00268             SuiteManager::GetInstance()
00269                 .GetSuiteHandler()
00270                 .RenderOptionsSuite3()
00271                 ->AEGP_Dispose(*renderOptions);
00272         }
00273     }
00274 };
00275
00276 class LayerRenderOptionsDeleter
00277 {
00278   public:
00279     void operator()(AEGP_LayerRenderOptionsH *layerRenderOptions)
00280     {
00281         if (layerRenderOptions && *layerRenderOptions)
00282         {
00283             SuiteManager::GetInstance()
00284                 .GetSuiteHandler()
00285                 .LayerRenderOptionsSuite2()
00286                 ->AEGP_Dispose(*layerRenderOptions);
00287         }
00288     }
00289 };
00290
00291 class MemHandleDeleter
00292 {
00293   public:
00294     void operator()(AEGP_MemHandle *memHandle)
00295     {
00296         if (memHandle && *memHandle)
00297         {
00298             SuiteManager::GetInstance()
00299                 .GetSuiteHandler()
00300                 .MemorySuite1()
00301                 ->AEGP_FreeMemHandle(*memHandle);
00302         }
00303     }
00304 };
00305
00306 class TextOutlineDeleter
```

```
00307 {
00308   public:
00309     void operator()(AEGP_TextOutlinesH *memHandle)
00310     {
00311         if (memHandle && *memHandle)
00312         {
00313             SuiteManager::GetInstance()
00314                 .GetSuiteHandler()
00315                 .TextLayerSuite1()
00316                 ->AEGP_DisposeTextOutlines(*memHandle);
00317         }
00318     }
00319 };
00320
00321 class AddKeyframesInfoDeleter
00322 {
00323   public:
00324     void operator()(AEGP_AddKeyframesInfoH *addKeyframesInfo)
00325     {
00326         if (addKeyframesInfo && *addKeyframesInfo)
00327         {
00328             SuiteManager::GetInstance()
00329                 .GetSuiteHandler()
00330                 .KeyframeSuite5()
00331                 ->AEGP_EndAddKeyframes(true, *addKeyframesInfo);
00332         }
00333     }
00334 };
00335
00336 class CollectionDeleter
00337 {
00338   public:
00339     void operator()(AEGP_Collection2H *collection)
00340     {
00341         if (collection && *collection)
00342         {
00343             SuiteManager::GetInstance()
00344                 .GetSuiteHandler()
00345                 .CollectionSuite2()
00346                 ->AEGP_DisposeCollection(*collection);
00347         }
00348     }
00349 };
00350
00351 class FrameReceiptDeleter
00352 {
00353   public:
00354     void operator()(AEGP_FrameReceiptH *frameReceipt)
00355     {
00356         if (frameReceipt && *frameReceipt)
00357         {
00358             SuiteManager::GetInstance()
00359                 .GetSuiteHandler()
00360                 .RenderSuite5()
00361                 ->AEGP_CheckinFrame(*frameReceipt);
00362         }
00363     }
00364 };
00365
00366 enum class AE_MemFlag
00367 {
00368     NONE = AEGP_MemFlag_NONE,
00369     CLEAR = AEGP_MemFlag_CLEAR,
00370     QUIET = AEGP_MemFlag_QUIET
00371 };
00372
00373 /**
00374  * @brief AE Memory Suite
00375  *
00376  * @details The Memory Suite provides access to the After Effects memory
00377  * management.
00378  *
00379  */
00380 class MemorySuite1
00381 {
00382   public:
00383     MemorySuite1() : m_suiteManager(SuiteManager::GetInstance()){};
00384     MemorySuite1(const MemorySuite1 &) = delete;
00385     MemorySuite1 &operator=(const MemorySuite1 &) = delete;
00386     MemorySuite1(MemorySuite1 &&) = delete;
00387     MemorySuite1 &operator=(MemorySuite1 &&) = delete;
00388
00389     MemHandlePtr NewMemHandle(const std::string &what, AEGP_MemSize size,
00390                               AE_MemFlag flags); /* New Mem Handle.*/
00391     void FreeMemHandle(MemHandlePtr memHandle);  /* Free Mem Handle.*/
00392     void LockMemHandle(MemHandlePtr memHandle,
00393                        void **ptrToPtr);          /* Lock Mem Handle.*/
```

```
00394     void UnlockMemHandle(MemHandlePtr memHandle); /* Unlock Mem Handle.*/
00395     AEGP_MemSize
00396     GetMemHandleSize(MemHandlePtr memHandle); /* Get Mem Handle Size.*/
00397     void ResizeMemHandle(const std::string &what, AEGP_MemSize newSize,
00398                          MemHandlePtr memHandle); /* Resize Mem Handle.*/
00399     void SetMemReportingOn(bool turnOn);          /* Set Mem Reporting On.*/
00400     std::tuple<A_long, A_long> GetMemStats();     /* Get Mem Stats.*/
00401     static inline MemHandlePtr createPtr(AEGP_MemHandle memHandle)
00402     {
00403         return std::shared_ptr<AEGP_MemHandle>(new AEGP_MemHandle(memHandle),
00404                                                MemHandleDeleter());
00405     }
00406
00407   private:
00408     SuiteManager &m_suiteManager;
00409 };
00410
00411 inline std::string ConvertUTF16ToUTF8(const A_UTF16Char *utf16String)
00412 {
00413     icu::UnicodeString unicodeString(
00414         reinterpret_cast<const UChar *>(utf16String));
00415     std::string utf8String;
00416     unicodeString.toUTF8String(utf8String);
00417     return utf8String;
00418 }
00419
00420 inline std::vector<UChar>
00421 ConvertUTF8ToUTF16UnSafe(const std::string &utf8String)
00422 {
00423     icu::UnicodeString unicodeString = icu::UnicodeString::fromUTF8(utf8String);
00424     std::vector<UChar> utf16Vector(unicodeString.length() +
00425                                    1); // +1 for null terminator
00426     UErrorCode status = U_ZERO_ERROR;
00427     unicodeString.extract(&utf16Vector[0], unicodeString.length() + 1, status);
00428
00429     // Check the status to ensure the operation was successful
00430     if (U_FAILURE(status))
00431     {
00432         // Handle the error, possibly clearing the vector or logging the failure
00433         utf16Vector.clear();
00434     }
00435     else
00436     {
00437         // Ensure null termination if needed. The extract method should already
00438         // do this, but this is just in case your logic requires it.
00439         utf16Vector[unicodeString.length()] = 0;
00440     }
00441
00442     return utf16Vector;
00443 }
00444
00445 inline std::vector<A_UTF16Char>
00446 ConvertUTF8ToUTF16(const std::string &utf8String)
00447 {
00448     auto utf16Vector = ConvertUTF8ToUTF16UnSafe(utf8String);
00449     return std::vector<A_UTF16Char>(utf16Vector.begin(), utf16Vector.end());
00450 }
00451
00452 inline std::string memHandleToString(AEGP_MemHandle memHandle)
00453 {
00454     A_Err err = A_Err_NONE;
00455     AEGP_SuiteHandler &suites = SuiteManager::GetInstance().GetSuiteHandler();
00456     A_UTF16Char *unicode_nameP = nullptr;
00457     MemorySuite1 memorySuite;
00458     MemHandlePtr ptr = memorySuite.createPtr(memHandle);
00459
00460     AE_CHECK(suites.MemorySuite1()->AEGP_LockMemHandle(
00461         memHandle, reinterpret_cast<void **>(&unicode_nameP)));
00462     std::string stringVal = ConvertUTF16ToUTF8(unicode_nameP);
00463     AE_CHECK(suites.MemorySuite1()->AEGP_UnlockMemHandle(memHandle));
00464
00465     return stringVal;
00466 }
00467
00468 /**
00469  * @brief AE Platforms.
00470  */
00471 enum class AE_Platform
00472 {
00473     WIN = AEGP_Platform_WIN,
00474     MAC = AEGP_Platform_MAC
00475 };
00476
00477 enum class AE_ProjBitDepth
00478 {
00479     _8 = AEGP_ProjBitDepth_8,
00480     _16 = AEGP_ProjBitDepth_16,
```

```
00481     _32 = AEGP_ProjBitDepth_32,
00482     NUM_VALID_DEPTHS = AEGP_ProjBitDepth_NUM_VALID_DEPTHS
00483 };
00484
00485 /**
00486  * @brief AE Color Profiles
00487  *
00488  */
00489 typedef std::tuple<double, double, double, double> ColorVal;
00490
00491 /**
00492  * @brief Convert AE ColorVal to ColorVal
00493  *
00494  * @param color
00495  * @return ColorVal
00496  */
00497 inline ColorVal toColorVal(const AEGP_ColorVal &color)
00498 {
00499     return std::make_tuple(color.alphaF, color.redF, color.greenF, color.blueF);
00500 }
00501
00502 /**
00503  * @brief Convert ColorVal to AE ColorVal
00504  *
00505  * @param color
00506  * @return AEGP_ColorVal
00507  */
00508 inline AEGP_ColorVal toAEGP_ColorVal(const ColorVal &color)
00509 {
00510     return {std::get<0>(color), std::get<1>(color), std::get<2>(color),
00511             std::get<3>(color)};
00512 }
00513
00514 enum class AE_CameraType
00515 {
00516     NONE = AEGP_CameraType_NONE,
00517     PERSPECTIVE = AEGP_CameraType_PERSPECTIVE,
00518     ORTHOGRAPHIC = AEGP_CameraType_ORTHOGRAPHIC,
00519     NUM_TYPES = AEGP_CameraType_NUM_TYPES
00520 };
00521
00522 enum class AE_TimeDisplayType
00523 {
00524     TIMECODE = AEGP_TimeDisplayType_TIMECODE,
00525     FRAMES = AEGP_TimeDisplayType_FRAMES,
00526     FEET_AND_FRAMES = AEGP_TimeDisplayType_FEET_AND_FRAMES
00527 };
00528
00529 enum class AE_FilmSizeUnits
00530 {
00531     NONE = AEGP_FilmSizeUnits_NONE,
00532     HORIZONTAL = AEGP_FilmSizeUnits_HORIZONTAL,
00533     VERTICAL = AEGP_FilmSizeUnits_VERTICAL,
00534     DIAGONAL = AEGP_FilmSizeUnits_DIAGONAL
00535 };
00536
00537 enum class AE_LightType
00538 {
00539     NONE = AEGP_LightType_NONE,
00540     PARALLEL = AEGP_LightType_PARALLEL,
00541     SPOT = AEGP_LightType_SPOT,
00542     POINT = AEGP_LightType_POINT,
00543     AMBIENT = AEGP_LightType_AMBIENT,
00544     RESERVED1 = AEGP_LightType_RESERVED1,
00545     NUM_TYPES = AEGP_LightType_NUM_TYPES
00546 };
00547
00548 enum class AE_LightFalloffType
00549 {
00550     NONE = AEGP_LightFalloff_NONE,
00551     SMOOTH = AEGP_LightFalloff_SMOOTH,
00552     INVERSE_SQUARE_CLAMPED = AEGP_LightFalloff_INVERSE_SQUARE_CLAMPED
00553 };
00554
00555 enum class AE_FootageDepth
00556 {
00557     _1 = AEGP_FootageDepth_1,
00558     _2 = AEGP_FootageDepth_2,
00559     _4 = AEGP_FootageDepth_4,
00560     _8 = AEGP_FootageDepth_8,
00561     _16 = AEGP_FootageDepth_16,
00562     _24 = AEGP_FootageDepth_24,
00563     _30 = AEGP_FootageDepth_30,
00564     _32 = AEGP_FootageDepth_32,
00565     GRAY_2 = AEGP_FootageDepth_GRAY_2,
00566     GRAY_4 = AEGP_FootageDepth_GRAY_4,
00567     GRAY_8 = AEGP_FootageDepth_GRAY_8,
```

```
00568     _48 = AEGP_FootageDepth_48,
00569     _64 = AEGP_FootageDepth_64,
00570     GRAY_16 = AEGP_FootageDepth_GRAY_16
00571 };
00572
00573 enum class AE_FramesPerFoot
00574 {
00575     _35MM = AEGP_FramesPerFoot_35MM,
00576     _16MM = AEGP_FramesPerFoot_16MM
00577 };
00578
00579 enum class AE_TimeDisplayMode
00580 {
00581     TIMECODE = AEGP_TimeDisplay_TIMECODE,
00582     FRAMES = AEGP_TimeDisplay_FRAMES
00583 };
00584
00585 enum class AE_SourceTimecodeDisplayMode
00586 {
00587     ZERO = AEGP_SourceTimecode_ZERO,
00588     SOURCE_TIMECODE = AEGP_SourceTimecode_SOURCE_TIMECODE
00589 };
00590
00591 enum class AE_FramesDisplayMode
00592 {
00593     ZERO_BASED = AEGP_Frames_ZERO_BASED,
00594     ONE_BASED = AEGP_Frames_ONE_BASED,
00595     TIMECODE_CONVERSION = AEGP_Frames_TIMECODE_CONVERSION
00596 };
00597
00598 class TimeDisplay3
00599 {
00600   public:
00601     TimeDisplay3()
00602     {
00603         m_timeDisplay.display_mode = AEGP_TimeDisplay_TIMECODE;
00604         m_timeDisplay.footage_display_mode = AEGP_SourceTimecode_ZERO;
00605         m_timeDisplay.display_dropframeB = FALSE;
00606         m_timeDisplay.use_feet_framesB = FALSE;
00607         m_timeDisplay.timebaseC = 0;
00608         m_timeDisplay.frames_per_footC = 0;
00609         m_timeDisplay.frames_display_mode = AEGP_Frames_ZERO_BASED;
00610     }
00611
00612     TimeDisplay3(AEGP_TimeDisplay3 timeDisplay) : m_timeDisplay(timeDisplay) {}
00613     TimeDisplay3(AE_TimeDisplayMode displayMode,
00614                  AE_SourceTimecodeDisplayMode footageDisplayMode,
00615                  bool displayDropFrame, bool useFeetFrames, char timeBase,
00616                  char framesPerFoot, AE_FramesDisplayMode framesDisplayMode)
00617     {
00618         m_timeDisplay.display_mode = AEGP_TimeDisplayMode(displayMode);
00619         m_timeDisplay.footage_display_mode =
00620             AEGP_SourceTimecodeDisplayMode(footageDisplayMode);
00621         m_timeDisplay.display_dropframeB = displayDropFrame;
00622         m_timeDisplay.use_feet_framesB = useFeetFrames;
00623         m_timeDisplay.timebaseC = timeBase;
00624         m_timeDisplay.frames_per_footC = framesPerFoot;
00625         m_timeDisplay.frames_display_mode =
00626             AEGP_FramesDisplayMode(framesDisplayMode);
00627     }
00628
00629     AEGP_TimeDisplay3 get() const { return m_timeDisplay; }
00630
00631     void set(AEGP_TimeDisplay3 timeDisplay) { m_timeDisplay = timeDisplay; }
00632
00633     AEGP_TimeDisplayMode getDisplayMode() const
00634     {
00635         return m_timeDisplay.display_mode;
00636     }
00637
00638     void setDisplayMode(AEGP_TimeDisplayMode displayMode)
00639     {
00640         m_timeDisplay.display_mode = displayMode;
00641     }
00642
00643     AEGP_SourceTimecodeDisplayMode getFootageDisplayMode() const
00644     {
00645         return m_timeDisplay.footage_display_mode;
00646     }
00647
00648     void
00649     setFootageDisplayMode(AEGP_SourceTimecodeDisplayMode footageDisplayMode)
00650     {
00651         m_timeDisplay.footage_display_mode = footageDisplayMode;
00652     }
00653
00654     bool getDisplayDropFrame() const
```

```
00655     {
00656         return m_timeDisplay.display_dropframeB;
00657     }
00658
00659     void setDisplayDropFrame(bool displayDropFrame)
00660     {
00661         m_timeDisplay.display_dropframeB = displayDropFrame;
00662     }
00663
00664     bool getUseFeetFrames() const { return m_timeDisplay.use_feet_framesB; }
00665
00666     void setUseFeetFrames(bool useFeetFrames)
00667     {
00668         m_timeDisplay.use_feet_framesB = useFeetFrames;
00669     }
00670
00671     char getTimeBase() const { return m_timeDisplay.timebaseC; }
00672
00673     void setTimeBase(char timeBase) { m_timeDisplay.timebaseC = timeBase; }
00674
00675     char getFramesPerFoot() const { return m_timeDisplay.frames_per_footC; }
00676
00677     void setFramesPerFoot(char framesPerFoot)
00678     {
00679         m_timeDisplay.frames_per_footC = framesPerFoot;
00680     }
00681
00682     AEGP_FramesDisplayMode getFramesDisplayMode() const
00683     {
00684         return m_timeDisplay.frames_display_mode;
00685     }
00686
00687     void setFramesDisplayMode(AEGP_FramesDisplayMode framesDisplayMode)
00688     {
00689         m_timeDisplay.frames_display_mode = framesDisplayMode;
00690     }
00691
00692   private:
00693     AEGP_TimeDisplay3 m_timeDisplay;
00694 };
00695
00696 /**
00697  * @brief AE Project Suite
00698  *
00699  * @details The Project Suite provides access to the After Effects project.
00700  *
00701  *
00702  */
00703 class ProjSuite6
00704 {
00705   public:
00706     ProjSuite6() : m_suiteManager(SuiteManager::GetInstance()){};
00707     ProjSuite6(const ProjSuite6 &) = delete;
00708     ProjSuite6 &operator=(const ProjSuite6 &) = delete;
00709     ProjSuite6(ProjSuite6 &&) = delete;
00710     ProjSuite6 &operator=(ProjSuite6 &&) = delete;
00711
00712     int GetNumProjects(); /* Get The Number of Projects in AE.*/
00713     ProjectPtr GetProjectByIndex(A_long projIndex); /* Get Project by Index.*/
00714     std::string GetProjectName(ProjectPtr project); /* Get Project Name.*/
00715     std::string GetProjectPath(ProjectPtr project); /* Get Project Path.*/
00716     ItemPtr
00717     GetProjectRootFolder(ProjectPtr project); /* Get Project Root Folder.*/
00718     void SaveProjectToPath(ProjectPtr project,
00719                            const std::string &path); /* Save Project to Path.*/
00720     TimeDisplay3
00721     GetProjectTimeDisplay(ProjectPtr project); /* Get Project Time Display.*/
00722     void SetProjectTimeDisplay(
00723         ProjectPtr project,
00724         TimeDisplay3 timeDisplay); /* Set Project Time Display.*/
00725     bool ProjectIsDirty(
00726         ProjectPtr project); /* Check if Project is Dirty (changed).*/
00727     void SaveProjectAs(ProjectPtr project,
00728                        const std::string &path); /* Save Project As.*/
00729     ProjectPtr NewProject();                     /* Create a New Project.*/
00730     ProjectPtr
00731     OpenProjectFromPath(const std::string &path); /* Open Project from Path.*/
00732     AE_ProjBitDepth
00733     GetProjectBitDepth(ProjectPtr project); /* Get Project Bit Depth.*/
00734     void
00735     SetProjectBitDepth(ProjectPtr project,
00736                        AE_ProjBitDepth bitDepth); /* Set Project Bit Depth.*/
00737
00738   private:
00739     SuiteManager &m_suiteManager;
00740 };
00741
```

```
00742 enum class AE_SoundEncoding
00743 {
00744     UNSIGNED_PCM = AEGP_SoundEncoding_UNSIGNED_PCM,
00745     SIGNED_PCM = AEGP_SoundEncoding_SIGNED_PCM,
00746     FLOAT = AEGP_SoundEncoding_FLOAT,
00747     END = AEGP_SoundEncoding_END,
00748     BEGIN = AEGP_SoundEncoding_BEGIN
00749 };
00750
00751 /**
00752  * @brief AE Sound Data Format
00753  *
00754  */
00755 class SoundDataFormat
00756 {
00757   public:
00758     SoundDataFormat()
00759     {
00760         m_soundDataFormat.sample_rateF = 0;
00761         m_soundDataFormat.encoding = AEGP_SoundEncoding_UNSIGNED_PCM;
00762         m_soundDataFormat.bytes_per_sampleL = 0;
00763         m_soundDataFormat.num_channelsL = 0;
00764     }
00765
00766     SoundDataFormat(AEGP_SoundDataFormat soundDataFormat)
00767         : m_soundDataFormat(soundDataFormat)
00768     {
00769     }
00770     SoundDataFormat(double sampleRate, AE_SoundEncoding encoding,
00771                     A_long bytesPerSample, A_long numChannels)
00772     {
00773         m_soundDataFormat.sample_rateF = sampleRate;
00774         m_soundDataFormat.encoding = AEGP_SoundEncoding(encoding);
00775         m_soundDataFormat.bytes_per_sampleL = bytesPerSample;
00776         m_soundDataFormat.num_channelsL = numChannels;
00777     }
00778
00779     /**
00780      * @brief Get the Sound Data Format object
00781      *
00782      * @return AEGP_SoundDataFormat
00783      */
00784     AEGP_SoundDataFormat get() const { return m_soundDataFormat; }
00785
00786     void set(AEGP_SoundDataFormat soundDataFormat)
00787     {
00788         m_soundDataFormat = soundDataFormat;
00789     }
00790
00791     double getSampleRate() const { return m_soundDataFormat.sample_rateF; }
00792
00793     void setSampleRate(double sampleRate)
00794     {
00795         m_soundDataFormat.sample_rateF = sampleRate;
00796     }
00797
00798     AE_SoundEncoding getEncoding() const
00799     {
00800         return AE_SoundEncoding(m_soundDataFormat.encoding);
00801     }
00802
00803     void setEncoding(AE_SoundEncoding encoding)
00804     {
00805         m_soundDataFormat.encoding = AEGP_SoundEncoding(encoding);
00806     }
00807
00808     A_long getBytesPerSample() const
00809     {
00810         return m_soundDataFormat.bytes_per_sampleL;
00811     }
00812
00813     void setBytesPerSample(A_long bytesPerSample)
00814     {
00815         m_soundDataFormat.bytes_per_sampleL = bytesPerSample;
00816     }
00817
00818     A_long getNumChannels() const { return m_soundDataFormat.num_channelsL; }
00819
00820     void setNumChannels(A_long numChannels)
00821     {
00822         m_soundDataFormat.num_channelsL = numChannels;
00823     }
00824
00825   private:
00826     AEGP_SoundDataFormat m_soundDataFormat;
00827 };
00828
```

```
00829 enum class AE_ItemType
00830 {
00831     NONE = AEGP_ItemType_NONE,
00832     FOLDER = AEGP_ItemType_FOLDER,
00833     COMP = AEGP_ItemType_COMP,
00834     SOLID = AEGP_ItemType_SOLID_defunct,
00835     FOOTAGE = AEGP_ItemType_FOOTAGE,
00836     NUM_TYPES = AEGP_ItemType_NUM_TYPES1
00837 };
00838
00839 enum class AE_ItemFlag
00840 {
00841     MISSING = AEGP_ItemFlag_MISSING,
00842     HAS_PROXY = AEGP_ItemFlag_HAS_PROXY,
00843     USING_PROXY = AEGP_ItemFlag_USING_PROXY,
00844     MISSING_PROXY = AEGP_ItemFlag_MISSING_PROXY,
00845     HAS_VIDEO = AEGP_ItemFlag_HAS_VIDEO,
00846     HAS_AUDIO = AEGP_ItemFlag_HAS_AUDIO,
00847     STILL = AEGP_ItemFlag_STILL,
00848     HAS_ACTIVE_AUDIO = AEGP_ItemFlag_HAS_ACTIVE_AUDIO
00849 };
00850
00851 enum class AE_Label
00852 {
00853     NONE = AEGP_Label_NONE,
00854     NO_LABEL = AEGP_Label_NO_LABEL,
00855     LABEL_1 = AEGP_Label_1,
00856     LABEL_2 = AEGP_Label_2,
00857     LABEL_3 = AEGP_Label_3,
00858     LABEL_4 = AEGP_Label_4,
00859     LABEL_5 = AEGP_Label_5,
00860     LABEL_6 = AEGP_Label_6,
00861     LABEL_7 = AEGP_Label_7,
00862     LABEL_8 = AEGP_Label_8,
00863     LABEL_9 = AEGP_Label_9,
00864     LABEL_10 = AEGP_Label_10,
00865     LABEL_11 = AEGP_Label_11,
00866     LABEL_12 = AEGP_Label_12,
00867     LABEL_13 = AEGP_Label_13,
00868     LABEL_14 = AEGP_Label_14,
00869     LABEL_15 = AEGP_Label_15,
00870     LABEL_16 = AEGP_Label_16,
00871     NUM_TYPES = AEGP_Label_NUMTYPES
00872 };
00873
00874 enum class AE_PersistentType
00875 {
00876     MACHINE_SPECIFIC = AEGP_PersistentType_MACHINE_SPECIFIC,
00877     MACHINE_INDEPENDENT = AEGP_PersistentType_MACHINE_INDEPENDENT,
00878     MACHINE_INDEPENDENT_RENDER = AEGP_PersistentType_MACHINE_INDEPENDENT_RENDER,
00879     MACHINE_INDEPENDENT_OUTPUT = AEGP_PersistentType_MACHINE_INDEPENDENT_OUTPUT,
00880     MACHINE_INDEPENDENT_COMPOSITION =
00881         AEGP_PersistentType_MACHINE_INDEPENDENT_COMPOSITION,
00882     MACHINE_SPECIFIC_TEXT = AEGP_PersistentType_MACHINE_SPECIFIC_TEXT,
00883     MACHINE_SPECIFIC_PAINT = AEGP_PersistentType_MACHINE_SPECIFIC_PAINT,
00884     MACHINE_SPECIFIC_EFFECTS = AEGP_PersistentType_MACHINE_SPECIFIC_EFFECTS,
00885     MACHINE_SPECIFIC_EXPRESSION_SNIPPETS =
00886         AEGP_PersistentType_MACHINE_SPECIFIC_EXPRESSION_SNIPPETS,
00887     MACHINE_SPECIFIC_SCRIPT_SNIPPETS =
00888         AEGP_PersistentType_MACHINE_SPECIFIC_SCRIPT_SNIPPETS,
00889     NUM_TYPES = AEGP_PersistentType_NUMTYPES
00890 };
00891
00892 /**
00893  * @brief AE Item Suite
00894  *
00895  * @details The Item Suite provides access to the After Effects project items.
00896  *
00897  */
00898 class ItemSuite9
00899 {
00900   public:
00901     ItemSuite9() : m_suiteManager(SuiteManager::GetInstance()){};
00902     ItemSuite9(const ItemSuite9 &) = delete;
00903     ItemSuite9 &operator=(const ItemSuite9 &) = delete;
00904     ItemSuite9(ItemSuite9 &&) = delete;
00905     ItemSuite9 &operator=(ItemSuite9 &&) = delete;
00906
00907     ItemPtr GetFirstProjItem(ProjectPtr project); /* Get First Project Item.*/
00908     ItemPtr GetNextProjItem(ProjectPtr project,
00909                             ItemPtr item); /* Get Next Project Item.*/
00910     ItemPtr GetActiveItem();               /* Get Active Item.*/
00911     bool IsItemSelected(ItemPtr item);     /* Check if Item is Selected.*/
00912     void SelectItem(ItemPtr item, bool select,
00913                     bool deselectOthers);                   /* Select Item.*/
00914     AE_ItemType GetItemType(ItemPtr item);                  /* Get Item Type.*/
00915     std::string GetTypeName(AE_ItemType itemType);          /* Get Type Name.*/
```

```
00916     std::string GetItemName(ItemPtr item);                    /* Get Item Name.*/
00917     void SetItemName(ItemPtr item, const std::string &name); /* Set Item Name.*/
00918     A_long GetItemID(ItemPtr item);                           /* Get Item ID.*/
00919     AE_ItemFlag GetItemFlags(ItemPtr item);            /* Get Item Flags.*/
00920     void SetItemUseProxy(ItemPtr item, bool useProxy); /* Set Item Use Proxy.*/
00921     ItemPtr GetItemParentFolder(ItemPtr item); /* Get Item Parent Folder.*/
00922     void SetItemParentFolder(ItemPtr item,
00923                              ItemPtr parentFolder); /* Set Item Parent Folder.*/
00924     A_Time GetItemDuration(ItemPtr item);           /* Get Item Duration.*/
00925     A_Time GetItemCurrentTime(ItemPtr item);        /* Get Item Current Time.*/
00926     std::tuple<A_long, A_long>
00927     GetItemDimensions(ItemPtr item); /* Get Item Dimensions.*/
00928     A_Ratio
00929     GetItemPixelAspectRatio(ItemPtr item); /* Get Item Pixel Aspect Ratio.*/
00930     void DeleteItem(ItemPtr item);         /* Delete Item.*/
00931     ItemPtr CreateNewFolder(const std::string &name,
00932                             ItemPtr parentFolder); /* Create New Folder.*/
00933     void SetItemCurrentTime(ItemPtr item,
00934                             A_Time newTime);  /* Set Item Current Time.*/
00935     std::string GetItemComment(ItemPtr item); /* Get Item Comment.*/
00936     void SetItemComment(ItemPtr item,
00937                         const std::string &comment); /* Set Item Comment.*/
00938     AE_Label GetItemLabel(ItemPtr item);             /* Get Item Label.*/
00939     void SetItemLabel(ItemPtr item, AE_Label label); /* Set Item Label.*/
00940     ItemViewPtr GetItemMRUView(ItemPtr item);        /* Get Item MRU View.*/
00941   private:
00942     SuiteManager &m_suiteManager;
00943 };
00944
00945 class ItemViewSuite1
00946 {
00947   public:
00948     ItemViewSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
00949
00950     A_Time GetItemViewPlaybackTime(
00951         ItemViewPtr itemView,
00952         bool &isCurrentlyPreviewing); /* Get Item View Playback Time.*/
00953
00954   private:
00955     SuiteManager &m_suiteManager;
00956 };
00957
00958 class SoundDataDeleter
00959 {
00960   public:
00961     void operator()(AEGP_SoundDataH *soundData)
00962     {
00963         if (soundData && *soundData)
00964         {
00965             SuiteManager::GetInstance()
00966                 .GetSuiteHandler()
00967                 .SoundDataSuite1()
00968                 ->AEGP_DisposeSoundData(*soundData);
00969         }
00970     }
00971 };
00972
00973 class SoundDataSuite1
00974 {
00975   public:
00976     SoundDataSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
00977     SoundDataSuite1(const SoundDataSuite1 &) = delete;
00978     SoundDataSuite1 &operator=(const SoundDataSuite1 &) = delete;
00979     SoundDataSuite1(SoundDataSuite1 &&) = delete;
00980     SoundDataSuite1 &operator=(SoundDataSuite1 &&) = delete;
00981
00982     SoundDataPtr NewSoundData(const SoundDataFormat &soundFormat);
00983
00984     SoundDataFormat GetSoundDataFormat(SoundDataPtr soundData);
00985
00986     void LockSoundDataSamples(SoundDataPtr soundData, void **samples);
00987     void UnlockSoundDataSamples(SoundDataPtr soundData);
00988     int GetNumSamples(SoundDataPtr soundData);
00989
00990   private:
00991     SuiteManager &m_suiteManager;
00992     inline SoundDataPtr toSoundDataPtr(AEGP_SoundDataH soundData)
00993     {
00994         return std::shared_ptr<AEGP_SoundDataH>(new AEGP_SoundDataH(soundData),
00995                                                 SoundDataDeleter());
00996     }
00997 };
00998
00999 /**
01000  * @brief AE Footage Suite
01001  *
01002  * @details The Footage Suite provides access to the After Effects project
```

```
01003  * footage.
01004  *
01005  */
01006 inline AEGP_DownsampleFactor
01007 toAEGP_DownsampleFactor(const std::tuple<A_short, A_short> &factor)
01008 {
01009     return {std::get<0>(factor), std::get<1>(factor)};
01010 }
01011
01012 /**
01013  * @brief Convert AEGP_DownsampleFactor to tuple
01014  *
01015  * @param factor
01016  * @return std::tuple<A_short, A_short>
01017  */
01018 inline std::tuple<short, short>
01019 toDownsampleFactor(const AEGP_DownsampleFactor &factor)
01020 {
01021     return std::make_tuple(factor.xS, factor.yS);
01022 }
01023
01024 enum class AE_CompFlag
01025 {
01026     SHOW_ALL_SHY = AEGP_CompFlag_SHOW_ALL_SHY, /* Show All Shy.*/
01027     ENABLE_MOTION_BLUR =
01028         AEGP_CompFlag_ENABLE_MOTION_BLUR, /* Enable Motion Blur.*/
01029     ENABLE_TIME_FILTER =
01030         AEGP_CompFlag_ENABLE_TIME_FILTER,          /* Enable Time Filter.*/
01031     GRID_TO_FRAMES = AEGP_CompFlag_GRID_TO_FRAMES, /* Grid to Frames.*/
01032     GRID_TO_FIELDS = AEGP_CompFlag_GRID_TO_FIELDS, /* Grid to Fields.*/
01033     USE_LOCAL_DSF = AEGP_CompFlag_USE_LOCAL_DSF,   /* Use Local DSF.*/
01034     DRAFT_3D = AEGP_CompFlag_DRAFT_3D,             /* Draft 3D.*/
01035     SHOW_GRAPH = AEGP_CompFlag_SHOW_GRAPH          /* Show Graph.*/
01036 };                                                 /* Comp Flag.*/
01037
01038 class CompSuite11
01039 {
01040   public:
01041     CompSuite11() : m_suiteManager(SuiteManager::GetInstance()){};
01042     CompSuite11(const CompSuite11 &) = delete;
01043     CompSuite11 &operator=(const CompSuite11 &) = delete;
01044     CompSuite11(CompSuite11 &&) = delete;
01045     CompSuite11 &operator=(CompSuite11 &&) = delete;
01046
01047     CompPtr GetCompFromItem(ItemPtr item); /* Get Comp from Item.*/
01048     ItemPtr GetItemFromComp(CompPtr comp); /* Get Item from Comp.*/
01049     std::tuple<short, short>
01050     GetCompDownsampleFactor(CompPtr comp); /* Get Comp Downsample Factor.*/
01051     void SetCompDownsampleFactor(CompPtr comp,
01052                                  const std::tuple<short, short>
01053                                      &factor); /* Set Comp Downsample Factor.*/
01054     ColorVal GetCompBGColor(CompPtr comp);     /* Get Comp BG Color.*/
01055     void SetCompBGColor(CompPtr comp,
01056                         const ColorVal &color); /* Set Comp BG Color.*/
01057     AE_CompFlag GetCompFlags(CompPtr comp);     /* Get Comp Flags.*/
01058     bool GetShowLayerNameOrSourceName(
01059         CompPtr comp); /* Get Show Layer Name or Source Name.*/
01060     void SetShowLayerNameOrSourceName(
01061         CompPtr comp,
01062         bool showLayerName); /* Set Show Layer Name or Source Name.*/
01063     bool GetShowBlendModes(CompPtr comp); /* Get Show Blend Modes.*/
01064     void SetShowBlendModes(CompPtr comp,
01065                            bool showBlendModes);     /* Set Show Blend Modes.*/
01066     double GetCompFramerate(CompPtr comp);           /* Get Comp Framerate.*/
01067     void SetCompFrameRate(CompPtr comp, double fps); /* Set Comp Frame Rate.*/
01068     std::tuple<A_Ratio, A_Ratio>
01069     GetCompShutterAnglePhase(CompPtr comp); /* Get Comp Shutter Angle Phase.*/
01070     std::tuple<A_Time, A_Time> GetCompShutterFrameRange(
01071         CompPtr comp, A_Time compTime); /* Get Comp Shutter Frame Range.*/
01072     int GetCompSuggestedMotionBlurSamples(
01073         CompPtr comp); /* Get Comp Suggested Motion Blur Samples.*/
01074     void SetCompSuggestedMotionBlurSamples(
01075         CompPtr comp, int samples); /* Set Comp Suggested Motion Blur Samples.*/
01076     int GetCompMotionBlurAdaptiveSampleLimit(
01077         CompPtr comp); /* Get
01078                           Comp Motion Blur Adaptive Sample Limit.*/
01079     void SetCompMotionBlurAdaptiveSampleLimit(
01080         CompPtr comp,
01081         int samples); /* Set Comp Motion Blur Adaptive Sample Limit.*/
01082     A_Time GetCompWorkAreaStart(CompPtr comp); /* Get Comp Work Area Start.*/
01083     A_Time
01084     GetCompWorkAreaDuration(CompPtr comp); /* Get Comp Work Area Duration.*/
01085     void SetCompWorkAreaStartAndDuration(
01086         CompPtr comp, A_Time workAreaStart,
01087         A_Time workAreaDuration); /* Set Comp Work Area Start and Duration.*/
01088     LayerPtr CreateSolidInComp(CompPtr comp, const std::string &name, int width,
01089                                int height, const ColorVal &color,
```

```
01090                               A_Time duration); /* Create Solid in Comp.*/
01091     LayerPtr
01092     CreateCameraInComp(CompPtr comp, const std::string &name,
01093                        A_FloatPoint centerPoint); /* Create Camera in Comp.*/
01094     LayerPtr
01095     CreateLightInComp(CompPtr comp, const std::string &name,
01096                       A_FloatPoint centerPoint); /* Create Light in Comp.*/
01097     CompPtr CreateComp(ItemPtr parentFolder, const std::string &name, int width,
01098                        int height, const A_Ratio &pixelAspectRatio,
01099                        A_Time duration,
01100                        const A_Ratio &framerate); /* Create Comp.*/
01101     Collection2Ptr GetNewCollectionFromCompSelection(
01102         AEGP_PluginID pluginId,
01103         CompPtr comp); /* Get New Collection from Comp Selection.*/
01104     A_Time
01105     GetCompDisplayStartTime(CompPtr comp); /* Get Comp Display Start Time.*/
01106     void
01107     SetCompDisplayStartTime(CompPtr comp,
01108                             A_Time startTime); /* Set Comp Display Start Time.*/
01109     void SetCompDuration(CompPtr comp, A_Time duration); /* Set Comp Duration.*/
01110     CompPtr DuplicateComp(CompPtr comp);                 /* Duplicate Comp.*/
01111     A_Time GetCompFrameDuration(CompPtr comp); /* Get Comp Frame Duration.*/
01112     CompPtr GetMostRecentlyUsedComp();         /* Get Most Recently Used Comp.*/
01113     LayerPtr
01114     CreateVectorLayerInComp(CompPtr comp); /* Create Vector Layer in Comp.*/
01115     StreamRefPtr
01116     GetNewCompMarkerStream(CompPtr parentComp); /* Get New Comp Marker Stream.*/
01117     bool
01118     GetCompDisplayDropFrame(CompPtr comp); /* Get Comp Display Drop Frame.*/
01119     void
01120     SetCompDisplayDropFrame(CompPtr comp,
01121                             bool dropFrame); /* Set Comp Display Drop Frame.*/
01122     void ReorderCompSelection(CompPtr comp,
01123                               int index); /* Reorder Comp Selection.*/
01124   private:
01125     SuiteManager &m_suiteManager;
01126 };
01127
01128 inline StreamRefPtr toStreamRefPtr(AEGP_StreamRefH streamRef)
01129 {
01130     return std::shared_ptr<AEGP_StreamRefH>(new AEGP_StreamRefH(streamRef),
01131                                             StreamRefDeleter());
01132 }
01133
01134 enum class AE_TransferFlags
01135 {
01136     PRESERVE_ALPHA = AEGP_TransferFlag_PRESERVE_ALPHA,
01137     RANDOMIZE_DISSOLVE = AEGP_TransferFlag_RANDOMIZE_DISSOLVE
01138 };
01139
01140 enum class AE_TrackMatte
01141 {
01142     NO_TRACK_MATTE = AEGP_TrackMatte_NO_TRACK_MATTE,
01143     ALPHA = AEGP_TrackMatte_ALPHA,
01144     NOT_ALPHA = AEGP_TrackMatte_NOT_ALPHA,
01145     LUMA = AEGP_TrackMatte_LUMA,
01146     NOT_LUMA = AEGP_TrackMatte_NOT_LUMA
01147 };
01148
01149 enum class AE_LayerQual
01150 {
01151     NONE = AEGP_LayerQual_NONE,
01152     WIREFRAME = AEGP_LayerQual_WIREFRAME,
01153     DRAFT = AEGP_LayerQual_DRAFT,
01154     BEST = AEGP_LayerQual_BEST
01155 };
01156
01157 enum class AE_LayerSamplingQual
01158 {
01159     BILINEAR = AEGP_LayerSamplingQual_BILINEAR,
01160     BICUBIC = AEGP_LayerSamplingQual_BICUBIC
01161 };
01162
01163 enum class AE_LayerFlag
01164 {
01165     NONE = AEGP_LayerFlag_NONE,
01166     VIDEO_ACTIVE = AEGP_LayerFlag_VIDEO_ACTIVE,
01167     AUDIO_ACTIVE = AEGP_LayerFlag_AUDIO_ACTIVE,
01168     EFFECTS_ACTIVE = AEGP_LayerFlag_EFFECTS_ACTIVE,
01169     MOTION_BLUR = AEGP_LayerFlag_MOTION_BLUR,
01170     FRAME_BLENDING = AEGP_LayerFlag_FRAME_BLENDING,
01171     LOCKED = AEGP_LayerFlag_LOCKED,
01172     SHY = AEGP_LayerFlag_SHY,
01173     COLLAPSE = AEGP_LayerFlag_COLLAPSE,
01174     AUTO_ORIENT_ROTATION = AEGP_LayerFlag_AUTO_ORIENT_ROTATION,
01175     ADJUSTMENT_LAYER = AEGP_LayerFlag_ADJUSTMENT_LAYER,
01176     TIME_REMAPPING = AEGP_LayerFlag_TIME_REMAPPING,
```

```
01177     LAYER_IS_3D = AEGP_LayerFlag_LAYER_IS_3D,
01178     LOOK_AT_CAMERA = AEGP_LayerFlag_LOOK_AT_CAMERA,
01179     LOOK_AT_POI = AEGP_LayerFlag_LOOK_AT_POI,
01180     SOLO = AEGP_LayerFlag_SOLO,
01181     MARKERS_LOCKED = AEGP_LayerFlag_MARKERS_LOCKED,
01182     NULL_LAYER = AEGP_LayerFlag_NULL_LAYER,
01183     HIDE_LOCKED_MASKS = AEGP_LayerFlag_HIDE_LOCKED_MASKS,
01184     GUIDE_LAYER = AEGP_LayerFlag_GUIDE_LAYER,
01185     ADVANCED_FRAME_BLENDING = AEGP_LayerFlag_ADVANCED_FRAME_BLENDING,
01186     SUBLAYERS_RENDER_SEPARATELY = AEGP_LayerFlag_SUBLAYERS_RENDER_SEPARATELY,
01187     ENVIRONMENT_LAYER = AEGP_LayerFlag_ENVIRONMENT_LAYER
01188 };
01189
01190 enum class AE_ObjectType
01191 {
01192     NONE = AEGP_ObjectType_NONE,
01193     AV = AEGP_ObjectType_AV,
01194     LIGHT = AEGP_ObjectType_LIGHT,
01195     CAMERA = AEGP_ObjectType_CAMERA,
01196     TEXT = AEGP_ObjectType_TEXT,
01197     VECTOR = AEGP_ObjectType_VECTOR,
01198     RESERVED1 = AEGP_ObjectType_RESERVED1,
01199     RESERVED2 = AEGP_ObjectType_RESERVED2,
01200     RESERVED3 = AEGP_ObjectType_RESERVED3,
01201     RESERVED4 = AEGP_ObjectType_RESERVED4,
01202     RESERVED5 = AEGP_ObjectType_RESERVED5,
01203     NUM_TYPES = AEGP_ObjectType_NUM_TYPES
01204 };
01205
01206 enum class AE_LTimeMode
01207 {
01208     LayerTime = AEGP_LTimeMode_LayerTime,
01209     CompTime = AEGP_LTimeMode_CompTime
01210 };
01211
01212 inline A_FloatRect
01213 toA_FloatRect(const std::tuple<double, double, double, double> &rect)
01214 {
01215     return {std::get<0>(rect), std::get<1>(rect), std::get<2>(rect),
01216             std::get<3>(rect)};
01217 }
01218
01219 inline std::tuple<double, double, double, double>
01220 toFloatRect(const A_FloatRect &rect)
01221 {
01222     return std::make_tuple(rect.left, rect.top, rect.right, rect.bottom);
01223 }
01224
01225 typedef std::tuple<double, double, double, double> FloatRect;
01226
01227 class LayerSuite9
01228 {
01229   public:
01230     LayerSuite9() : m_suiteManager(SuiteManager::GetInstance()){};
01231     LayerSuite9(const LayerSuite9 &) = delete;
01232     LayerSuite9 &operator=(const LayerSuite9 &) = delete;
01233     LayerSuite9(LayerSuite9 &&) = delete;
01234     LayerSuite9 &operator=(LayerSuite9 &&) = delete;
01235
01236     A_long GetCompNumLayers(CompPtr comp); /* Get Comp Num Layers.*/
01237     LayerPtr
01238     GetCompLayerByIndex(CompPtr comp,
01239                         A_long layerIndex);      /* Get Comp Layer By Index.*/
01240     LayerPtr GetActiveLayer();                   /* Get Active Layer.*/
01241     A_long GetLayerIndex(LayerPtr layer);        /* Get Layer Index.*/
01242     ItemPtr GetLayerSourceItem(LayerPtr layer);  /* Get Layer Source Item.*/
01243     A_long GetLayerSourceItemID(LayerPtr layer); /* Get Layer Source Item ID.*/
01244     CompPtr GetLayerParentComp(LayerPtr layer);  /* Get Layer Parent Comp.*/
01245     std::tuple<std::string, std::string>
01246     GetLayerName(LayerPtr layer);                 /* Get Layer Name.*/
01247     AE_LayerQual GetLayerQuality(LayerPtr layer); /* Get Layer Quality.*/
01248     void SetLayerQuality(LayerPtr layer,
01249                          AE_LayerQual quality); /* Set Layer Quality.*/
01250     AE_LayerFlag GetLayerFlags(LayerPtr layer); /* Get Layer Flags.*/
01251     void SetLayerFlag(LayerPtr layer, AE_LayerFlag singleFlag,
01252                       bool value);              /* Set Layer Flag.*/
01253     bool IsLayerVideoReallyOn(LayerPtr layer); /* Is Layer Video Really On.*/
01254     bool IsLayerAudioReallyOn(LayerPtr layer); /* Is Layer Audio Really On.*/
01255     A_Time
01256     GetLayerCurrentTime(LayerPtr layer,
01257                         AE_LTimeMode timeMode); /* Get Layer Current Time.*/
01258     A_Time GetLayerInPoint(LayerPtr layer,
01259                            AE_LTimeMode timeMode); /* Get Layer In Point.*/
01260     A_Time GetLayerDuration(LayerPtr layer,
01261                             AE_LTimeMode timeMode); /* Get Layer Duration.*/
01262     void SetLayerInPointAndDuration(
01263         LayerPtr layer, AE_LTimeMode timeMode, A_Time inPoint,
```

```
01264         A_Time duration);                 /* Set Layer In Point and Duration.*/
01265     A_Time GetLayerOffset(LayerPtr layer); /* Get Layer Offset.*/
01266     void SetLayerOffset(LayerPtr layer, A_Time offset); /* Set Layer Offset.*/
01267     A_Ratio GetLayerStretch(LayerPtr layer);            /* Get Layer Stretch.*/
01268     void SetLayerStretch(LayerPtr layer,
01269                          A_Ratio stretch); /* Set Layer Stretch.*/
01270     std::tuple<AE_TransferFlags, AE_TrackMatte>
01271     GetLayerTransferMode(LayerPtr layer); /* Get Layer Transfer Mode.*/
01272     void SetLayerTransferMode(
01273         LayerPtr layer, AE_TransferFlags flags,
01274         AE_TrackMatte trackMatte); /* Set Layer Transfer Mode.*/
01275     bool IsAddLayerValid(ItemPtr itemToAdd,
01276                          CompPtr intoComp); /* Is Add Layer Valid.*/
01277     LayerPtr AddLayer(ItemPtr itemToAdd, CompPtr intoComp); /* Add Layer.*/
01278     void ReorderLayer(LayerPtr layer, A_long layerIndex);   /* Reorder Layer.*/
01279     FloatRect GetLayerMaskedBounds(LayerPtr layer, AE_LTimeMode timeMode,
01280                                    A_Time time); /* Get Layer Masked Bounds.*/
01281     AE_ObjectType
01282     GetLayerObjectType(LayerPtr layer); /* Get Layer Object Type.*/
01283     bool IsLayer3D(LayerPtr layer);     /* Is Layer 3D.*/
01284     bool IsLayer2D(LayerPtr layer);     /* Is Layer 2D.*/
01285     bool IsVideoActive(LayerPtr layer, AE_LTimeMode timeMode,
01286                        A_Time time); /* Is Video Active.*/
01287     bool IsLayerUsedAsTrackMatte(
01288         LayerPtr layer,
01289         bool fillMustBeActive); /* Is Layer Used As Track Matte.*/
01290     bool
01291     DoesLayerHaveTrackMatte(LayerPtr layer); /* Does Layer Have Track Matte.*/
01292     A_Time
01293     ConvertCompToLayerTime(LayerPtr layer,
01294                            A_Time compTime); /* Convert Comp To Layer Time.*/
01295     A_Time
01296     ConvertLayerToCompTime(LayerPtr layer,
01297                            A_Time layerTime); /* Convert Layer To Comp Time.*/
01298     A_long GetLayerDancingRandValue(
01299         LayerPtr layer, A_Time compTime); /* Get Layer Dancing Rand Value.*/
01300     AEGP_LayerIDVal GetLayerID(LayerPtr layer); /* Get Layer ID.*/
01301     A_Matrix4
01302     GetLayerToWorldXform(LayerPtr layer,
01303                          A_Time compTime); /* Get Layer To World Xform.*/
01304     A_Matrix4 GetLayerToWorldXformFromView(
01305         LayerPtr layer, A_Time viewTime,
01306         A_Time compTime); /* Get Layer To World Xform From View.*/
01307     void SetLayerName(LayerPtr layer,
01308                       const std::string &newName); /* Set Layer Name.*/
01309     LayerPtr GetLayerParent(LayerPtr layer);       /* Get Layer Parent.*/
01310     void SetLayerParent(LayerPtr layer,
01311                         LayerPtr parentLayer);   /* Set Layer Parent.*/
01312     void DeleteLayer(LayerPtr layer);            /* Delete Layer.*/
01313     LayerPtr DuplicateLayer(LayerPtr origLayer); /* Duplicate Layer.*/
01314     LayerPtr
01315     GetLayerFromLayerID(CompPtr parentComp,
01316                         AEGP_LayerIDVal id);    /* Get Layer From Layer ID.*/
01317     AEGP_LabelID GetLayerLabel(LayerPtr layer); /* Get Layer Label.*/
01318     void SetLayerLabel(LayerPtr layer,
01319                        AEGP_LabelID label); /* Set Layer Label.*/
01320     AE_LayerSamplingQual
01321     GetLayerSamplingQuality(LayerPtr layer); /* Get Layer Sampling Quality.*/
01322     void SetLayerSamplingQuality(
01323         LayerPtr layer,
01324         AE_LayerSamplingQual quality); /* Set Layer Sampling Quality.*/
01325     LayerPtr GetTrackMatteLayer(LayerPtr layer); /* Get Track Matte Layer.*/
01326     void SetTrackMatte(LayerPtr layer, LayerPtr trackMatteLayer,
01327                        AE_TrackMatte trackMatteType); /* Set Track Matte.*/
01328     void RemoveTrackMatte(LayerPtr layer);            /* Remove Track Matte.*/
01329
01330   private:
01331     SuiteManager &m_suiteManager;
01332 };
01333
01334 enum class AE_LayerStream
01335 {
01336     // Valid for all layer types
01337     ANCHORPOINT = AEGP_LayerStream_ANCHORPOINT,
01338     POSITION = AEGP_LayerStream_POSITION,
01339     SCALE = AEGP_LayerStream_SCALE,
01340     ROTATION = AEGP_LayerStream_ROTATION,
01341     ROTATE_Z = AEGP_LayerStream_ROTATE_Z,
01342     OPACITY = AEGP_LayerStream_OPACITY,
01343     AUDIO = AEGP_LayerStream_AUDIO,
01344     MARKER = AEGP_LayerStream_MARKER,
01345     TIME_REMAP = AEGP_LayerStream_TIME_REMAP,
01346     ROTATE_X = AEGP_LayerStream_ROTATE_X,
01347     ROTATE_Y = AEGP_LayerStream_ROTATE_Y,
01348     ORIENTATION = AEGP_LayerStream_ORIENTATION,
01349
01350     // only valid for AEGP_ObjectType == AEGP_ObjectType_CAMERA
```

```
01351     ZOOM = AEGP_LayerStream_ZOOM,
01352     DEPTH_OF_FIELD = AEGP_LayerStream_DEPTH_OF_FIELD,
01353     FOCUS_DISTANCE = AEGP_LayerStream_FOCUS_DISTANCE,
01354     APERTURE = AEGP_LayerStream_APERTURE,
01355     BLUR_LEVEL = AEGP_LayerStream_BLUR_LEVEL,
01356     IRIS_SHAPE = AEGP_LayerStream_IRIS_SHAPE,
01357     IRIS_ROTATION = AEGP_LayerStream_IRIS_ROTATION,
01358     IRIS_ROUNDNESS = AEGP_LayerStream_IRIS_ROUNDNESS,
01359     IRIS_ASPECT_RATIO = AEGP_LayerStream_IRIS_ASPECT_RATIO,
01360     IRIS_DIFFRACTION_FRINGE = AEGP_LayerStream_IRIS_DIFFRACTION_FRINGE,
01361     IRIS_HIGHLIGHT_GAIN = AEGP_LayerStream_IRIS_HIGHLIGHT_GAIN,
01362     IRIS_HIGHLIGHT_THRESHOLD = AEGP_LayerStream_IRIS_HIGHLIGHT_THRESHOLD,
01363     IRIS_HIGHLIGHT_SATURATION = AEGP_LayerStream_IRIS_HIGHLIGHT_SATURATION,
01364
01365     // only valid for AEGP_ObjectType == AEGP_ObjectType_LIGHT
01366     INTENSITY = AEGP_LayerStream_INTENSITY,
01367     COLOR = AEGP_LayerStream_COLOR,
01368     CONE_ANGLE = AEGP_LayerStream_CONE_ANGLE,
01369     CONE_FEATHER = AEGP_LayerStream_CONE_FEATHER,
01370     SHADOW_DARKNESS = AEGP_LayerStream_SHADOW_DARKNESS,
01371     SHADOW_DIFFUSION = AEGP_LayerStream_SHADOW_DIFFUSION,
01372     LIGHT_FALLOFF_TYPE = AEGP_LayerStream_LIGHT_FALLOFF_TYPE,
01373     LIGHT_FALLOFF_START = AEGP_LayerStream_LIGHT_FALLOFF_START,
01374     LIGHT_FALLOFF_DISTANCE = AEGP_LayerStream_LIGHT_FALLOFF_DISTANCE,
01375
01376     // only valid for AEGP_ObjectType == AEGP_ObjectType_AV
01377     ACCEPTS_SHADOWS = AEGP_LayerStream_ACCEPTS_SHADOWS,
01378     ACCEPTS_LIGHTS = AEGP_LayerStream_ACCEPTS_LIGHTS,
01379     AMBIENT_COEFF = AEGP_LayerStream_AMBIENT_COEFF,
01380     DIFFUSE_COEFF = AEGP_LayerStream_DIFFUSE_COEFF,
01381     SPECULAR_INTENSITY = AEGP_LayerStream_SPECULAR_INTENSITY,
01382     SPECULAR_SHININESS = AEGP_LayerStream_SPECULAR_SHININESS,
01383     CASTS_SHADOWS = AEGP_LayerStream_CASTS_SHADOWS,
01384     LIGHT_TRANSMISSION = AEGP_LayerStream_LIGHT_TRANSMISSION,
01385     METAL = AEGP_LayerStream_METAL,
01386     REFLECTION_INTENSITY = AEGP_LayerStream_REFLECTION_INTENSITY,
01387     REFLECTION_SHARPNESS = AEGP_LayerStream_REFLECTION_SHARPNESS,
01388     REFLECTION_ROLLOFF = AEGP_LayerStream_REFLECTION_ROLLOFF,
01389     TRANSPARENCY_COEFF = AEGP_LayerStream_TRANSPARENCY_COEFF,
01390     TRANSPARENCY_ROLLOFF = AEGP_LayerStream_TRANSPARENCY_ROLLOFF,
01391     INDEX_OF_REFRACTION = AEGP_LayerStream_INDEX_OF_REFRACTION,
01392     EXTRUSION_BEVEL_STYLE = AEGP_LayerStream_EXTRUSION_BEVEL_STYLE,
01393     EXTRUSION_BEVEL_DIRECTION = AEGP_LayerStream_EXTRUSION_BEVEL_DIRECTION,
01394     EXTRUSION_BEVEL_DEPTH = AEGP_LayerStream_EXTRUSION_BEVEL_DEPTH,
01395     EXTRUSION_HOLE_BEVEL_DEPTH = AEGP_LayerStream_EXTRUSION_HOLE_BEVEL_DEPTH,
01396     EXTRUSION_DEPTH = AEGP_LayerStream_EXTRUSION_DEPTH,
01397     PLANE_CURVATURE = AEGP_LayerStream_PLANE_CURVATURE,
01398     PLANE_SUBDIVISION = AEGP_LayerStream_PLANE_SUBDIVISION
01399 };
01400
01401 enum class AE_MaskStream
01402 {
01403     OUTLINE = AEGP_MaskStream_OUTLINE,
01404     OPACITY = AEGP_MaskStream_OPACITY,
01405     FEATHER = AEGP_MaskStream_FEATHER,
01406     EXPANSION = AEGP_MaskStream_EXPANSION
01407 };
01408
01409 enum class AE_StreamFlag
01410 {
01411     NONE = AEGP_StreamFlag_NONE,
01412     HAS_MIN = AEGP_StreamFlag_HAS_MIN,
01413     HAS_MAX = AEGP_StreamFlag_HAS_MAX,
01414     IS_SPATIAL = AEGP_StreamFlag_IS_SPATIAL
01415 };
01416
01417 enum class AE_KeyInterp
01418 {
01419     NONE = AEGP_KeyInterp_NONE,
01420     LINEAR = AEGP_KeyInterp_LINEAR,
01421     BEZIER = AEGP_KeyInterp_BEZIER,
01422     HOLD = AEGP_KeyInterp_HOLD
01423 };
01424
01425 enum class AE_KeyInterpMask
01426 {
01427     NONE = AEGP_KeyInterpMask_NONE,
01428     LINEAR = AEGP_KeyInterpMask_LINEAR,
01429     BEZIER = AEGP_KeyInterpMask_BEZIER,
01430     HOLD = AEGP_KeyInterpMask_HOLD,
01431     CUSTOM = AEGP_KeyInterpMask_CUSTOM,
01432     ANY = AEGP_KeyInterpMask_ANY
01433 };
01434
01435 inline AEGP_TwoDVal toAEGP_TwoDVal(const std::tuple<double, double> &val)
01436 {
01437     return {std::get<0>(val), std::get<1>(val)};
```

```
01438 }
01439
01440 inline std::tuple<double, double> toTwoDVal(const AEGP_TwoDVal &val)
01441 {
01442     return std::make_tuple(val.x, val.y);
01443 }
01444
01445 inline AEGP_ThreeDVal
01446 toAEGP_ThreeDVal(const std::tuple<double, double, double> &val)
01447 {
01448     return {std::get<0>(val), std::get<1>(val), std::get<2>(val)};
01449 }
01450
01451 inline std::tuple<double, double, double> toThreeDVal(const AEGP_ThreeDVal &val)
01452 {
01453     return std::make_tuple(val.x, val.y, val.z);
01454 }
01455
01456 enum class AE_StreamType
01457 {
01458     NONE = AEGP_StreamType_NO_DATA,
01459     ThreeD_SPATIAL = AEGP_StreamType_ThreeD_SPATIAL,
01460     ThreeD = AEGP_StreamType_ThreeD,
01461     TwoD_SPATIAL = AEGP_StreamType_TwoD_SPATIAL,
01462     TwoD = AEGP_StreamType_TwoD,
01463     OneD = AEGP_StreamType_OneD,
01464     COLOR = AEGP_StreamType_COLOR,
01465     ARB = AEGP_StreamType_ARB,
01466     MARKER = AEGP_StreamType_MARKER,
01467     LAYER_ID = AEGP_StreamType_LAYER_ID,
01468     MASK_ID = AEGP_StreamType_MASK_ID,
01469     MASK = AEGP_StreamType_MASK,
01470     TEXT_DOCUMENT = AEGP_StreamType_TEXT_DOCUMENT
01471 };
01472
01473 typedef std::tuple<double, double> AE_KeyframeEase;
01474
01475 inline AEGP_KeyframeEase
01476 toAEGP_KeyframeEase(const std::tuple<double, double> &val)
01477 {
01478     return {std::get<0>(val), std::get<1>(val)};
01479 }
01480
01481 inline std::tuple<double, double> toKeyframeEase(const AEGP_KeyframeEase &val)
01482 {
01483     return std::make_tuple(val.speedF, val.influenceF);
01484 }
01485
01486 class StreamSuite6
01487 {
01488   public:
01489     StreamSuite6() : m_suiteManager(SuiteManager::GetInstance()){};
01490     StreamSuite6(const StreamSuite6 &) = delete;
01491
01492     StreamSuite6 &operator=(const StreamSuite6 &) = delete;
01493
01494     StreamSuite6(StreamSuite6 &&) = delete;
01495
01496     StreamSuite6 &operator=(StreamSuite6 &&) = delete;
01497
01498     bool IsStreamLegal(LayerPtr layer,
01499                        AE_LayerStream whichStream); /* Is Stream Legal.*/
01500     bool CanVaryOverTime(StreamRefPtr stream);      /* Can Vary Over Time.*/
01501     AE_KeyInterpMask
01502     GetValidInterpolations(StreamRefPtr stream); /* Get Valid Interpolations.*/
01503     StreamRefPtr
01504     GetNewLayerStream(LayerPtr layer,
01505                       AE_LayerStream whichStream); /* Get New Layer Stream.*/
01506     A_long GetEffectNumParamStreams(
01507         EffectRefPtr effectRef); /* Get Effect Num Param Streams.*/
01508     StreamRefPtr GetNewEffectStreamByIndex(
01509         EffectRefPtr effectRef,
01510         A_long paramIndex); /* Get New Effect Stream By Index.*/
01511     StreamRefPtr
01512     GetNewMaskStream(MaskRefPtr maskRef,
01513                      AE_MaskStream whichStream); /* Get New Mask Stream.*/
01514     std::string GetStreamName(StreamRefPtr stream,
01515                               bool forceEnglish); /* Get Stream Name.*/
01516     std::string
01517     GetStreamUnitsText(StreamRefPtr stream,
01518                        bool forceEnglish); /* Get Stream Units Text.*/
01519     std::tuple<AE_StreamFlag, double, double>
01520     GetStreamProperties(StreamRefPtr stream);      /* Get Stream Properties.*/
01521     bool IsStreamTimevarying(StreamRefPtr stream); /* Is Stream Timevarying.*/
01522     AE_StreamType GetStreamType(StreamRefPtr stream); /* Get Stream Type.*/
01523     AEGP_StreamValue2
01524     GetNewStreamValue(StreamRefPtr stream, AE_LTimeMode timeMode, A_Time time,
```

```
01525                       bool preExpression);            /* Get New Stream Value.*/
01526     void DisposeStreamValue(AEGP_StreamValue2 value); /* Dispose Stream Value.*/
01527     void SetStreamValue(StreamRefPtr stream,
01528                         AEGP_StreamValue2 value); /* Set Stream Value.*/
01529     std::tuple<AEGP_StreamVal2, AE_StreamType>
01530     GetLayerStreamValue(LayerPtr layer, AE_LayerStream whichStream,
01531                         AE_LTimeMode timeMode, A_Time time,
01532                         bool preExpression); /* Get Layer Stream Value.*/
01533
01534     StreamRefPtr
01535     DuplicateStreamRef(StreamRefPtr stream);    /* Duplicate Stream Ref.*/
01536     int GetUniqueStreamID(StreamRefPtr stream); /* Get Unique Stream ID.*/
01537
01538     static inline StreamRefPtr createPtr(AEGP_StreamRefH streamRef)
01539     {
01540         return std::shared_ptr<AEGP_StreamRefH>(new AEGP_StreamRefH(streamRef),
01541                                                 StreamRefDeleter());
01542     }
01543
01544   private:
01545     SuiteManager &m_suiteManager;
01546 };
01547
01548 enum class AE_StreamGroupingType
01549 {
01550     NONE = AEGP_StreamGroupingType_NONE,
01551     LEAF = AEGP_StreamGroupingType_LEAF,
01552     NAMED_GROUP = AEGP_StreamGroupingType_NAMED_GROUP,
01553     INDEXED_GROUP = AEGP_StreamGroupingType_INDEXED_GROUP
01554 };
01555
01556 enum class AE_DynStreamFlag
01557 {
01558     ACTIVE_EYEBALL = AEGP_DynStreamFlag_ACTIVE_EYEBALL,
01559     HIDDEN = AEGP_DynStreamFlag_HIDDEN,
01560     DISABLED = AEGP_DynStreamFlag_DISABLED,
01561     ELIDED = AEGP_DynStreamFlag_ELIDED,
01562     SHOWN_WHEN_EMPTY = AEGP_DynStreamFlag_SHOWN_WHEN_EMPTY,
01563     SKIP_REVEAL_WHEN_UNHIDDEN = AEGP_DynStreamFlag_SKIP_REVEAL_WHEN_UNHIDDEN
01564 };
01565
01566 // TODO: Add support for AEGP_StreamSuite4
01567 class DynamicStreamSuite4
01568 {
01569   public:
01570     DynamicStreamSuite4() : m_suiteManager(SuiteManager::GetInstance()){};
01571     DynamicStreamSuite4(const DynamicStreamSuite4 &) = delete;
01572     DynamicStreamSuite4 &operator=(const DynamicStreamSuite4 &) = delete;
01573     DynamicStreamSuite4(DynamicStreamSuite4 &&) = delete;
01574     DynamicStreamSuite4 &operator=(DynamicStreamSuite4 &&) = delete;
01575
01576     StreamRefPtr
01577     GetNewStreamRefForLayer(LayerPtr layer); /* Get New Stream Ref For Layer.*/
01578     StreamRefPtr
01579     GetNewStreamRefForMask(MaskRefPtr mask); /* Get New Stream Ref For Mask.*/
01580     A_long GetStreamDepth(StreamRefPtr stream); /* Get Stream Depth.*/
01581     AE_StreamGroupingType
01582     GetStreamGroupingType(StreamRefPtr stream); /* Get Stream Grouping Type.*/
01583     A_long
01584     GetNumStreamsInGroup(StreamRefPtr stream); /* Get Num Streams In Group.*/
01585     AE_DynStreamFlag
01586     GetDynamicStreamFlags(StreamRefPtr stream); /* Get Dynamic Stream Flags.*/
01587     void SetDynamicStreamFlag(StreamRefPtr stream, AE_DynStreamFlag oneFlag,
01588                               bool undoable,
01589                               bool set); /* Set Dynamic Stream Flag.*/
01590     StreamRefPtr
01591     GetNewStreamRefByIndex(StreamRefPtr parentGroup,
01592                            A_long index); /* Get New Stream Ref By Index.*/
01593     StreamRefPtr GetNewStreamRefByMatchname(
01594         StreamRefPtr parentGroup,
01595         const std::string &matchName); /* Get New Stream Ref By Matchname.*/
01596     void DeleteStream(StreamRefPtr stream); /* Delete Stream.*/
01597     void ReorderStream(StreamRefPtr stream,
01598                        A_long newIndex);         /* Reorder Stream.*/
01599     A_long DuplicateStream(StreamRefPtr stream); /* Duplicate Stream.*/
01600     void SetStreamName(StreamRefPtr stream,
01601                        const std::string &newName); /* Set Stream Name.*/
01602     bool CanAddStream(StreamRefPtr parentGroup,
01603                       const std::string &matchName); /* Can Add Stream.*/
01604     StreamRefPtr AddStream(StreamRefPtr parentGroup,
01605                            const std::string &matchName); /* Add Stream.*/
01606     std::string GetMatchname(StreamRefPtr stream);        /* Get Matchname.*/
01607     StreamRefPtr
01608     GetNewParentStreamRef(StreamRefPtr stream); /* Get New Parent Stream Ref.*/
01609     bool GetStreamIsModified(StreamRefPtr stream); /* Get Stream Is Modified.*/
01610     bool IsSeparationLeader(StreamRefPtr stream);  /* Is Separation Leader.*/
01611     bool AreDimensionsSeparated(
```

```
01612         StreamRefPtr leaderStream); /* Are Dimensions Separated.*/
01613     void SetDimensionsSeparated(StreamRefPtr leaderStream,
01614                                 bool separated); /* Set Dimensions Separated.*/
01615     StreamRefPtr GetSeparationFollower(
01616         A_long dimension,
01617         StreamRefPtr leaderStream); /* Get Separation Follower.*/
01618     bool IsSeparationFollower(StreamRefPtr stream); /* Is Separation Follower.*/
01619     StreamRefPtr GetSeparationLeader(
01620         StreamRefPtr followerStream); /* Get Separation Leader.*/
01621     A_short
01622     GetSeparationDimension(StreamRefPtr stream); /* Get Separation Dimension.*/
01623   private:
01624     SuiteManager &m_suiteManager;
01625 };
01626
01627 using StreamVal =
01628     std::variant<AEGP_OneDVal, AEGP_TwoDVal, AEGP_ThreeDVal, AEGP_ColorVal,
01629                  MarkerValPtr, A_long, MaskOutlineValPtr, TextDocumentPtr>;
01630
01631 inline StreamVal CreateStream(AEGP_StreamValue2 val)
01632 {
01633     AE_StreamType streamType;
01634     streamType = StreamSuite6().GetStreamType(toStreamRefPtr(val.streamH));
01635     switch (streamType)
01636     {
01637     case AE_StreamType::OneD:
01638         return val.val.one_d;
01639     case AE_StreamType::TwoD:
01640     case AE_StreamType::TwoD_SPATIAL:
01641         return val.val.two_d;
01642     case AE_StreamType::ThreeD:
01643     case AE_StreamType::ThreeD_SPATIAL:
01644         return val.val.three_d;
01645     case AE_StreamType::COLOR:
01646         return val.val.color;
01647     case AE_StreamType::MARKER:
01648         return std::shared_ptr<AEGP_MarkerValP>(
01649             new AEGP_MarkerValP(val.val.markerP), MarkerDeleter());
01650     case AE_StreamType::LAYER_ID:
01651         return val.val.layer_id;
01652     case AE_StreamType::MASK_ID:
01653         return val.val.mask_id;
01654     case AE_StreamType::MASK:
01655         return std::make_shared<AEGP_MaskOutlineValH>(val.val.mask);
01656     case AE_StreamType::TEXT_DOCUMENT:
01657         return std::make_shared<AEGP_TextDocumentH>(val.val.text_documentH);
01658     }
01659     StreamSuite6().DisposeStreamValue(val);
01660 };
01661
01662 enum class AE_KeyframeFlag
01663 {
01664     NONE = AEGP_KeyframeFlag_NONE,
01665     TEMPORAL_CONTINUOUS = AEGP_KeyframeFlag_TEMPORAL_CONTINUOUS,
01666     TEMPORAL_AUTOBEZIER = AEGP_KeyframeFlag_TEMPORAL_AUTOBEZIER,
01667     SPATIAL_CONTINUOUS = AEGP_KeyframeFlag_SPATIAL_CONTINUOUS,
01668     SPATIAL_AUTOBEZIER = AEGP_KeyframeFlag_SPATIAL_AUTOBEZIER,
01669     ROVING = AEGP_KeyframeFlag_ROVING
01670 };
01671
01672 class KeyframeSuite5
01673 {
01674   public:
01675     KeyframeSuite5() : m_suiteManager(SuiteManager::GetInstance()){};
01676     KeyframeSuite5(const KeyframeSuite5 &) = delete;
01677     KeyframeSuite5 &operator=(const KeyframeSuite5 &) = delete;
01678     KeyframeSuite5(KeyframeSuite5 &&) = delete;
01679     KeyframeSuite5 &operator=(KeyframeSuite5 &&) = delete;
01680
01681     A_long GetStreamNumKFs(StreamRefPtr stream); /* Get Stream Num KFs.*/
01682     A_Time GetKeyframeTime(StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01683                            AE_LTimeMode timeMode); /* Get Keyframe Time.*/
01684     AEGP_KeyframeIndex InsertKeyframe(StreamRefPtr stream,
01685                                       AE_LTimeMode timeMode,
01686                                       const A_Time &time); /* Insert Keyframe.*/
01687     void DeleteKeyframe(StreamRefPtr stream,
01688                         AEGP_KeyframeIndex keyIndex); /* Delete Keyframe.*/
01689     AEGP_StreamValue2 GetNewKeyframeValue(
01690         StreamRefPtr stream,
01691         AEGP_KeyframeIndex keyIndex); /* Get New Keyframe Value.*/
01692     void SetKeyframeValue(StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01693                           AEGP_StreamValue2 value); /* Set Keyframe Value.*/
01694     A_short GetStreamValueDimensionality(
01695         StreamRefPtr stream); /* Get Stream Value Dimensionality.*/
01696     A_short GetStreamTemporalDimensionality(
01697         StreamRefPtr stream); /* Get Stream Temporal Dimensionality.*/
01698     std::tuple<AEGP_StreamValue2, AEGP_StreamValue2>
```

```
01699     GetNewKeyframeSpatialTangents(
01700         StreamRefPtr stream,
01701         AEGP_KeyframeIndex keyIndex); /* Get New Keyframe Spatial Tangents.*/
01702     void SetKeyframeSpatialTangents(
01703         StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01704         AEGP_StreamValue2 inTan,
01705         AEGP_StreamValue2 outTan); /* Set Keyframe Spatial Tangents.*/
01706     std::tuple<AE_KeyframeEase, AE_KeyframeEase>
01707     GetKeyframeTemporalEase(StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01708                             A_long dimension); /* Get Keyframe Temporal Ease.*/
01709     void SetKeyframeTemporalEase(
01710         StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, A_long dimension,
01711         AE_KeyframeEase inEase,
01712         AE_KeyframeEase outEase); /* Set Keyframe Temporal Ease.*/
01713     AE_KeyframeFlag
01714     GetKeyframeFlags(StreamRefPtr stream,
01715                      AEGP_KeyframeIndex keyIndex); /* Get Keyframe Flags.*/
01716     void SetKeyframeFlag(StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01717                          AE_KeyframeFlag flag,
01718                          bool value); /* Set Keyframe Flag.*/
01719     std::tuple<AE_KeyInterp, AE_KeyInterp> GetKeyframeInterpolation(
01720         StreamRefPtr stream,
01721         AEGP_KeyframeIndex keyIndex); /* Get Keyframe Interpolation.*/
01722     void SetKeyframeInterpolation(
01723         StreamRefPtr stream, AEGP_KeyframeIndex keyIndex, AE_KeyInterp inInterp,
01724         AE_KeyInterp outInterp); /* Set Keyframe Interpolation.*/
01725     AddKeyframesInfoPtr
01726     StartAddKeyframes(StreamRefPtr stream); /* Start Add Keyframes.*/
01727     AEGP_KeyframeIndex AddKeyframes(AddKeyframesInfoPtr akH,
01728                                     AE_LTimeMode timeMode,
01729                                     const A_Time &time); /* Add Keyframes.*/
01730     void SetAddKeyframe(AddKeyframesInfoPtr akH, AEGP_KeyframeIndex keyIndex,
01731                         AEGP_StreamValue2 value); /* Set Add Keyframe.*/
01732     A_long GetKeyframeLabelColorIndex(
01733         StreamRefPtr stream,
01734         AEGP_KeyframeIndex keyIndex); /* Get Keyframe Label Color Index.*/
01735     void SetKeyframeLabelColorIndex(
01736         StreamRefPtr stream, AEGP_KeyframeIndex keyIndex,
01737         A_long keyLabel); /* Set Keyframe Label Color Index.*/
01738
01739   private:
01740     SuiteManager &m_suiteManager;
01741     static inline AddKeyframesInfoPtr createPtr(AEGP_AddKeyframesInfoH ref)
01742     {
01743         return std::shared_ptr<AEGP_AddKeyframesInfoH>(
01744             new AEGP_AddKeyframesInfoH(ref), AddKeyframesInfoDeleter());
01745     }
01746 };
01747
01748 class TextDocumentSuite1
01749 {
01750   public:
01751     TextDocumentSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
01752     TextDocumentSuite1(const TextDocumentSuite1 &) = delete;
01753     TextDocumentSuite1 &operator=(const TextDocumentSuite1 &) = delete;
01754     TextDocumentSuite1(TextDocumentSuite1 &&) = delete;
01755     TextDocumentSuite1 &operator=(TextDocumentSuite1 &&) = delete;
01756
01757     std::string getNewText(TextDocumentPtr text_documentH);
01758     void setText(TextDocumentPtr text_documentH, const std::string &unicodePS);
01759
01760   private:
01761     SuiteManager &m_suiteManager;
01762 };
01763
01764 class MarkerSuite3
01765 {
01766   public:
01767     MarkerSuite3() : m_suiteManager(SuiteManager::GetInstance()){};
01768     MarkerSuite3(const MarkerSuite3 &) = delete;
01769     MarkerSuite3 &operator=(const MarkerSuite3 &) = delete;
01770     MarkerSuite3(MarkerSuite3 &&) = delete;
01771     MarkerSuite3 &operator=(MarkerSuite3 &&) = delete;
01772
01773     MarkerValPtr getNewMarker();
01774     void disposeMarker(MarkerValPtr markerP);
01775     MarkerValPtr duplicateMarker(MarkerValPtr markerP);
01776     void setMarkerFlag(MarkerValPtr markerP, AEGP_MarkerFlagType flagType,
01777                        bool valueB);
01778     bool getMarkerFlag(MarkerValPtr markerP, AEGP_MarkerFlagType flagType);
01779     std::string getMarkerString(MarkerValPtr markerP,
01780                                 AEGP_MarkerStringType strType);
01781     void setMarkerString(MarkerValPtr markerP, AEGP_MarkerStringType strType,
01782                          const std::string &unicodeP, A_long lengthL);
01783     A_long countCuePointParams(MarkerValPtr markerP);
01784     std::tuple<std::string, std::string> getIndCuePointParam(
01785
```

```
01786         MarkerValPtr markerP, A_long param_indexL);
01787     void setIndCuePointParam(MarkerValPtr markerP, A_long param_indexL,
01788                              const std::string &unicodeKeyP, A_long key_lengthL,
01789                              const std::string &unicodeValueP,
01790                              A_long value_lengthL);
01791     void insertCuePointParam(MarkerValPtr markerP, A_long param_indexL);
01792     void deleteIndCuePointParam(MarkerValPtr markerP, A_long param_indexL);
01793     void setMarkerDuration(MarkerValPtr markerP, const A_Time &durationPT);
01794     A_Time getMarkerDuration(MarkerValPtr markerP);
01795     void setMarkerLabel(MarkerValPtr markerP, A_long value);
01796     A_long getMarkerLabel(MarkerValPtr markerP);
01797
01798   private:
01799     SuiteManager &m_suiteManager;
01800     static inline MarkerValPtr createPtr(AEGP_MarkerValP ref)
01801     {
01802         return std::shared_ptr<AEGP_MarkerValP>(new AEGP_MarkerValP(ref),
01803                                                 MarkerDeleter());
01804     }
01805 };
01806
01807 class TextLayerSuite1
01808 {
01809   public:
01810     TextLayerSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
01811     TextLayerSuite1(const TextLayerSuite1 &) = delete;
01812     TextLayerSuite1 &operator=(const TextLayerSuite1 &) = delete;
01813     TextLayerSuite1(TextLayerSuite1 &&) = delete;
01814     TextLayerSuite1 &operator=(TextLayerSuite1 &&) = delete;
01815
01816     TextOutlinesPtr getNewTextOutlines(LayerPtr layer,
01817                                        const A_Time &layer_time);
01818     int getNumTextOutlines(TextOutlinesPtr outlines);
01819     PF_PathOutlinePtr getIndexedTextOutline(TextOutlinesPtr outlines,
01820                                             int path_index);
01821
01822   private:
01823     SuiteManager &m_suiteManager;
01824     static inline TextOutlinesPtr createPtr(AEGP_TextOutlinesH ref)
01825     {
01826         return std::shared_ptr<AEGP_TextOutlinesH>(new AEGP_TextOutlinesH(ref),
01827                                                    TextOutlineDeleter());
01828     }
01829 };
01830
01831 enum class AE_EffectFlags
01832 {
01833     NONE = AEGP_EffectFlags_NONE,
01834     ACTIVE = AEGP_EffectFlags_ACTIVE,
01835     AUDIO_ONLY = AEGP_EffectFlags_AUDIO_ONLY,
01836     AUDIO_TOO = AEGP_EffectFlags_AUDIO_TOO,
01837     MISSING = AEGP_EffectFlags_MISSING
01838 };
01839
01840 class EffectSuite4
01841 {
01842   public:
01843     EffectSuite4() : m_suiteManager(SuiteManager::GetInstance()){};
01844     EffectSuite4(const EffectSuite4 &) = delete;
01845     EffectSuite4 &operator=(const EffectSuite4 &) = delete;
01846     EffectSuite4(EffectSuite4 &&) = delete;
01847     EffectSuite4 &operator=(EffectSuite4 &&) = delete;
01848
01849     A_long getLayerNumEffects(LayerPtr layer);
01850     EffectRefPtr getLayerEffectByIndex(LayerPtr layer,
01851                                        AEGP_EffectIndex layer_effect_index);
01852     AEGP_InstalledEffectKey
01853     getInstalledKeyFromLayerEffect(EffectRefPtr effect_ref);
01854     std::tuple<PF_ParamType, PF_ParamDefUnion>
01855     getEffectParamUnionByIndex(EffectRefPtr effect_ref,
01856                                PF_ParamIndex param_index);
01857     AE_EffectFlags getEffectFlags(EffectRefPtr effect_ref);
01858     void setEffectFlags(EffectRefPtr effect_ref,
01859                         AE_EffectFlags effect_flags_set_mask,
01860                         AE_EffectFlags effect_flags);
01861     void reorderEffect(EffectRefPtr effect_ref, A_long effect_index);
01862     void effectCallGeneric(EffectRefPtr effect_ref, const A_Time *timePT,
01863                            PF_Cmd effect_cmd, void *effect_extraPV);
01864     void disposeEffect(EffectRefPtr effect_ref);
01865     EffectRefPtr applyEffect(LayerPtr layer,
01866                              AEGP_InstalledEffectKey installed_effect_key);
01867     void deleteLayerEffect(EffectRefPtr effect_ref);
01868     A_long getNumInstalledEffects();
01869     AEGP_InstalledEffectKey
01870     getNextInstalledEffect(AEGP_InstalledEffectKey installed_effect_key);
01871     std::string getEffectName(AEGP_InstalledEffectKey installed_effect_key);
01872     std::string
```

```
01873     getEffectMatchName(AEGP_InstalledEffectKey installed_effect_key);
01874     std::string getEffectCategory(AEGP_InstalledEffectKey installed_effect_key);
01875     EffectRefPtr duplicateEffect(EffectRefPtr original_effect_ref);
01876     A_u_long numEffectMask(EffectRefPtr effect_ref);
01877     AEGP_MaskIDVal getEffectMaskID(EffectRefPtr effect_ref,
01878                                    A_u_long mask_indexL);
01879     StreamRefPtr addEffectMask(EffectRefPtr effect_ref, AEGP_MaskIDVal id_val);
01880     void removeEffectMask(EffectRefPtr effect_ref, AEGP_MaskIDVal id_val);
01881     StreamRefPtr setEffectMask(EffectRefPtr effect_ref, A_u_long mask_indexL,
01882                                AEGP_MaskIDVal id_val);
01883
01884   private:
01885     SuiteManager &m_suiteManager;
01886     static inline EffectRefPtr createPtr(AEGP_EffectRefH ref)
01887     {
01888         return std::shared_ptr<AEGP_EffectRefH>(new AEGP_EffectRefH(ref),
01889                                                 EffectDeleter());
01890     }
01891 };
01892
01893 enum class AE_MaskMBlur
01894 {
01895     SAME_AS_LAYER = AEGP_MaskMBlur_SAME_AS_LAYER,
01896     OFF = AEGP_MaskMBlur_OFF,
01897     ON = AEGP_MaskMBlur_ON
01898 };
01899
01900 enum class AE_MaskFeatherFalloff
01901 {
01902     SMOOTH = AEGP_MaskFeatherFalloff_SMOOTH,
01903     LINEAR = AEGP_MaskFeatherFalloff_LINEAR
01904 };
01905
01906 enum class AE_MaskFeatherInterp
01907 {
01908     NORMAL = AEGP_MaskFeatherInterp_NORMAL,
01909     HOLD_CW = AEGP_MaskFeatherInterp_HOLD_CW
01910 };
01911
01912 enum class AE_MaskFeatherType
01913 {
01914     OUTER = AEGP_MaskFeatherType_OUTER,
01915     INNER = AEGP_MaskFeatherType_INNER
01916 };
01917
01918 enum class AE_MaskMode
01919 {
01920     NONE = PF_MaskMode_NONE,
01921     ADD = PF_MaskMode_ADD,
01922     SUBTRACT = PF_MaskMode_SUBTRACT,
01923     INTERSECT = PF_MaskMode_INTERSECT,
01924     LIGHTEN = PF_MaskMode_LIGHTEN,
01925     DARKEN = PF_MaskMode_DARKEN,
01926     DIFF = PF_MaskMode_DIFFERENCE,
01927     ACCUM = PF_MaskMode_ACCUM
01928 };
01929
01930 class MaskSuite6
01931 {
01932   public:
01933     MaskSuite6() : m_suiteManager(SuiteManager::GetInstance()){};
01934     MaskSuite6(const MaskSuite6 &) = delete;
01935     MaskSuite6 &operator=(const MaskSuite6 &) = delete;
01936     MaskSuite6(MaskSuite6 &&) = delete;
01937     MaskSuite6 &operator=(MaskSuite6 &&) = delete;
01938
01939     A_long getLayerNumMasks(LayerPtr aegp_layerH);
01940     MaskRefPtr getLayerMaskByIndex(LayerPtr aegp_layerH,
01941                                    AEGP_MaskIndex mask_indexL);
01942     void disposeMask(MaskRefPtr mask_refH);
01943     bool getMaskInvert(MaskRefPtr mask_refH);
01944     void setMaskInvert(MaskRefPtr mask_refH, bool invertB);
01945     AE_MaskMode getMaskMode(MaskRefPtr mask_refH);
01946     void setMaskMode(MaskRefPtr maskH, AE_MaskMode mode);
01947     AE_MaskMBlur getMaskMotionBlurState(MaskRefPtr mask_refH);
01948     void setMaskMotionBlurState(MaskRefPtr mask_refH, AE_MaskMBlur blur_state);
01949     AE_MaskFeatherFalloff getMaskFeatherFalloff(MaskRefPtr mask_refH);
01950     void setMaskFeatherFalloff(MaskRefPtr mask_refH,
01951                                AE_MaskFeatherFalloff feather_falloffP);
01952     AEGP_MaskIDVal getMaskID(MaskRefPtr mask_refH);
01953     MaskRefPtr createNewMask(LayerPtr layerH, A_long mask_indexPL0);
01954     void deleteMaskFromLayer(MaskRefPtr mask_refH);
01955     ColorVal getMaskColor(MaskRefPtr mask_refH);
01956     void setMaskColor(MaskRefPtr mask_refH, ColorVal colorP);
01957     bool getMaskLockState(MaskRefPtr mask_refH);
01958     void setMaskLockState(MaskRefPtr mask_refH, bool lockB);
01959     bool getMaskIsRotoBezier(MaskRefPtr mask_refH);
```

```
01960     void setMaskIsRotoBezier(MaskRefPtr mask_refH, bool is_roto_bezierB);
01961     MaskRefPtr duplicateMask(MaskRefPtr orig_mask_refH);
01962
01963   private:
01964     SuiteManager &m_suiteManager;
01965     inline MaskRefPtr createPtr(AEGP_MaskRefH ref)
01966     {
01967         return std::shared_ptr<AEGP_MaskRefH>(new AEGP_MaskRefH(ref),
01968                                               MaskDeleter());
01969     }
01970 };
01971
01972 inline AEGP_MaskFeather createAEGP_MaskFeather()
01973 {
01974     AEGP_MaskFeather feather;
01975     feather.segment = 0;
01976     feather.segment_sF = 0;
01977     feather.radiusF = 0;
01978     feather.ui_corner_angleF = 0;
01979     feather.tensionF = 0;
01980     feather.interp = AEGP_MaskFeatherInterp_NORMAL;
01981     feather.type = AEGP_MaskFeatherType_OUTER;
01982     return feather;
01983 }
01984
01985 inline std::tuple<A_long, PF_FpLong, PF_FpLong, PF_FpShort, PF_FpShort,
01986                   AEGP_MaskFeatherInterp, AEGP_MaskFeatherType>
01987 getAEGP_MaskFeatherInfo(const AEGP_MaskFeather &feather)
01988 {
01989     return std::make_tuple(feather.segment, feather.segment_sF, feather.radiusF,
01990                            feather.ui_corner_angleF, feather.tensionF,
01991                            feather.interp, feather.type);
01992 }
01993
01994 class MaskOutlineSuite3
01995 {
01996   public:
01997     MaskOutlineSuite3() : m_suiteManager(SuiteManager::GetInstance()){};
01998     MaskOutlineSuite3(const MaskOutlineSuite3 &) = delete;
01999     MaskOutlineSuite3 &operator=(const MaskOutlineSuite3 &) = delete;
02000     MaskOutlineSuite3(MaskOutlineSuite3 &&) = delete;
02001     MaskOutlineSuite3 &operator=(MaskOutlineSuite3 &&) = delete;
02002
02003     bool isMaskOutlineOpen(MaskOutlineValPtr mask_outlineH);
02004     void setMaskOutlineOpen(MaskOutlineValPtr mask_outlineH, bool openB);
02005     A_long getMaskOutlineNumSegments(MaskOutlineValPtr mask_outlineH);
02006     AEGP_MaskVertex getMaskOutlineVertexInfo(MaskOutlineValPtr mask_outlineH,
02007                                              AEGP_VertexIndex which_pointL);
02008     void setMaskOutlineVertexInfo(MaskOutlineValPtr mask_outlineH,
02009                                   AEGP_VertexIndex which_pointL,
02010                                   const AEGP_MaskVertex &vertexP);
02011     void createVertex(MaskOutlineValPtr mask_outlineH,
02012                       AEGP_VertexIndex insert_position);
02013     void deleteVertex(MaskOutlineValPtr mask_outlineH, AEGP_VertexIndex index);
02014     A_long getMaskOutlineNumFeathers(MaskOutlineValPtr mask_outlineH);
02015     AEGP_MaskFeather
02016     getMaskOutlineFeatherInfo(MaskOutlineValPtr mask_outlineH,
02017                               AEGP_FeatherIndex which_featherL);
02018     void setMaskOutlineFeatherInfo(MaskOutlineValPtr mask_outlineH,
02019                                    AEGP_VertexIndex which_featherL,
02020                                    const AEGP_MaskFeather &featherP);
02021     AEGP_FeatherIndex
02022     createMaskOutlineFeather(MaskOutlineValPtr mask_outlineH,
02023                              const AEGP_MaskFeather &featherP0);
02024     void deleteMaskOutlineFeather(MaskOutlineValPtr mask_outlineH,
02025                                   AEGP_FeatherIndex index);
02026
02027   private:
02028     SuiteManager &m_suiteManager;
02029     static inline MaskOutlineValPtr createPtr(AEGP_MaskOutlineValH ref)
02030     {
02031         return std::make_shared<AEGP_MaskOutlineValH>(ref);
02032     }
02033 };
02034
02035 enum class AE_AlphaFlags
02036 {
02037     PREMUL = AEGP_AlphaPremul,
02038     INVERTED = AEGP_AlphaInverted,
02039     ALPHA_IGNORE = AEGP_AlphaIgnore
02040 };
02041
02042 inline AEGP_AlphaLabel createAEGP_AlphaLabel()
02043 {
02044     AEGP_AlphaLabel label;
02045     label.flags = 0;
02046     label.redCu = 0;
```

```
02047     label.greenCu = 0;
02048     label.blueCu = 0;
02049     return label;
02050 }
02051
02052 inline std::tuple<AEGP_AlphaFlags, A_u_char, A_u_char, A_u_char>
02053 getAEGP_AlphaLabelInfo(const AEGP_AlphaLabel &label)
02054 {
02055     return std::make_tuple(label.flags, label.redCu, label.greenCu,
02056                            label.blueCu);
02057 }
02058
02059 enum class AE_PulldownPhase
02060 {
02061     NO_PULLDOWN = AEGP_PulldownPhase_NO_PULLDOWN,
02062     WSSWW = AEGP_PulldownPhase_WSSWW,
02063     SSWWW = AEGP_PulldownPhase_SSWWW,
02064     SWWWS = AEGP_PulldownPhase_SWWWS,
02065     WWWSS = AEGP_PulldownPhase_WWWSS,
02066     WWSSW = AEGP_PulldownPhase_WWSSW,
02067     WWWSW = AEGP_PulldownPhase_WWWSW,
02068     WWSWW = AEGP_PulldownPhase_WWSWW,
02069     WSWWW = AEGP_PulldownPhase_WSWWW,
02070     SWWWW = AEGP_PulldownPhase_SWWWW,
02071     WWWWS = AEGP_PulldownPhase_WWWWS
02072 };
02073
02074 inline AEGP_LoopBehavior createAEGP_LoopBehavior()
02075 {
02076     AEGP_LoopBehavior behavior;
02077     behavior.loops = 0;
02078     behavior.reserved = 0;
02079     return behavior;
02080 }
02081
02082 inline std::tuple<A_long, A_long>
02083 getAEGP_LoopBehaviorInfo(const AEGP_LoopBehavior &behavior)
02084 {
02085     return std::make_tuple(behavior.loops, behavior.reserved);
02086 }
02087
02088 enum class AE_LayerDrawStyle
02089 {
02090     LAYER_BOUNDS = AEGP_LayerDrawStyle_LAYER_BOUNDS,
02091     DOCUMENT_BOUNDS = AEGP_LayerDrawStyle_DOCUMENT_BOUNDS
02092 };
02093
02094 inline AEGP_FootageLayerKey createAEGP_FootageLayerKey()
02095 {
02096     AEGP_FootageLayerKey key;
02097     key.layer_idL = AEGP_LayerID_UNKNOWN;
02098     key.layer_indexL = AEGP_LayerIndex_MERGED;
02099     key.nameAC[0] = '\0';
02100     key.layer_draw_style = AEGP_LayerDrawStyle_LAYER_BOUNDS;
02101     return key;
02102 }
02103
02104 inline std::tuple<A_long, A_long, std::string, AEGP_LayerDrawStyle>
02105 getAEGP_FootageLayerKeyInfo(const AEGP_FootageLayerKey &key)
02106 {
02107     return std::make_tuple(key.layer_idL, key.layer_indexL, key.nameAC,
02108                            key.layer_draw_style);
02109 }
02110
02111 inline AEGP_FileSequenceImportOptions createAEGP_FileSequenceImportOptions()
02112 {
02113     AEGP_FileSequenceImportOptions options;
02114     options.all_in_folderB = false;
02115     options.force_alphabeticalB = false;
02116     options.start_frameL = 0;
02117     options.end_frameL = 0;
02118     return options;
02119 }
02120
02121 inline std::tuple<bool, bool, A_long, A_long>
02122 getAEGP_FileSequenceImportOptionsInfo(
02123     const AEGP_FileSequenceImportOptions &options)
02124 {
02125     return std::make_tuple(options.all_in_folderB, options.force_alphabeticalB,
02126                            options.start_frameL, options.end_frameL);
02127 }
02128
02129 #define FOOTAGE_MAIN_FILE_INDEX 0
02130
02131 enum class AE_InterpretationStyle
02132 {
02133     NO_DIALOG_GUESS = AEGP_InterpretationStyle_NO_DIALOG_GUESS,
```

```
02134     DIALOG_OK = AEGP_InterpretationStyle_DIALOG_OK,
02135     NO_DIALOG_NO_GUESS = AEGP_InterpretationStyle_NO_DIALOG_NO_GUESS
02136 };
02137
02138 class FootageSuite5
02139 {
02140   public:
02141     FootageSuite5() : m_suiteManager(SuiteManager::GetInstance()){};
02142     FootageSuite5(const FootageSuite5 &) = delete;
02143     FootageSuite5 &operator=(const FootageSuite5 &) = delete;
02144     FootageSuite5(FootageSuite5 &&) = delete;
02145     FootageSuite5 &operator=(FootageSuite5 &&) = delete;
02146
02147     FootagePtr getMainFootageFromItem(ItemPtr itemH);
02148     FootagePtr getProxyFootageFromItem(ItemPtr itemH);
02149     std::tuple<A_long, A_long> getFootageNumFiles(FootagePtr footageH);
02150     std::string getFootagePath(FootagePtr footageH, A_long frame_numL,
02151                                A_long file_indexL);
02152     AEGP_FootageSignature getFootageSignature(FootagePtr footageH);
02153     FootagePtr newFootage(
02154
02155         std::string pathZ, AEGP_FootageLayerKey layer_infoP0,
02156         AEGP_FileSequenceImportOptions *sequence_optionsP0,
02157         AE_InterpretationStyle interp_style);
02158     ItemPtr addFootageToProject(FootagePtr footageH, ItemPtr folderH);
02159     void setItemProxyFootage(FootagePtr footageH, ItemPtr itemH);
02160     void replaceItemMainFootage(FootagePtr footageH, ItemPtr itemH);
02161     void disposeFootage(FootagePtr footageH);
02162     AEGP_FootageInterp getFootageInterpretation(ItemPtr itemH, bool proxyB);
02163     void setFootageInterpretation(ItemPtr itemH, bool proxyB,
02164                                   const AEGP_FootageInterp *interpP);
02165     AEGP_FootageLayerKey getFootageLayerKey(FootagePtr footageH);
02166     FootagePtr newPlaceholderFootage(std::string nameZ, A_long width,
02167                                      A_long height, A_Time durationPT);
02168     FootagePtr newPlaceholderFootageWithPath(std::string pathZ,
02169                                              AE_Platform path_platform,
02170                                              AEIO_FileType file_type,
02171                                              A_long widthL, A_long heightL,
02172                                              A_Time durationPT);
02173     FootagePtr newSolidFootage(std::string nameZ, A_long width, A_long height,
02174                                ColorVal colorP);
02175     ColorVal getSolidFootageColor(ItemPtr itemH, bool proxyB);
02176     void setSolidFootageColor(ItemPtr itemH, bool proxyB, ColorVal colorP);
02177     void setSolidFootageDimensions(ItemPtr itemH, bool proxyB, A_long widthL,
02178                                    A_long heightL);
02179     AEGP_SoundDataFormat getFootageSoundDataFormat(FootagePtr footageH);
02180     AEGP_FileSequenceImportOptions
02181     getFootageSequenceImportOptions(FootagePtr footageH);
02182
02183   private:
02184     SuiteManager &m_suiteManager;
02185     static inline FootagePtr createPtr(AEGP_FootageH ref)
02186     {
02187         return std::shared_ptr<AEGP_FootageH>(new AEGP_FootageH(ref),
02188                                               FootageDeleter());
02189     }
02190 };
02191
02192 enum class AE_PluginPathType
02193 {
02194     PLUGIN = AEGP_GetPathTypes_PLUGIN,
02195     USER_PLUGIN = AEGP_GetPathTypes_USER_PLUGIN,
02196     ALLUSER_PLUGIN = AEGP_GetPathTypes_ALLUSER_PLUGIN,
02197     APP = AEGP_GetPathTypes_APP
02198 };
02199
02200 class UtilitySuite6
02201 {
02202   public:
02203     UtilitySuite6() : m_suiteManager(SuiteManager::GetInstance()){};
02204     UtilitySuite6(const UtilitySuite6 &) = delete;
02205     UtilitySuite6 &operator=(const UtilitySuite6 &) = delete;
02206     UtilitySuite6(UtilitySuite6 &&) = delete;
02207     UtilitySuite6 &operator=(UtilitySuite6 &&) = delete;
02208
02209     void reportInfo(const std::string &info_string);
02210     void reportInfoUnicode(const std::string &info_string);
02211     std::tuple<A_short, A_short> getDriverPluginInitFuncVersion();
02212     std::tuple<A_short, A_short> getDriverImplementationVersion();
02213     void startQuietErrors();
02214     void endQuietErrors(bool report_quieted_errorsB);
02215     std::string getLastErrorMessage(A_long buffer_size);
02216     void startUndoGroup(const std::string &undo_name);
02217     void endUndoGroup();
02218     void *getMainHWND();
02219     void showHideAllFloaters(bool include_tool_palB);
02220     ColorVal getPaintPalForeColor();                        /*Brush Tool Panel*/
```

```
02221     ColorVal getPaintPalBackColor();                       /*Brush Tool Panel*/
02222     void setPaintPalForeColor(const ColorVal &fore_color); /*Brush Tool Panel*/
02223     void setPaintPalBackColor(const ColorVal &back_color); /*Brush Tool Panel*/
02224     std::tuple<bool, ColorVal> getCharPalFillColor();   /*Character Tool Panel*/
02225     std::tuple<bool, ColorVal> getCharPalStrokeColor(); /*Character Tool Panel*/
02226     void
02227     setCharPalFillColor(const ColorVal &fill_color); /*Character Tool Panel*/
02228     void setCharPalStrokeColor(
02229         const ColorVal &stroke_color);    /*Character Tool Panel*/
02230     bool charPalIsFillColorUIFrontmost(); /*Returns whether or not the fill
02231                                             color is frontmost. If it isnt,
02232                                             the stroke color is frontmost.*/
02233     A_Ratio convertFpLongToHSFRatio(A_FpLong numberF);
02234     A_FpLong convertHSFRatioToFpLong(A_Ratio ratioR);
02235     void causeIdleRoutinesToBeCalled();
02236     bool getSuppressInteractiveUI();
02237     void writeToOSConsole(const std::string &text);
02238     void writeToDebugLog(const std::string &subsystem,
02239                          const std::string &eventType, const std::string &text);
02240     std::string getPluginPath(AE_PluginPathType path_type);
02241
02242   private:
02243     SuiteManager &m_suiteManager;
02244 };
02245
02246 enum class AE_RenderQueueState
02247 {
02248     STOPPED = AEGP_RenderQueueState_STOPPED,
02249     PAUSED = AEGP_RenderQueueState_PAUSED,
02250     RENDERING = AEGP_RenderQueueState_RENDERING
02251 };
02252
02253 enum class AE_RenderItemStatus
02254 {
02255     NONE = AEGP_RenderItemStatus_NONE,
02256     WILL_CONTINUE = AEGP_RenderItemStatus_WILL_CONTINUE,
02257     NEEDS_OUTPUT = AEGP_RenderItemStatus_NEEDS_OUTPUT,
02258     UNQUEUED = AEGP_RenderItemStatus_UNQUEUED,
02259     QUEUED = AEGP_RenderItemStatus_QUEUED,
02260     RENDERING = AEGP_RenderItemStatus_RENDERING,
02261     USER_STOPPED = AEGP_RenderItemStatus_USER_STOPPED,
02262     ERR_STOPPED = AEGP_RenderItemStatus_ERR_STOPPED,
02263     DONE = AEGP_RenderItemStatus_DONE
02264 };
02265
02266 enum class AE_LogType
02267 {
02268     NONE = AEGP_LogType_NONE,
02269     ERRORS_ONLY = AEGP_LogType_ERRORS_ONLY,
02270     PLUS_SETTINGS = AEGP_LogType_PLUS_SETTINGS,
02271     PER_FRAME_INFO = AEGP_LogType_PER_FRAME_INFO
02272 };
02273
02274 enum class AE_EmbeddingType
02275 {
02276     NONE = AEGP_Embedding_NONE,
02277     NOTHING = AEGP_Embedding_NOTHING,
02278     LINK = AEGP_Embedding_LINK,
02279     LINK_AND_COPY = AEGP_Embedding_LINK_AND_COPY
02280 };
02281
02282 enum class AE_PostRenderAction
02283 {
02284     NONE = AEGP_PostRenderOptions_NONE,
02285     IMPORT = AEGP_PostRenderOptions_IMPORT,
02286     IMPORT_AND_REPLACE_USAGE = AEGP_PostRenderOptions_IMPORT_AND_REPLACE_USAGE,
02287     SET_PROXY = AEGP_PostRenderOptions_SET_PROXY
02288 };
02289
02290 enum class AE_OutputTypes
02291 {
02292     NONE = AEGP_OutputType_NONE,
02293     VIDEO = AEGP_OutputType_VIDEO,
02294     AUDIO = AEGP_OutputType_AUDIO
02295 };
02296
02297 enum class AE_VideoChannels
02298 {
02299     NONE = AEGP_VideoChannels_NONE,
02300     RGB = AEGP_VideoChannels_RGB,
02301     RGBA = AEGP_VideoChannels_RGBA,
02302     ALPHA = AEGP_VideoChannels_ALPHA
02303 };
02304
02305 enum class AE_StretchQuality
02306 {
02307     NONE = AEGP_StretchQual_NONE,
```

```
02308     LOW = AEGP_StretchQual_LOW,
02309     HIGH = AEGP_StretchQual_HIGH
02310 };
02311
02312 enum class AE_OutputColorType
02313 {
02314     STRAIGHT = AEGP_OutputColorType_STRAIGHT,
02315     PREMUL = AEGP_OutputColorType_PREMUL
02316 };
02317
02318 class RenderQueueSuite1
02319 {
02320   public:
02321     RenderQueueSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
02322     RenderQueueSuite1(const RenderQueueSuite1 &) = delete;
02323     RenderQueueSuite1 &operator=(const RenderQueueSuite1 &) = delete;
02324     RenderQueueSuite1(RenderQueueSuite1 &&) = delete;
02325     RenderQueueSuite1 &operator=(RenderQueueSuite1 &&) = delete;
02326
02327     void addCompToRenderQueue(CompPtr comp, const std::string &path);
02328     void setRenderQueueState(AE_RenderQueueState state);
02329     AE_RenderQueueState getRenderQueueState();
02330
02331   private:
02332     SuiteManager &m_suiteManager;
02333 };
02334
02335 class RenderQueueItemSuite4
02336 {
02337   public:
02338     RenderQueueItemSuite4() : m_suiteManager(SuiteManager::GetInstance()){};
02339     RenderQueueItemSuite4(const RenderQueueItemSuite4 &) = delete;
02340     RenderQueueItemSuite4 &operator=(const RenderQueueItemSuite4 &) = delete;
02341     RenderQueueItemSuite4(RenderQueueItemSuite4 &&) = delete;
02342     RenderQueueItemSuite4 &operator=(RenderQueueItemSuite4 &&) = delete;
02343
02344     A_long getNumRQItems();
02345     RQItemRefPtr getRQItemByIndex(A_long rq_item_index);
02346     RQItemRefPtr getNextRQItem(RQItemRefPtr current_rq_item);
02347     A_long getNumOutputModulesForRQItem(RQItemRefPtr rq_item);
02348     AE_RenderItemStatus getRenderState(RQItemRefPtr rq_item);
02349     void setRenderState(RQItemRefPtr rq_item, AE_RenderItemStatus status);
02350     A_Time getStartedTime(RQItemRefPtr rq_item);
02351     A_Time getElapsedTime(RQItemRefPtr rq_item);
02352     AE_LogType getLogType(RQItemRefPtr rq_item);
02353     void setLogType(RQItemRefPtr rq_item, AE_LogType logtype);
02354     void removeOutputModule(RQItemRefPtr rq_item, OutputModuleRefPtr outmod);
02355     std::string getComment(RQItemRefPtr rq_item);
02356     void setComment(RQItemRefPtr rq_item, const std::string &comment);
02357     CompPtr getCompFromRQItem(RQItemRefPtr rq_item);
02358     void deleteRQItem(RQItemRefPtr rq_item);
02359
02360   private:
02361     SuiteManager &m_suiteManager;
02362 };
02363
02364 class OutputModuleSuite4
02365 {
02366   public:
02367     OutputModuleSuite4() : m_suiteManager(SuiteManager::GetInstance()){};
02368     OutputModuleSuite4(const OutputModuleSuite4 &) = delete;
02369     OutputModuleSuite4 &operator=(const OutputModuleSuite4 &) = delete;
02370     OutputModuleSuite4(OutputModuleSuite4 &&) = delete;
02371     OutputModuleSuite4 &operator=(OutputModuleSuite4 &&) = delete;
02372
02373     OutputModuleRefPtr getOutputModuleByIndex(RQItemRefPtr rq_itemH,
02374                                               A_long outmod_indexL);
02375     AE_EmbeddingType getEmbedOptions(RQItemRefPtr rq_itemH,
02376                                      OutputModuleRefPtr outmodH);
02377     void setEmbedOptions(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02378                          AE_EmbeddingType embed_options);
02379     AE_PostRenderAction getPostRenderAction(RQItemRefPtr rq_itemH,
02380                                             OutputModuleRefPtr outmodH);
02381     void setPostRenderAction(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02382                              AE_PostRenderAction post_render_action);
02383     AE_OutputTypes getEnabledOutputs(RQItemRefPtr rq_itemH,
02384                                      OutputModuleRefPtr outmodH);
02385     void setEnabledOutputs(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02386                            AE_OutputTypes enabled_types);
02387     AE_VideoChannels getOutputChannels(RQItemRefPtr rq_itemH,
02388                                        OutputModuleRefPtr outmodH);
02389     void setOutputChannels(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02390                            AE_VideoChannels output_channels);
02391     std::tuple<bool, AE_StretchQuality, bool>
02392     getStretchInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH);
02393     void setStretchInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02394                         bool is_enabledB, AE_StretchQuality stretch_quality);
```

```
02395     std::tuple<bool, A_Rect> getCropInfo(RQItemRefPtr rq_itemH,
02396                                          OutputModuleRefPtr outmodH);
02397     void setCropInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02398                      bool enableB, A_Rect crop_rect);
02399     std::tuple<AEGP_SoundDataFormat, bool>
02400     getSoundFormatInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH);
02401     void setSoundFormatInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02402                             AEGP_SoundDataFormat sound_format_info,
02403                             bool audio_enabledB);
02404     std::string getOutputFilePath(RQItemRefPtr rq_itemH,
02405                                   OutputModuleRefPtr outmodH);
02406     void setOutputFilePath(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH,
02407                            const std::string &path);
02408     OutputModuleRefPtr addDefaultOutputModule(RQItemRefPtr rq_itemH);
02409     std::tuple<std::string, std::string, bool, bool>
02410     getExtraOutputModuleInfo(RQItemRefPtr rq_itemH, OutputModuleRefPtr outmodH);
02411
02412   private:
02413     SuiteManager &m_suiteManager;
02414 };
02415
02416 enum class AE_WorldType
02417 {
02418     NONE = AEGP_WorldType_NONE,
02419     W8 = AEGP_WorldType_8,
02420     W16 = AEGP_WorldType_16,
02421     W32 = AEGP_WorldType_32
02422 };
02423
02424 enum class AE_MatteMode
02425 {
02426     STRAIGHT = AEGP_MatteMode_STRAIGHT,
02427     PREMUL_BLACK = AEGP_MatteMode_PREMUL_BLACK,
02428     PREMUL_BG_COLOR = AEGP_MatteMode_PREMUL_BG_COLOR
02429 };
02430
02431 enum class AE_ChannelOrder
02432 {
02433     ARGB = AEGP_ChannelOrder_ARGB,
02434     BGRA = AEGP_ChannelOrder_BGRA
02435 };
02436
02437 enum class AE_ItemQuality
02438 {
02439     DRAFT = AEGP_ItemQuality_DRAFT,
02440     BEST = AEGP_ItemQuality_BEST
02441 };
02442
02443 class WorldSuite3
02444 {
02445   public:
02446     WorldSuite3() : m_suiteManager(SuiteManager::GetInstance()){};
02447     WorldSuite3(const WorldSuite3 &) = delete;
02448     WorldSuite3 &operator=(const WorldSuite3 &) = delete;
02449     WorldSuite3(WorldSuite3 &&) = delete;
02450     WorldSuite3 &operator=(WorldSuite3 &&) = delete;
02451
02452     WorldPtr newWorld(AE_WorldType type, A_long widthL, A_long heightL);
02453     AE_WorldType getType(WorldPtr worldH);
02454     std::tuple<A_long, A_long> getSize(WorldPtr worldH);
02455     A_u_long getRowBytes(WorldPtr worldH);
02456     PF_Pixel8 *getBaseAddr8(WorldPtr worldH);
02457     PF_Pixel16 *getBaseAddr16(WorldPtr worldH);
02458     PF_PixelFloat *getBaseAddr32(WorldPtr worldH);
02459     PF_EffectWorld fillOutPFEffectWorld(WorldPtr worldH);
02460     void fastBlur(A_FpLong radiusF, PF_ModeFlags mode, PF_Quality quality,
02461                   WorldPtr worldH);
02462     PlatformWorldPtr newPlatformWorld(AEGP_WorldType type, A_long widthL,
02463                                       A_long heightL);
02464     WorldPtr newReferenceFromPlatformWorld(PlatformWorldPtr platform_worldH);
02465
02466       static inline WorldPtr createPtr(AEGP_WorldH ref)
02467     {
02468         return std::shared_ptr<AEGP_WorldH>(new AEGP_WorldH(ref),
02469                                             WorldDeleter());
02470     }
02471     static inline PlatformWorldPtr createPlatformPtr(AEGP_PlatformWorldH ref)
02472     {
02473         return std::shared_ptr<AEGP_PlatformWorldH>(
02474             new AEGP_PlatformWorldH(ref), PlatformDeleter());
02475     }
02476   private:
02477     SuiteManager &m_suiteManager;
02478
02479 };
02480
02481 class RenderOptionsSuite4
```

```
02482 {
02483   public:
02484     RenderOptionsSuite4() : m_suiteManager(SuiteManager::GetInstance()){};
02485     RenderOptionsSuite4(const RenderOptionsSuite4 &) = delete;
02486     RenderOptionsSuite4 &operator=(const RenderOptionsSuite4 &) = delete;
02487     RenderOptionsSuite4(RenderOptionsSuite4 &&) = delete;
02488     RenderOptionsSuite4 &operator=(RenderOptionsSuite4 &&) = delete;
02489
02490     RenderOptionsPtr newFromItem(ItemPtr itemH);
02491     RenderOptionsPtr duplicate(RenderOptionsPtr optionsH);
02492     void setTime(RenderOptionsPtr optionsH, A_Time time);
02493     A_Time getTime(RenderOptionsPtr optionsH);
02494     void setTimeStep(RenderOptionsPtr optionsH, A_Time time_step);
02495     A_Time getTimeStep(RenderOptionsPtr optionsH);
02496     void setFieldRender(RenderOptionsPtr optionsH, PF_Field field_render);
02497     PF_Field getFieldRender(RenderOptionsPtr optionsH);
02498     void setWorldType(RenderOptionsPtr optionsH, AE_WorldType type);
02499     AE_WorldType getWorldType(RenderOptionsPtr optionsH);
02500     void setDownsampleFactor(RenderOptionsPtr optionsH, A_short x, A_short y);
02501     std::tuple<A_short, A_short> getDownsampleFactor(RenderOptionsPtr optionsH);
02502     void setRegionOfInterest(RenderOptionsPtr optionsH, const A_LRect *roiP);
02503     A_LRect getRegionOfInterest(RenderOptionsPtr optionsH);
02504     void setMatteMode(RenderOptionsPtr optionsH, AE_MatteMode mode);
02505     AE_MatteMode getMatteMode(RenderOptionsPtr optionsH);
02506     void setChannelOrder(RenderOptionsPtr optionsH,
02507                          AE_ChannelOrder channel_order);
02508     AE_ChannelOrder getChannelOrder(RenderOptionsPtr optionsH);
02509     bool getRenderGuideLayers(RenderOptionsPtr optionsH);
02510     void setRenderGuideLayers(RenderOptionsPtr optionsH, bool render_themB);
02511
02512   private:
02513     SuiteManager &m_suiteManager;
02514     static inline RenderOptionsPtr createPtr(AEGP_RenderOptionsH ref)
02515     {
02516         return std::shared_ptr<AEGP_RenderOptionsH>(
02517             new AEGP_RenderOptionsH(ref), RenderOptionsDeleter());
02518     }
02519 };
02520
02521 class LayerRenderOptionsSuite2
02522 {
02523   public:
02524     LayerRenderOptionsSuite2() : m_suiteManager(SuiteManager::GetInstance()){};
02525     LayerRenderOptionsSuite2(const LayerRenderOptionsSuite2 &) = delete;
02526     LayerRenderOptionsSuite2 &
02527     operator=(const LayerRenderOptionsSuite2 &) = delete;
02528     LayerRenderOptionsSuite2(LayerRenderOptionsSuite2 &&) = delete;
02529     LayerRenderOptionsSuite2 &operator=(LayerRenderOptionsSuite2 &&) = delete;
02530
02531     LayerRenderOptionsPtr newFromLayer(LayerPtr layer);
02532     LayerRenderOptionsPtr newFromUpstreamOfEffect(EffectRefPtr effect_ref);
02533     LayerRenderOptionsPtr newFromDownstreamOfEffect(EffectRefPtr effect_ref);
02534     LayerRenderOptionsPtr duplicate(LayerRenderOptionsPtr optionsH);
02535     void dispose(LayerRenderOptionsPtr optionsH);
02536     void setTime(LayerRenderOptionsPtr optionsH, A_Time time);
02537     A_Time getTime(LayerRenderOptionsPtr optionsH);
02538     void setTimeStep(LayerRenderOptionsPtr optionsH, A_Time time_step);
02539     A_Time getTimeStep(LayerRenderOptionsPtr optionsH);
02540     void setWorldType(LayerRenderOptionsPtr optionsH, AE_WorldType type);
02541     AE_WorldType getWorldType(LayerRenderOptionsPtr optionsH);
02542     void setDownsampleFactor(LayerRenderOptionsPtr optionsH, A_short x,
02543                              A_short y);
02544     std::tuple<A_short, A_short>
02545     getDownsampleFactor(LayerRenderOptionsPtr optionsH);
02546     void setMatteMode(LayerRenderOptionsPtr optionsH, AE_MatteMode mode);
02547     AE_MatteMode getMatteMode(LayerRenderOptionsPtr optionsH);
02548
02549   private:
02550     SuiteManager &m_suiteManager;
02551     static inline LayerRenderOptionsPtr createPtr(AEGP_LayerRenderOptionsH ref)
02552     {
02553         return std::shared_ptr<AEGP_LayerRenderOptionsH>(
02554             new AEGP_LayerRenderOptionsH(ref), LayerRenderOptionsDeleter());
02555     }
02556 };
02557
02558 class RenderSuite5
02559 {
02560   public:
02561     RenderSuite5() : m_suiteManager(SuiteManager::GetInstance()){};
02562     RenderSuite5(const RenderSuite5 &) = delete;
02563     RenderSuite5 &operator=(const RenderSuite5 &) = delete;
02564     RenderSuite5(RenderSuite5 &&) = delete;
02565     RenderSuite5 &operator=(RenderSuite5 &&) = delete;
02566
02567     FrameReceiptPtr renderAndCheckoutFrame(RenderOptionsPtr optionsH);
02568     FrameReceiptPtr renderAndCheckoutLayerFrame(
```

```
02569         LayerRenderOptionsPtr optionsH);
02570
02571     WorldPtr getReceiptWorld(FrameReceiptPtr receiptH);
02572     A_LRect getRenderedRegion(FrameReceiptPtr receiptH);
02573     bool isRenderedFrameSufficient(RenderOptionsPtr rendered_optionsH,
02574                                    RenderOptionsPtr proposed_optionsH);
02575     TimeStampPtr getCurrentTimestamp();
02576     bool hasItemChangedSinceTimestamp(ItemPtr itemH,A_Time start_timeP,
02577                                       A_Time durationP,
02578                                       TimeStampPtr time_stampP);
02579     bool isItemWorthwhileToRender(RenderOptionsPtr roH,
02580                                       TimeStampPtr time_stampP);
02581     void checkinRenderedFrame(RenderOptionsPtr roH,
02582                               TimeStampPtr time_stampP,
02583                               A_u_long ticks_to_renderL,
02584                               PlatformWorldPtr imageH);
02585     std::string getReceiptGuid(FrameReceiptPtr receiptH);
02586
02587   private:
02588     SuiteManager &m_suiteManager;
02589     static inline FrameReceiptPtr createPtr(AEGP_FrameReceiptH ref)
02590     {
02591         return std::shared_ptr<AEGP_FrameReceiptH>(new AEGP_FrameReceiptH(ref),
02592                                                    FrameReceiptDeleter());
02593     }
02594 };
02595 enum class AE_CollectionItemType
02596 {
02597     NONE = AEGP_CollectionItemType_NONE,
02598     LAYER = AEGP_CollectionItemType_LAYER,
02599     MASK = AEGP_CollectionItemType_MASK,
02600     EFFECT = AEGP_CollectionItemType_EFFECT,
02601     STREAM = AEGP_CollectionItemType_STREAM,
02602     KEYFRAME = AEGP_CollectionItemType_KEYFRAME,
02603     MASK_VERTEX = AEGP_CollectionItemType_MASK_VERTEX,
02604     STREAMREF = AEGP_CollectionItemType_STREAMREF
02605 };
02606
02607 enum class AE_StreamCollectionItemType
02608 {
02609     NONE = AEGP_StreamCollectionItemType_NONE,
02610     LAYER = AEGP_StreamCollectionItemType_LAYER,
02611     MASK = AEGP_StreamCollectionItemType_MASK,
02612     EFFECT = AEGP_StreamCollectionItemType_EFFECT
02613 };
02614
02615 class CollectionSuite2
02616 {
02617   public:
02618     CollectionSuite2() : m_suiteManager(SuiteManager::GetInstance()){};
02619     CollectionSuite2(const CollectionSuite2 &) = delete;
02620     CollectionSuite2 &operator=(const CollectionSuite2 &) = delete;
02621     CollectionSuite2(CollectionSuite2 &&) = delete;
02622     CollectionSuite2 &operator=(CollectionSuite2 &&) = delete;
02623
02624     Collection2Ptr newCollection();
02625     void disposeCollection(Collection2Ptr collectionH);
02626     A_long getCollectionNumItems(Collection2Ptr collectionH);
02627     AEGP_CollectionItemV2 getCollectionItemByIndex(Collection2Ptr collectionH,
02628                                                    A_long indexL);
02629     void collectionPushBack(Collection2Ptr collectionH,
02630                             const AEGP_CollectionItemV2 &itemP);
02631     void collectionErase(Collection2Ptr collectionH, A_long index_firstL,
02632                          A_long index_lastL);
02633
02634   private:
02635     SuiteManager &m_suiteManager;
02636     static inline Collection2Ptr createPtr(AEGP_Collection2H ref)
02637     {
02638         return std::shared_ptr<AEGP_Collection2H>(new AEGP_Collection2H(ref),
02639                                                   CollectionDeleter());
02640     }
02641 };
02642
02643 enum class AE_WindowType
02644 {
02645     NONE = AEGP_WindType_NONE,
02646     PROJECT = AEGP_WindType_PROJECT,
02647     COMP = AEGP_WindType_COMP,
02648     TIME_LAYOUT = AEGP_WindType_TIME_LAYOUT,
02649     LAYER = AEGP_WindType_LAYER,
02650     FOOTAGE = AEGP_WindType_FOOTAGE,
02651     RENDER_QUEUE = AEGP_WindType_RENDER_QUEUE,
02652     QT = AEGP_WindType_QT,
02653     DIALOG = AEGP_WindType_DIALOG,
02654     FLOWCHART = AEGP_WindType_FLOWCHART,
02655     EFFECT = AEGP_WindType_EFFECT,
```

```
02656     OTHER = AEGP_WindType_OTHER
02657 };
02658
02659 class RegisterSuite5
02660 {
02661   public:
02662     RegisterSuite5() : m_suiteManager(SuiteManager::GetInstance()){};
02663     RegisterSuite5(const RegisterSuite5 &) = delete;
02664     RegisterSuite5 &operator=(const RegisterSuite5 &) = delete;
02665     RegisterSuite5(RegisterSuite5 &&) = delete;
02666     RegisterSuite5 &operator=(RegisterSuite5 &&) = delete;
02667
02668     void registerCommandHook(AEGP_HookPriority hook_priority,
02669                              AEGP_Command command,
02670                              AEGP_CommandHook command_hook_func,
02671                              AEGP_CommandRefcon refconP);
02672     void registerUpdateMenuHook(AEGP_UpdateMenuHook update_menu_hook_func,
02673                                 AEGP_UpdateMenuRefcon refconP);
02674     void registerDeathHook(AEGP_DeathHook death_hook_func,
02675                            AEGP_DeathRefcon refconP);
02676     void registerIdleHook(AEGP_IdleHook idle_hook_func,
02677                           AEGP_IdleRefcon refconP);
02678     void registerPresetLocalizationString(const std::string &english_nameZ,
02679                                           const std::string &localized_nameZ);
02680
02681   private:
02682     SuiteManager &m_suiteManager;
02683 };
02684
02685 enum class AE_MenuID
02686 {
02687     NONE = AEGP_Menu_NONE,
02688     APPLE = AEGP_Menu_APPLE,
02689     FILE = AEGP_Menu_FILE,
02690     EDIT = AEGP_Menu_EDIT,
02691     COMPOSITION = AEGP_Menu_COMPOSITION,
02692     LAYER = AEGP_Menu_LAYER,
02693     EFFECT = AEGP_Menu_EFFECT,
02694     WINDOW = AEGP_Menu_WINDOW,
02695     FLOATERS = AEGP_Menu_FLOATERS,
02696     KF_ASSIST = AEGP_Menu_KF_ASSIST,
02697     IMPORT = AEGP_Menu_IMPORT,
02698     SAVE_FRAME_AS = AEGP_Menu_SAVE_FRAME_AS,
02699     PREFS = AEGP_Menu_PREFS,
02700     EXPORT = AEGP_Menu_EXPORT,
02701     ANIMATION = AEGP_Menu_ANIMATION,
02702     PURGE = AEGP_Menu_PURGE,
02703     NEW = AEGP_Menu_NEW
02704 };
02705
02706 #define INSERT_SORTED (-2)
02707 #define INSERT_BOTTOM (-1)
02708 #define INSERT_TOP 0
02709
02710 class CommandSuite1
02711 {
02712   public:
02713     CommandSuite1() : m_suiteManager(SuiteManager::GetInstance()){};
02714     CommandSuite1(const CommandSuite1 &) = delete;
02715     CommandSuite1 &operator=(const CommandSuite1 &) = delete;
02716     CommandSuite1(CommandSuite1 &&) = delete;
02717     CommandSuite1 &operator=(CommandSuite1 &&) = delete;
02718
02719     AEGP_Command getUniqueCommand();
02720     void insertMenuCommand(AEGP_Command command, const std::string &nameZ,
02721                            AE_MenuID menu_id, A_long after_itemL);
02722     void removeMenuCommand(AEGP_Command command);
02723     void setMenuCommandName(AEGP_Command command, const std::string &nameZ);
02724     void enableCommand(AEGP_Command command);
02725     void disableCommand(AEGP_Command command);
02726     void checkMarkMenuCommand(AEGP_Command command, A_Boolean checkB);
02727     void doCommand(AEGP_Command command);
02728
02729   private:
02730     SuiteManager &m_suiteManager;
02731 };
02732
02733 #endif // AE_MAIN_HPP
```

7.7 AEGP/Core/Base/Collection.hpp File Reference

A class to represent a collection of items.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
#include "AETK/AEGP/Memory/AEMemory.hpp"
```

Classes

- class Collection< T >

  A class to represent a collection of items.

7.7.1 Detailed Description

A class to represent a collection of items.

Collection.hpp

Author

tjerf

Date

March 2024

7.8 Collection.hpp

Go to the documentation of this file.

```
00001 /*************************************************************/ /**
00002                                                                * \file
00003                                                                *Collection.hpp
00004                                                                * \brief  A
00005                                                                *class to
00006                                                                *represent
00007                                                                *a
00008                                                                *collection
00009                                                                *of items
00010                                                                *
00011                                                                * \author
00012                                                                *tjerf
00013                                                                * \date
00014                                                                *March 2024
00015
      ***************************************************************************/
00016
00017 #ifndef COLLECTION_HPP
00018 #define COLLECTION_HPP
00019
00020 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00021 #include "AETK/AEGP/Memory/AEMemory.hpp"
00022
00023 /**
00024  * \class Collection
00025  * \brief A class to represent a collection of items.
00026  *
00027  * This class provides various methods to manipulate and perform operations on a
00028  * collection of items. It supports functionalities such as appending,
00029  * extending, inserting, removing, popping, clearing, reversing, sorting, and
00030  * more.
00031  *
00032  * \tparam T The type of items in the collection.
00033  */
00034 template <typename T> class Collection
00035 {
00036   public:
00037     /**
00038      * \brief Default constructor.
```

```
00039      */
00040     Collection() = default;
00041
00042     /**
00043      * \brief Constructor that initializes the collection with the given items.
00044      *
00045      * \param collection The initial collection of items.
00046      */
00047     Collection(std::vector<T> collection) : m_collection(collection) {}
00048
00049     /**
00050      * \brief Destructor.
00051      */
00052     ~Collection() = default;
00053
00054     /**
00055      * \brief Get the collection of items.
00056      *
00057      * \return std::vector<T> The collection of items.
00058      */
00059     std::vector<T> GetCollection() { return m_collection; }
00060
00061     /**
00062      * \brief Set the collection of items.
00063      *
00064      * \param collection The new collection of items.
00065      */
00066     void SetCollection(std::vector<T> collection) { m_collection = collection; }
00067
00068     /**
00069      * \brief Get an iterator pointing to the beginning of the collection.
00070      *
00071      * \return auto An iterator pointing to the beginning of the collection.
00072      */
00073     auto begin() { return m_collection.begin(); }
00074
00075     /**
00076      * \brief Get an iterator pointing to the end of the collection.
00077      *
00078      * \return auto An iterator pointing to the end of the collection.
00079      */
00080     auto end() { return m_collection.end(); }
00081
00082     /**
00083      * \brief Get the size of the collection.
00084      *
00085      * \return size_t The size of the collection.
00086      */
00087     virtual size_t size() { return m_collection.size(); }
00088
00089     /**
00090      * \brief Get the item at the specified index.
00091      *
00092      * \param index The index of the item.
00093      * \return T The item at the specified index.
00094      */
00095     virtual T operator[](size_t index) { return m_collection[index]; }
00096
00097     /**
00098      * \brief Append an item to the end of the collection.
00099      *
00100      * \param item The item to be appended.
00101      */
00102     virtual void append(T item) { m_collection.push_back(item); }
00103
00104     /**
00105      * \brief Extend the collection by appending multiple items.
00106      *
00107      * \param items The items to be appended.
00108      */
00109     virtual void extend(std::vector<T> items)
00110     {
00111         m_collection.insert(m_collection.end(), items.begin(), items.end());
00112     }
00113
00114     /**
00115      * \brief Insert an item at the specified index.
00116      *
00117      * \param index The index at which the item should be inserted.
00118      * \param item The item to be inserted.
00119      */
00120     virtual void insert(size_t index, T item)
00121     {
00122         m_collection.insert(m_collection.begin() + index, item);
00123     }
00124
00125     /**
```

```
00126      * \brief Remove the first occurrence of an item from the collection.
00127      *
00128      * \param item The item to be removed.
00129      */
00130     virtual void remove(T item)
00131     {
00132         m_collection.erase(
00133             std::remove(m_collection.begin(), m_collection.end(), item),
00134             m_collection.end());
00135     }
00136
00137     /**
00138      * \brief Remove the item at the specified index.
00139      *
00140      * \param index The index of the item to be removed.
00141      */
00142     virtual void pop(size_t index)
00143     {
00144         m_collection.erase(m_collection.begin() + index);
00145     }
00146
00147     /**
00148      * \brief Clear the collection, removing all items.
00149      */
00150     virtual void clear() { m_collection.clear(); }
00151
00152     /**
00153      * \brief Get the index of the first occurrence of an item in the
00154      * collection.
00155      *
00156      * \param item The item to search for.
00157      * \return size_t The index of the item, or -1 if not found.
00158      */
00159     virtual size_t index(T item)
00160     {
00161         return std::distance(
00162             m_collection.begin(),
00163             std::find(m_collection.begin(), m_collection.end(), item));
00164     }
00165
00166     /**
00167      * \brief Check if the collection contains a specific item.
00168      *
00169      * \param item The item to search for.
00170      * \return bool True if the item is found, false otherwise.
00171      */
00172     virtual bool contains(T item)
00173     {
00174         return std::find(m_collection.begin(), m_collection.end(), item) !=
00175                m_collection.end();
00176     }
00177
00178     /**
00179      * \brief Create a copy of the collection.
00180      *
00181      * \return std::vector<T> A copy of the collection.
00182      */
00183     virtual std::vector<T> copy() { return m_collection; }
00184
00185     /**
00186      * \brief Get a slice of the collection from the specified start index to
00187      * the specified end index.
00188      *
00189      * \param start The start index of the slice.
00190      * \param end The end index of the slice.
00191      * \return std::vector<T> A slice of the collection.
00192      */
00193     virtual std::vector<T> slice(int start, int end)
00194     {
00195         return std::vector<T>(m_collection.begin() + start,
00196                               m_collection.begin() + end);
00197     }
00198
00199     /**
00200      * \brief Get a slice of the collection from the specified start index to
00201      * the end of the collection.
00202      *
00203      * \param start The start index of the slice.
00204      * \return std::vector<T> A slice of the collection.
00205      */
00206     virtual std::vector<T> slice(int start)
00207     {
00208         return std::vector<T>(m_collection.begin() + start, m_collection.end());
00209     }
00210
00211     /**
00212      * \brief Get a slice of the entire collection.
```

```
00213      *
00214      * \return std::vector<T> A slice of the collection.
00215      */
00216     virtual std::vector<T> slice() { return m_collection; }
00217
00218     /**
00219      * \brief Reverse the order of the items in the collection.
00220      */
00221     virtual void reverse()
00222     {
00223         std::reverse(m_collection.begin(), m_collection.end());
00224     }
00225
00226     /**
00227      * \brief Sort the items in the collection in ascending order.
00228      */
00229     virtual void sort() { std::sort(m_collection.begin(), m_collection.end()); }
00230
00231     /**
00232      * \brief Sort the items in the collection using a custom comparison
00233      * function.
00234      *
00235      * \param compare The comparison function to be used for sorting.
00236      */
00237     virtual void sort(std::function<bool(T, T)> compare)
00238     {
00239         std::sort(m_collection.begin(), m_collection.end(), compare);
00240     }
00241
00242   protected:
00243     std::vector<T> m_collection; /**< The collection of items. */
00244 };
00245
00246 #endif // COLLECTION_HPP
```

7.9 AEGP/Core/Base/Import.hpp File Reference

An Asset Importer for After Effects Abstracts importing of assets (and various configurations) into After Effects.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class ImportOptions

  Represents the options for importing assets into After Effects.
- struct ImportOptions::Config

  Represents the configuration options for importing assets.

7.9.1 Detailed Description

An Asset Importer for After Effects Abstracts importing of assets (and various configurations) into After Effects.

Import.hpp

Author

tjerf

Date

March 2024

7.10 Import.hpp

Go to the documentation of this file.

```
00001 /*********************************************************************/ /**
00002                                                                        * \file
00003                                                                        *Import.hpp
00004                                                                        * \brief An
00005                                                                        *Asset
00006                                                                        *Importer
00007                                                                        *for After
00008                                                                        *Effects
00009                                                                        * Abstracts
00010                                                                        *importing
00011                                                                        *of assets
00012                                                                        *(and
00013                                                                        *various
00014                                                                        *configurations)
00015                                                                        *into After
00016                                                                        *Effects
00017                                                                        *
00018                                                                        * \author
00019                                                                        *tjerf
00020                                                                        * \date
00021                                                                        *March 2024
00022
      **********************************************************************/
00023
00024 #pragma once
00025
00026 #ifndef IMPORT_HPP
00027 #define IMPORT_HPP
00028
00029 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00030
00031 /**
00032  * \class ImportOptions
00033  * \brief Represents the options for importing assets into After Effects.
00034  *
00035  * The ImportOptions class provides a set of configuration options for importing
00036  * assets into After Effects. It supports importing assets as still images,
00037  * image sequences, compositions, or individual layers.
00038  */
00039 class ImportOptions
00040 {
00041   public:
00042     /**
00043      * \brief Default constructor for ImportOptions.
00044      */
00045     ImportOptions();
00046
00047     /**
00048      * \brief Destructor for ImportOptions.
00049      */
00050     ~ImportOptions();
00051
00052     /**
00053      * \struct Config
00054      * \brief Represents the configuration options for importing assets.
00055      *
00056      * The Config struct defines additional configuration parameters for
00057      * importing assets. It includes options such as frame rate, width, height,
00058      * name, and duration.
00059      */
00060     struct Config
00061     {
00062         std::optional<double>
00063             frameRate;             ///< The frame rate of the imported assets.
00064         std::optional<int> width;  ///< The width of the imported assets.
00065         std::optional<int> height; ///< The height of the imported assets.
00066         std::optional<std::string> name; ///< The name of the imported assets.
00067         std::optional<double>
00068             duration; ///< The duration of the imported assets.
00069     };
00070
00071     /**
00072      * \brief Imports assets into After Effects with the specified configuration
00073      * options.
00074      *
00075      * The importAssets function imports assets into After Effects based on the
00076      * provided files and configuration options. It supports importing assets as
00077      * still images, image sequences, compositions, or individual layers.
00078      *
00079      * \param files The files to import. It can be a single file or a vector of
00080      * files. \param config The configuration options for importing the assets.
00081      * Defaults to an empty configuration. \return bool True if the assets were
```

```
00082      * imported successfully, false otherwise.
00083      */
00084     bool importAssets(
00085         const std::variant<std::string, std::vector<std::string>> &files,
00086         const Config &config = {});
00087
00088   private:
00089     /**
00090      * \brief Imports assets as individual files.
00091      *
00092      * The importAsFiles function imports assets as individual files into After
00093      * Effects.
00094      *
00095      * \param files The files to import.
00096      * \param config The configuration options for importing the assets.
00097      * \return bool True if the assets were imported successfully, false
00098      * otherwise.
00099      */
00100     bool importAsFiles(const std::vector<std::string> &files,
00101                        const Config &config);
00102
00103     /**
00104      * \brief Imports assets as image sequences.
00105      *
00106      * The importAsFrames function imports assets as image sequences into After
00107      * Effects.
00108      *
00109      * \param files The files to import.
00110      * \param config The configuration options for importing the assets.
00111      * \return bool True if the assets were imported successfully, false
00112      * otherwise.
00113      */
00114     bool importAsFrames(const std::vector<std::string> &files,
00115                         const Config &config);
00116
00117     /**
00118      * \brief Imports assets as compositions.
00119      *
00120      * The importAsComp function imports assets as compositions into After
00121      * Effects.
00122      *
00123      * \param files The files to import.
00124      * \param config The configuration options for importing the assets.
00125      * \return bool True if the assets were imported successfully, false
00126      * otherwise.
00127      */
00128     bool importAsComp(const std::vector<std::string> &files,
00129                       const Config &config);
00130
00131     /**
00132      * \brief Imports assets as individual layers.
00133      *
00134      * The importAsLayers function imports assets as individual layers into
00135      * After Effects.
00136      *
00137      * \param files The files to import.
00138      * \param config The configuration options for importing the assets.
00139      * \return bool True if the assets were imported successfully, false
00140      * otherwise.
00141      */
00142     bool importAsLayers(const std::vector<std::string> &files,
00143                         const Config &config);
00144
00145     /**
00146      * \brief Stores the last error message encountered during import.
00147      */
00148     std::string lastError;
00149
00150     /**
00151      * \brief Expands the files to import.
00152      *
00153      * The expandFiles function expands the files to import based on the
00154      * provided variant.
00155      *
00156      * \param files The files to expand.
00157      * \return std::vector<std::string> The expanded files.
00158      */
00159     static std::vector<std::string> expandFiles(
00160         const std::variant<std::string, std::vector<std::string>> &files);
00161
00162     /**
00163      * \brief Verifies the validity of the files to import.
00164      *
00165      * The verifyFiles function verifies the validity of the files to import.
00166      *
00167      * \param files The files to verify.
00168      * \return bool True if the files are valid, false otherwise.
```

```
00169      */
00170     static bool verifyFiles(const std::vector<std::string> &files);
00171 };
00172
00173 /**
00174  * \brief Example usage of the ImportOptions class.
00175  *
00176  * The ImportOptions class can be used to import assets into After Effects with
00177  * specific configuration options.
00178  *
00179  * \code{.cpp}
00180  * ImportOptions import;
00181  * ImportOptions::Config config;
00182  *
00183  * // Define configuration
00184  * config.frameRate = 24;
00185  * config.width = 1920;
00186  * config.height = 1080;
00187  * config.duration = 10;
00188  * config.name = "MyImportedComp";
00189  *
00190  * // Import as composition
00191  * import.importAssets("path/to/file.mov", config);
00192  * \endcode
00193  */
00194 #endif // IMPORT_HPP
```

7.11 AEGP/Core/Base/Item.hpp File Reference

After Effects Item class.

`#include "AETK/AEGP/Core/Base/AEGeneral.hpp"`

Classes

- class AE::Item

  Represents an After Effects item.

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.11.1 Detailed Description

After Effects Item class.

AEItem.hpp

Author

tjerf

Date

March 2024

7.12 Item.hpp

Go to the documentation of this file.

```
00001 /*********************************************************************/ /**
00002                                                                        * \file
00003                                                                        *AEItem.hpp
00004                                                                        * \brief
00005                                                                        *After
00006                                                                        *Effects
00007                                                                        *Item class
00008                                                                        *
00009                                                                        * \author
00010                                                                        *tjerf
00011                                                                        * \date
00012                                                                        *March 2024
00013 
    **********************************************************************/
00014 
00015 #ifndef AEITEM_HPP
00016 #define AEITEM_HPP
00017 
00018 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00019 
00020 namespace AE
00021 {
00022 
00023 /**
00024  * \class Item
00025  * \brief Represents an After Effects item.
00026  */
00027 class Item
00028 {
00029   public:
00030     /**
00031      * \brief Default constructor.
00032      */
00033     Item(){};
00034 
00035     /**
00036      * \brief Constructor that takes an existing item pointer.
00037      * \param item The item pointer.
00038      */
00039     Item(ItemPtr item) : m_item(item){};
00040 
00041     /**
00042      * \brief Destructor.
00043      */
00044     virtual ~Item(){};
00045 
00046     /**
00047      * \brief Returns the active item.
00048      * \return The active item.
00049      */
00050     static Item ActiveItem();
00051 
00052     /**
00053      * \brief Selects the item.
00054      * \param deselectOthers Flag indicating whether to deselect other items.
00055      */
00056     void Select(bool deselectOthers = true);
00057 
00058     /**
00059      * \brief Deselects the item.
00060      */
00061     void Deselect();
00062 
00063     /**
00064      * \brief Checks if the item is selected.
00065      * \return True if the item is selected, false otherwise.
00066      */
00067     bool IsSelected() const;
00068 
00069     /**
00070      * \brief Returns the name of the item.
00071      * \return The name of the item.
00072      */
00073     std::string Name() const;
00074 
00075     /**
00076      * \brief Returns the dimensions of the item.
00077      * \return A tuple containing the width and height of the item.
00078      */
00079     std::tuple<int, int> Dimensions() const;
00080 
00081     /**
```

```
00082      * \brief Returns the width of the item.
00083      * \return The width of the item.
00084      */
00085     int Width() const;
00086
00087     /**
00088      * \brief Returns the height of the item.
00089      * \return The height of the item.
00090      */
00091     int Height() const;
00092
00093     /**
00094      * \brief Deletes the item.
00095      */
00096     void Delete();
00097
00098     /**
00099      * \brief Checks if the item is missing.
00100      * \return True if the item is missing, false otherwise.
00101      */
00102     bool Missing() const;
00103
00104     /**
00105      * \brief Checks if the item has a proxy.
00106      * \return True if the item has a proxy, false otherwise.
00107      */
00108     bool HasProxy() const;
00109
00110     /**
00111      * \brief Checks if the item is using a proxy.
00112      * \return True if the item is using a proxy, false otherwise.
00113      */
00114     bool UsingProxy() const;
00115
00116     /**
00117      * \brief Checks if the item is missing a proxy.
00118      * \return True if the item is missing a proxy, false otherwise.
00119      */
00120     bool MissingProxy() const;
00121
00122     /**
00123      * \brief Checks if the item has video.
00124      * \return True if the item has video, false otherwise.
00125      */
00126     bool HasVideo() const;
00127
00128     /**
00129      * \brief Checks if the item has audio.
00130      * \return True if the item has audio, false otherwise.
00131      */
00132     bool HasAudio() const;
00133
00134     /**
00135      * \brief Checks if the item is a still image.
00136      * \return True if the item is a still image, false otherwise.
00137      */
00138     bool Still() const;
00139
00140     /**
00141      * \brief Checks if the item has active audio.
00142      * \return True if the item has active audio, false otherwise.
00143      */
00144     bool ActiveAudio() const;
00145
00146     /**
00147      * \brief Returns the duration of the item.
00148      * \return The duration of the item.
00149      */
00150     double Duration() const;
00151
00152     /**
00153      * \brief Returns the current time of the item.
00154      * \return The current time of the item.
00155      */
00156     double Time() const;
00157
00158   private:
00159     ItemPtr m_item;         ///< The item pointer.
00160     ItemSuite9 m_itemSuite; ///< The item suite.
00161 };
00162
00163 } // namespace AE
00164
00165 #endif /* AEITEM_HPP */
```

7.13 AEGP/Core/Base/Layer.hpp File Reference

Layer class for After Effects.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AE::Layer

  Represents a layer in After Effects.

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.13.1 Detailed Description

Layer class for After Effects.

AELayer.hpp

Author

tjerf

Date

March 2024

7.14 Layer.hpp

Go to the documentation of this file.

```
00001 /*************************************************************/ /**
00002                                                                * \file
00003                                                                *AELayer.hpp
00004                                                                * \brief
00005                                                                *Layer
00006                                                                *class for
00007                                                                *After
00008                                                                *Effects
00009                                                                *
00010                                                                * \author
00011                                                                *tjerf
00012                                                                * \date
00013                                                                *March 2024
00014
      *********************************************************************/
00015
00016 #ifndef AELAYER_HPP
00017 #define AELAYER_HPP
00018
00019 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00020 //#include <Item.hpp>
00021 //#include <Comp.hpp>
00022 namespace AE
```

```
00023 {
00024 class Item;
00025 class CompItem;
00026 /**
00027  * \class Layer
00028  * \brief Represents a layer in After Effects.
00029  *
00030  * This class provides functionality to manipulate and retrieve information
00031  * about a layer in After Effects.
00032  */
00033 class Layer
00034 {
00035   public:
00036     /**
00037      * \brief Default constructor for Layer class.
00038      */
00039     Layer(){};
00040
00041     /**
00042      * \brief Virtual destructor for Layer class.
00043      */
00044     virtual ~Layer() = default;
00045
00046     /**
00047      * \brief Constructor for Layer class that takes a LayerPtr as input.
00048      * \param layer A pointer to a Layer object.
00049      */
00050     Layer(LayerPtr layer) : m_layer(layer){};
00051
00052     /**
00053      * \brief Get the underlying LayerPtr object.
00054      * \return LayerPtr A pointer to the underlying Layer object.
00055      */
00056     LayerPtr getLayer() { return m_layer; }
00057
00058     /**
00059      * \brief Set the underlying LayerPtr object.
00060      * \param layer A pointer to a Layer object.
00061      */
00062     void setLayer(LayerPtr layer) { m_layer = layer; }
00063
00064     /**
00065      * \brief Get the active layer.
00066      * \return Layer The active layer.
00067      */
00068     static Layer activeLayer();
00069
00070     /**
00071      * \brief Get the index of the layer.
00072      * \return int The index of the layer.
00073      */
00074     int index() const;
00075
00076     /**
00077      * \brief Reorder the layer to a new index.
00078      * \param newIndex The new index for the layer.
00079      */
00080     void reOrder(int newIndex);
00081
00082     /**
00083      * \brief Get the source item of the layer.
00084      * \return std::shared_ptr<Item> A shared pointer to the source item of the
00085      * layer.
00086      */
00087     std::shared_ptr<Item> source() const;
00088
00089     /**
00090      * \brief Get the source ID of the layer.
00091      * \return int The source ID of the layer.
00092      */
00093     int sourceID() const;
00094
00095     /**
00096      * \brief Get the parent composition of the layer.
00097      * \return CompItem The parent composition of the layer.
00098      */
00099     CompItem parentComp() const;
00100
00101     /**
00102      * \brief Get the name of the layer.
00103      * \return std::string The name of the layer.
00104      */
00105     std::string name() const;
00106
00107     /**
00108      * \brief Get the name of the source of the layer.
00109      * \return std::string The name of the source of the layer.
```

```
00110      */
00111     std::string sourceName() const;
00112
00113     /**
00114      * \brief Get the quality of the layer.
00115      * \return AE_LayerQual The quality of the layer.
00116      */
00117     AE_LayerQual quality() const;
00118
00119     /**
00120      * \brief Set the quality of the layer.
00121      * \param quality The quality of the layer.
00122      */
00123     void setQuality(AE_LayerQual quality);
00124
00125     /**
00126      * \brief Check if video is active for the layer.
00127      * \return bool True if video is active, false otherwise.
00128      */
00129     bool videoActive() const;
00130
00131     /**
00132      * \brief Check if audio is active for the layer.
00133      * \return bool True if audio is active, false otherwise.
00134      */
00135     bool audioActive() const;
00136
00137     /**
00138      * \brief Check if effects are active for the layer.
00139      * \return bool True if effects are active, false otherwise.
00140      */
00141     bool effectsActive() const;
00142
00143     /**
00144      * \brief Check if motion blur is active for the layer.
00145      * \return bool True if motion blur is active, false otherwise.
00146      */
00147     bool motionBlur() const;
00148
00149     /**
00150      * \brief Check if frame blending is active for the layer.
00151      * \return bool True if frame blending is active, false otherwise.
00152      */
00153     bool frameBlending() const;
00154
00155     /**
00156      * \brief Check if the layer is locked.
00157      * \return bool True if the layer is locked, false otherwise.
00158      */
00159     bool locked() const;
00160
00161     /**
00162      * \brief Check if the layer is shy.
00163      * \return bool True if the layer is shy, false otherwise.
00164      */
00165     bool shy() const;
00166
00167     /**
00168      * \brief Check if the layer is collapsed.
00169      * \return bool True if the layer is collapsed, false otherwise.
00170      */
00171     bool collapsed() const;
00172
00173     /**
00174      * \brief Check if auto-orientation is active for the layer.
00175      * \return bool True if auto-orientation is active, false otherwise.
00176      */
00177     bool autoOrient() const;
00178
00179     /**
00180      * \brief Check if the layer is an adjustment layer.
00181      * \return bool True if the layer is an adjustment layer, false otherwise.
00182      */
00183     bool adjustmentLayer() const;
00184
00185     /**
00186      * \brief Check if time remapping is active for the layer.
00187      * \return bool True if time remapping is active, false otherwise.
00188      */
00189     bool timeRemap() const;
00190
00191     /**
00192      * \brief Check if the layer is in 3D mode.
00193      * \return bool True if the layer is in 3D mode, false otherwise.
00194      */
00195     bool is3D() const;
00196
```

```
00197     /**
00198      * \brief Check if the layer is set to look at the camera.
00199      * \return bool True if the layer is set to look at the camera, false
00200      * otherwise.
00201      */
00202     bool lookAtCamera() const;
00203
00204     /**
00205      * \brief Check if the layer is set to look at the point of interest.
00206      * \return bool True if the layer is set to look at the point of interest,
00207      * false otherwise.
00208      */
00209     bool lookAtPOI() const;
00210
00211     /**
00212      * \brief Check if the layer is soloed.
00213      * \return bool True if the layer is soloed, false otherwise.
00214      */
00215     bool solo() const;
00216
00217     /**
00218      * \brief Check if the layer's markers are locked.
00219      * \return bool True if the layer's markers are locked, false otherwise.
00220      */
00221     bool markersLocked() const;
00222
00223     /**
00224      * \brief Check if the layer is a null layer.
00225      * \return bool True if the layer is a null layer, false otherwise.
00226      */
00227     bool nullLayer() const;
00228
00229     /**
00230      * \brief Check if locked masks are hidden for the layer.
00231      * \return bool True if locked masks are hidden, false otherwise.
00232      */
00233     bool hideLockedMask() const;
00234
00235     /**
00236      * \brief Check if the layer is a guide layer.
00237      * \return bool True if the layer is a guide layer, false otherwise.
00238      */
00239     bool guideLayer() const;
00240
00241     /**
00242      * \brief Check if the layer is set to render separately.
00243      * \return bool True if the layer is set to render separately, false
00244      * otherwise.
00245      */
00246     bool renderSeparately() const;
00247
00248     /**
00249      * \brief Check if the layer is an environment layer.
00250      * \return bool True if the layer is an environment layer, false otherwise.
00251      */
00252     bool environmentLayer() const;
00253
00254     /**
00255      * \brief Set the video active state for the layer.
00256      * \param active The video active state.
00257      */
00258     void setVideoActive(bool active);
00259
00260     /**
00261      * \brief Set the audio active state for the layer.
00262      * \param active The audio active state.
00263      */
00264     void setAudioActive(bool active);
00265
00266     /**
00267      * \brief Set the effects active state for the layer.
00268      * \param active The effects active state.
00269      */
00270     void setEffectsActive(bool active);
00271
00272     /**
00273      * \brief Set the motion blur state for the layer.
00274      * \param active The motion blur state.
00275      */
00276     void setMotionBlur(bool active);
00277
00278     /**
00279      * \brief Set the frame blending state for the layer.
00280      * \param active The frame blending state.
00281      */
00282     void setFrameBlending(bool active);
00283
```

```
00284     /**
00285      * \brief Set the locked state for the layer.
00286      * \param active The locked state.
00287      */
00288     void setLocked(bool active);
00289
00290     /**
00291      * \brief Set the shy state for the layer.
00292      * \param active The shy state.
00293      */
00294     void setShy(bool active);
00295
00296     /**
00297      * \brief Set the collapsed state for the layer.
00298      * \param active The collapsed state.
00299      */
00300     void setCollapsed(bool active);
00301
00302     /**
00303      * \brief Set the auto-orientation state for the layer.
00304      * \param active The auto-orientation state.
00305      */
00306     void setAutoOrient(bool active);
00307
00308     /**
00309      * \brief Set the adjustment layer state for the layer.
00310      * \param active The adjustment layer state.
00311      */
00312     void setAdjustmentLayer(bool active);
00313
00314     /**
00315      * \brief Set the time remap state for the layer.
00316      * \param active The time remap state.
00317      */
00318     void setTimeRemap(bool active);
00319
00320     /**
00321      * \brief Set the 3D state for the layer.
00322      * \param active The 3D state.
00323      */
00324     void setIs3D(bool active);
00325
00326     /**
00327      * \brief Set the look at camera state for the layer.
00328      * \param active The look at camera state.
00329      */
00330     void setLookAtCamera(bool active);
00331
00332     /**
00333      * \brief Set the look at point of interest state for the layer.
00334      * \param active The look at point of interest state.
00335      */
00336     void setLookAtPOI(bool active);
00337
00338     /**
00339      * \brief Set the solo state for the layer.
00340      * \param active The solo state.
00341      */
00342     void setSolo(bool active);
00343
00344     /**
00345      * \brief Set the markers locked state for the layer.
00346      * \param active The markers locked state.
00347      */
00348     void setMarkersLocked(bool active);
00349
00350     /**
00351      * \brief Set the null layer state for the layer.
00352      * \param active The null layer state.
00353      */
00354     void setNullLayer(bool active);
00355
00356     /**
00357      * \brief Set the hide locked mask state for the layer.
00358      * \param active The hide locked mask state.
00359      */
00360     void setHideLockedMask(bool active);
00361
00362     /**
00363      * \brief Set the guide layer state for the layer.
00364      * \param active The guide layer state.
00365      */
00366     void setGuideLayer(bool active);
00367
00368     /**
00369      * \brief Set the render separately state for the layer.
00370      * \param active The render separately state.
```

```
00371      */
00372     void setRenderSeparately(bool active);
00373
00374     /**
00375      * \brief Set the environment layer state for the layer.
00376      * \param active The environment layer state.
00377      */
00378     void setEnvironmentLayer(bool active);
00379
00380     /**
00381      * \brief Check if video is on for the layer.
00382      * \return bool True if video is on, false otherwise.
00383      */
00384     bool isVideoOn() const;
00385
00386     /**
00387      * \brief Check if audio is on for the layer.
00388      * \return bool True if audio is on, false otherwise.
00389      */
00390     bool isAudioOn() const;
00391
00392     /**
00393      * \brief Get the time of the layer.
00394      * \return double The time of the layer.
00395      */
00396     double time() const;
00397
00398     /**
00399      * \brief Get the in point of the layer.
00400      * \return double The in point of the layer.
00401      */
00402     double inPoint() const;
00403
00404     /**
00405      * \brief Get the duration of the layer.
00406      * \return double The duration of the layer.
00407      */
00408     double duration() const;
00409
00410     /**
00411      * \brief Set the in point and duration of the layer.
00412      * \param inPoint The in point of the layer.
00413      * \param duration The duration of the layer.
00414      */
00415     void setInPointAndDuration(double inPoint, double duration);
00416
00417     /**
00418      * \brief Get the offset of the layer.
00419      * \return double The offset of the layer.
00420      */
00421     double offset() const;
00422
00423     /**
00424      * \brief Set the offset of the layer.
00425      * \param offset The offset of the layer.
00426      */
00427     void setOffset(double offset);
00428
00429     /**
00430      * \brief Get the stretch of the layer.
00431      * \return double The stretch of the layer.
00432      */
00433     double stretch() const;
00434
00435     /**
00436      * \brief Set the stretch of the layer.
00437      * \param stretch The stretch of the layer.
00438      */
00439     void setStretch(double stretch);
00440
00441     /**
00442      * \brief Delete the layer.
00443      */
00444     void Delete();
00445
00446     /**
00447      * \brief Duplicate the layer.
00448      * \return Layer A duplicate of the layer.
00449      */
00450     Layer duplicate();
00451
00452     /**
00453      * \brief Get the sampling quality of the layer.
00454      * \return AE_LayerSamplingQual The sampling quality of the layer.
00455      */
00456     AE_LayerSamplingQual samplingQuality() const;
00457
```

```
00458     /**
00459      * \brief Set the sampling quality of the layer.
00460      * \param quality The sampling quality of the layer.
00461      */
00462     void setSamplingQuality(AE_LayerSamplingQual quality);
00463
00464   private:
00465     LayerPtr m_layer;
00466 };
00467 } // namespace AE
00468
00469 #endif // AELAYER_HPP
```

7.15 AEGP/Core/Base/Properties.hpp File Reference

Property classes for After Effects properties.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AE::PropertyBase
- class AE::PropertyGroup
- class AE::OneDProperty
- class AE::TwoDProperty
- class AE::ThreeDProperty
- class AE::ColorProperty
- class AE::MarkerProperty
- class AE::LayerIDProperty
- class AE::MaskIDProperty
- class AE::MaskProperty
- class AE::TextDocumentProperty

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.15.1 Detailed Description

Property classes for After Effects properties.

Properties.hpp

Author

tjerf

Date

March 2024

7.16 Properties.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************/ /**
00002                                                                           * \file
00003                                                                           *Properties.hpp
00004                                                                           * \brief
00005                                                                           *Property
00006                                                                           *classes
00007                                                                           *for After
00008                                                                           *Effects
00009                                                                           *properties
00010                                                                           *
00011                                                                           * \author
00012                                                                           *tjerf
00013                                                                           * \date
00014                                                                           *March 2024
00015 
      **************************************************************************/
00016 #pragma once
00017 
00018 #ifndef PROPERTIES_HPP
00019 #define PROPERTIES_HPP
00020 
00021 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00022 
00023 namespace AE
00024 {
00025 // forward declare property classes
00026 class PropertyBase;
00027 class PropertyGroup;
00028 class OneDProperty;
00029 class TwoDProperty;
00030 class ThreeDProperty;
00031 class ColorProperty;
00032 class MarkerProperty;
00033 class LayerIDProperty;
00034 class MaskIDProperty;
00035 class MaskProperty;
00036 class TextDocumentProperty;
00037 
00038 class PropertyBase
00039 {
00040   protected:
00041     StreamRefPtr streamRef; // Reference to the AE property stream
00042     AE_StreamType valueType;
00043 
00044   public:
00045     PropertyBase(StreamRefPtr stream, AE_StreamType valType)
00046         : streamRef(stream), valueType(valType)
00047     {
00048     }
00049 
00050     virtual ~PropertyBase() = default;
00051 
00052     std::string getName() const;
00053     std::string getUnits() const;
00054     bool canAddProperty(const std::string &name) const;
00055     bool isLegal() const;
00056     bool isTimeVarying() const;
00057     bool isHidden() const;
00058     bool isElided() const;
00059     std::shared_ptr<PropertyGroup> getParentGroup() const;
00060 
00061     std::string getMatchName() const;
00062     void setName(const std::string &name);
00063     void reorder(int newIndex);
00064     void deleteProperty();
00065     // Accessor for the property's value type
00066     AE_StreamType getValueType() const { return valueType; }
00067 };
00068 
00069 class PropertyGroup : public PropertyBase
00070 {
00071   public:
00072     PropertyGroup(StreamRefPtr stream,
00073                   AE_StreamType valType = AE_StreamType::NONE)
00074         : PropertyBase(stream, valType)
00075     {
00076     }
00077 
00078     virtual ~PropertyGroup() = default;
00079 
00080     // Common functionality for all property groups
00081     virtual int getNumProperties() const = 0;
```

```
00082     virtual std::shared_ptr<PropertyBase> getProperty(int index) const = 0;
00083     virtual std::shared_ptr<PropertyBase>
00084     getProperty(const std::string &name) const = 0;
00085
00086     template <typename EnumType>
00087     std::shared_ptr<PropertyBase> getProperty(EnumType name) const;
00088
00089     virtual std::shared_ptr<PropertyBase>
00090     addProperty(const std::string &name) = 0;
00091
00092     template <typename EnumType>
00093     std::shared_ptr<PropertyBase> addProperty(EnumType name);
00094
00095     virtual void removeProperty(const std::string &name) = 0;
00096     virtual void removeProperty(int index) = 0;
00097
00098     template <typename EnumType> void removeProperty(EnumType name);
00099 };
00100
00101 class OneDProperty : public PropertyBase
00102 {
00103   public:
00104     OneDProperty(StreamRefPtr stream)
00105         : PropertyBase(stream, AE_StreamType::OneD)
00106     {
00107     }
00108
00109     virtual ~OneDProperty() = default;
00110
00111     double getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00112     void setValue(double value, AE_LTimeMode timeMode, double time = 0.0);
00113 };
00114
00115 class TwoDProperty : public PropertyBase
00116 {
00117   public:
00118     TwoDProperty(StreamRefPtr stream)
00119         : PropertyBase(stream, AE_StreamType::TwoD)
00120     {
00121     }
00122
00123     virtual ~TwoDProperty() = default;
00124
00125     AEGP_TwoDVal getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00126     void setValue(AEGP_TwoDVal value, AE_LTimeMode timeMode, double time = 0.0);
00127 };
00128
00129 class ThreeDProperty : public PropertyBase
00130 {
00131   public:
00132     ThreeDProperty(StreamRefPtr stream)
00133         : PropertyBase(stream, AE_StreamType::ThreeD)
00134     {
00135     }
00136
00137     virtual ~ThreeDProperty() = default;
00138
00139     AEGP_ThreeDVal getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00140     void setValue(AEGP_ThreeDVal value, AE_LTimeMode timeMode,
00141                   double time = 0.0);
00142 };
00143
00144 class ColorProperty : public PropertyBase
00145 {
00146   public:
00147     ColorProperty(StreamRefPtr stream)
00148         : PropertyBase(stream, AE_StreamType::COLOR)
00149     {
00150     }
00151
00152     virtual ~ColorProperty() = default;
00153
00154     AEGP_ColorVal getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00155     void setValue(AEGP_ColorVal value, AE_LTimeMode timeMode,
00156                   double time = 0.0);
00157 };
00158
00159 class MarkerProperty : public PropertyBase
00160 {
00161   public:
00162     MarkerProperty(StreamRefPtr stream)
00163         : PropertyBase(stream, AE_StreamType::MARKER)
00164     {
00165     }
00166
00167     virtual ~MarkerProperty() = default;
00168
```

```
00169     MarkerValPtr getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00170     void setValue(MarkerValPtr value, AE_LTimeMode timeMode, double time = 0.0);
00171 };
00172
00173 class LayerIDProperty : public PropertyBase
00174 {
00175   public:
00176     LayerIDProperty(StreamRefPtr stream)
00177         : PropertyBase(stream, AE_StreamType::LAYER_ID)
00178     {
00179     }
00180
00181     virtual ~LayerIDProperty() = default;
00182
00183     A_long getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00184     void setValue(A_long value, AE_LTimeMode timeMode, double time = 0.0);
00185 };
00186
00187 class MaskIDProperty : public PropertyBase
00188 {
00189   public:
00190     MaskIDProperty(StreamRefPtr stream)
00191         : PropertyBase(stream, AE_StreamType::MASK_ID)
00192     {
00193     }
00194
00195     virtual ~MaskIDProperty() = default;
00196
00197     A_long getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00198     void setValue(A_long value, AE_LTimeMode timeMode, double time = 0.0);
00199 };
00200
00201 class MaskProperty : public PropertyBase
00202 {
00203   public:
00204     MaskProperty(StreamRefPtr stream)
00205         : PropertyBase(stream, AE_StreamType::MASK)
00206     {
00207     }
00208
00209     virtual ~MaskProperty() = default;
00210
00211     MaskOutlineValPtr getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00212     void setValue(MaskOutlineValPtr value, AE_LTimeMode timeMode,
00213                   double time = 0.0);
00214 };
00215
00216 class TextDocumentProperty : public PropertyBase
00217 {
00218   public:
00219     TextDocumentProperty(StreamRefPtr stream)
00220         : PropertyBase(stream, AE_StreamType::TEXT_DOCUMENT)
00221     {
00222     }
00223
00224     virtual ~TextDocumentProperty() = default;
00225
00226     TextDocumentPtr getValue(AE_LTimeMode timeMode, double time = 0.0) const;
00227     void setValue(TextDocumentPtr value, AE_LTimeMode timeMode,
00228                   double time = 0.0);
00229 };
00230
00231 } // namespace AE
00232
00233 #endif // PROPERTIES_HPP
```

7.17 AEGP/Core/Base/SuiteManager.hpp File Reference

Singleton class managing the After Effects suite handler and plugin ID.

```
#include "Util/AEGP_SuiteHandler.h"
#include "Headers/AE_Macros.h"
```

Classes

- class SuiteManager

  Singleton class managing the After Effects suite handler and plugin ID.

7.17.1 Detailed Description

Singleton class managing the After Effects suite handler and plugin ID.

SuiteManager.hpp

Author

tjerf

Date

March 2024

7.18 SuiteManager.hpp

Go to the documentation of this file.

```
00001 /**
00002  * \file
00003  *SuiteManager.hpp
00004  * \brief
00005  *Singleton
00006  *class
00007  *managing
00008  *the After
00009  *Effects
00010  *suite
00011  *handler
00012  *and plugin
00013  *ID.
00014  *
00015  * \author
00016  *tjerf
00017  * \date
00018  *March 2024
00019  **************************************************************************/
00020
00021 #pragma once
00022
00023 #include "Util/AEGP_SuiteHandler.h"
00024 #include "Headers/AE_Macros.h"
00025
00026 /*
00027  * File: SuiteManager.h
00028  * Description: Singleton class managing the After Effects suite handler and
00029  * plugin ID.
00030  *
00031  * Guidelines for Contributors:
00032  * 1. Singleton Pattern: Recognize that SuiteManager is a singleton and should
00033  * not be instantiated directly.
00034  * 2. Suite Handling: Understand how SuiteManager provides access to AE suites.
00035  * 3. No Alteration: Do not modify this file. It is crucial for the stable
00036  * operation of the entire plugin.
00037  */
00038
00039 /**
00040  * @class SuiteManager
00041  * @brief Singleton class managing the After Effects suite handler and plugin
00042  * ID.
00043  *
00044  * The SuiteManager class is responsible for managing the After Effects suite
00045  * handler and plugin ID. It follows the Singleton pattern to ensure that only
00046  * one instance of the class can exist. The class provides methods to initialize
00047  * the suite handler, get the suite handler, set the plugin ID, and get the
00048  * plugin ID.
00049  */
00050 class SuiteManager
00051 {
00052   public:
00053     /**
00054      * @brief Gets the singleton instance of SuiteManager.
00055      *
00056      * This method returns the singleton instance of the SuiteManager class.
```

```
00057      *
00058      * @return SuiteManager& The reference to the singleton instance of
00059      * SuiteManager.
00060      */
00061     static SuiteManager &GetInstance()
00062     {
00063         static SuiteManager instance;
00064         return instance;
00065     }
00066
00067     // Deleted copy constructor and assignment operator to ensure singleton
00068     SuiteManager(SuiteManager const &) = delete;
00069     void operator=(SuiteManager const &) = delete;
00070
00071     /**
00072      * @brief Initializes the suite handler.
00073      *
00074      * This method initializes the suite handler with the provided SPBasicSuite
00075      * pointer. It should be called before accessing any AE suites.
00076      *
00077      * @param pica_basicP The SPBasicSuite pointer.
00078      */
00079     void InitializeSuiteHandler(SPBasicSuite *pica_basicP)
00080     {
00081         if (!suitesInitialized)
00082         {
00083             suites = new AEGP_SuiteHandler(pica_basicP);
00084             suitesInitialized = true;
00085         }
00086     }
00087
00088     /**
00089      * @brief Gets the suite handler.
00090      *
00091      * This method returns a reference to the suite handler.
00092      *
00093      * @return AEGP_SuiteHandler& The reference to the suite handler.
00094      */
00095     AEGP_SuiteHandler &GetSuiteHandler() { return *suites; }
00096
00097     /**
00098      * @brief Sets the plugin ID.
00099      *
00100      * This method sets the plugin ID with the provided AEGP_PluginID pointer.
00101      *
00102      * @param pluginIDPtr The AEGP_PluginID pointer.
00103      */
00104     void SetPluginID(AEGP_PluginID *pluginIDPtr)
00105     {
00106         this->pluginIDPtr = pluginIDPtr;
00107     }
00108
00109     /**
00110      * @brief Gets the plugin ID.
00111      *
00112      * This method returns a constant pointer to the plugin ID.
00113      *
00114      * @return const AEGP_PluginID* The constant pointer to the plugin ID.
00115      */
00116     AEGP_PluginID *GetPluginID() const { return pluginIDPtr; }
00117
00118   private:
00119     /**
00120      * @brief Default constructor.
00121      *
00122      * The default constructor is private to prevent direct instantiation of the
00123      * SuiteManager class.
00124      */
00125     SuiteManager()
00126         : suites(nullptr), suitesInitialized(false), pluginIDPtr(nullptr)
00127     {
00128     }
00129
00130     AEGP_SuiteHandler *suites; /**< Pointer to the suite handler. */
00131     bool suitesInitialized; /**< Flag indicating if the suite handler has been
00132                                initialized. */
00133     AEGP_PluginID *pluginIDPtr; /**< Pointer to the plugin ID. */
00134 };
```

7.19 AEGP/Core/Comp.hpp File Reference

A header file for CompItem class.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
#include "AETK/AEGP/Core/Base/Item.hpp"
```

Classes

- class AE::CompItem

  A class for representing a composition in After Effects.

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.19.1 Detailed Description

A header file for CompItem class.

AEComp.hpp

Author

tjerf

Date

March 2024

7.20 Comp.hpp

Go to the documentation of this file.

```
00001 /*************************************************************/ /**
00002                                                                * \file
00003                                                                *AEComp.hpp
00004                                                                * \brief  A
00005                                                                *header
00006                                                                *file for
00007                                                                *CompItem
00008                                                                *class
00009                                                                *
00010                                                                * \author
00011                                                                *tjerf
00012                                                                * \date
00013                                                                *March 2024
00014
      ********************************************************************/
00015
00016 #pragma once
00017
00018 #ifndef AEComp_hpp
00019 #define AEComp_hpp
00020
00021 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00022 #include "AETK/AEGP/Core/Base/Item.hpp"
00023 //in cpp, #include <Layer.hpp>
00024
00025 namespace AE
00026 {
00027 class Layer; // Forward declaration of Layer class
00028 /**
```

```
00029  * \brief A class for representing a composition in After Effects
00030  *
00031  * This class represents a composition in After Effects. It inherits from the
00032  * base class Item.
00033  */
00034 class CompItem : public Item
00035 {
00036   public:
00037     /**
00038      * \brief Default constructor for CompItem
00039      */
00040     CompItem();
00041
00042     /**
00043      * \brief Constructor for CompItem that takes an ItemPtr
00044      *
00045      * \param item A pointer to an Item object
00046      */
00047     CompItem(ItemPtr item) : m_item(item) { m_comp = compFromItem(); }
00048
00049     /**
00050      * \brief Constructor for CompItem that takes a CompPtr
00051      *
00052      * \param comp A pointer to a Comp object
00053      */
00054     CompItem(CompPtr comp) : m_comp(comp) { m_item = itemFromComp(); }
00055
00056     /**
00057      * \brief Copy constructor for CompItem
00058      *
00059      * \param comp A reference to a CompItem object to be copied
00060      */
00061     CompItem(CompItem const &comp) : m_item(comp.m_item)
00062     {
00063         m_comp = comp.m_comp;
00064     }
00065
00066     /**
00067      * \brief Get the layers of the composition
00068      *
00069      * \return std::vector<Layer> A vector of Layer objects representing the
00070      * layers in the composition
00071      */
00072     std::vector<Layer> layers() const;
00073
00074     /**
00075      * \brief Get the layer at the given index
00076      *
00077      * \param index The index of the layer to retrieve
00078      * \return std::shared_ptr<Layer> A shared pointer to the Layer object at
00079      * the given index
00080      */
00081     std::shared_ptr<Layer> layer(int index) const;
00082
00083     /**
00084      * \brief Get the layer with the given name
00085      *
00086      * \param name The name of the layer to retrieve
00087      * \return std::shared_ptr<Layer> A shared pointer to the Layer object with
00088      * the given name
00089      */
00090     std::shared_ptr<Layer> layer(std::string name) const;
00091
00092     /**
00093      * \brief Add a layer to the composition
00094      *
00095      * \param itemToAdd A shared pointer to the Item object to be added as a
00096      * layer \return std::shared_ptr<Layer> A shared pointer to the newly added
00097      * Layer object
00098      */
00099     std::shared_ptr<Layer> addLayer(std::shared_ptr<Item> itemToAdd);
00100
00101     /**
00102      * \brief Remove a layer from the composition
00103      *
00104      * \param itemToRemove A shared pointer to the Layer object to be removed
00105      */
00106     void removeLayer(std::shared_ptr<Layer> itemToRemove);
00107
00108     /**
00109      * \brief Remove a layer from the composition based on its index
00110      *
00111      * \param index The index of the layer to be removed
00112      */
00113     void removeLayer(int index);
00114
00115     /**
```

```
00116      * \brief Remove a layer from the composition based on its name
00117      *
00118      * \param name The name of the layer to be removed
00119      */
00120     void removeLayer(std::string name);
00121
00122     /**
00123      * \brief Show or hide layer names in the composition
00124      *
00125      * \param show A boolean value indicating whether to show or hide layer
00126      * names
00127      */
00128     void showLayerNames(bool show);
00129
00130   private:
00131     CompPtr m_comp; // Pointer to a Comp object
00132     ItemPtr m_item; // Pointer to an Item object
00133
00134     /**
00135      * \brief Get a CompPtr from an ItemPtr
00136      *
00137      * \return CompPtr A pointer to a Comp object
00138      */
00139     inline CompPtr compFromItem()
00140     {
00141         return CompSuite11().GetCompFromItem(m_item);
00142     }
00143
00144     /**
00145      * \brief Get an ItemPtr from a CompPtr
00146      *
00147      * \return ItemPtr A pointer to an Item object
00148      */
00149     inline ItemPtr itemFromComp()
00150     {
00151         return CompSuite11().GetItemFromComp(m_comp);
00152     }
00153 };
00154 } // namespace AE
00155 #endif /* AEComp_hpp */
```

7.21 AEGP/Core/Effects.hpp File Reference

A header file for the Effects class.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.21.1 Detailed Description

A header file for the Effects class.

Author

tjerf

Date

March 2024

7.22 Effects.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************//**
00002  * \file   Effects.hpp
00003  * \brief  A header file for the Effects class
00004  *
00005  * \author tjerf
00006  * \date   March 2024
00007  *****************************************************************************/
00008
00009 #ifndef EFFECTS_HPP
00010 #define EFFECTS_HPP
00011
00012 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00013
00014 namespace AE
00015 {
00016
00017 } // namespace AE
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027 #endif //EFFECTS_HPP
```

7.23 AEGP/Core/Project.hpp File Reference

A class representing an After Effects Project.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AE::Project

  A class representing an After Effects Project.

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.23.1 Detailed Description

A class representing an After Effects Project.

Project.hpp

Author

tjerf

Date

March 2024

7.24 Project.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************/ /**
00002                                                                           * \file
00003                                                                           *Project.hpp
00004                                                                           * \brief  A
00005                                                                           *class
00006                                                                           *representing
00007                                                                           *an After
00008                                                                           *Effects
00009                                                                           *Project
00010                                                                           *
00011                                                                           *
00012                                                                           * \author
00013                                                                           *tjerf
00014                                                                           * \date
00015                                                                           *March 2024
00016 
      **************************************************************************/
00017 
00018 #ifndef PROJECT_HPP
00019 #define PROJECT_HPP
00020 
00021 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00022 
00023 namespace AE
00024 {
00025 
00026 /**
00027  * @brief A class representing an After Effects Project
00028  *
00029  * This class represents an After Effects project and provides methods to
00030  * interact with it.
00031  */
00032 class Project
00033 {
00034   public:
00035     /**
00036      * @brief Default constructor
00037      *
00038      * Initializes a new Project object by calling the init() function.
00039      */
00040     Project() : m_proj(init()) {}
00041 
00042     /**
00043      * @brief Destructor
00044      *
00045      * Default destructor for the Project class.
00046      */
00047     ~Project() = default;
00048 
00049     /**
00050      * @brief Open an After Effects Project
00051      *
00052      * Opens an After Effects project from the specified path.
00053      *
00054      * @param path The path to the project file.
00055      * @return Project The opened project.
00056      */
00057     static Project open(const std::string &path);
00058 
00059     /**
00060      * @brief Create a new After Effects Project
00061      *
00062      * Creates a new After Effects project and saves it to the specified path.
00063      * If no path is provided, the project will be saved to the default
00064      * location.
00065      *
00066      * @param path The path to save the project to.
00067      * @return Project The newly created project.
00068      */
00069     static Project newProject(const std::string &path = "");
00070 
00071     /**
00072      * @brief Get the name of the project
00073      *
00074      * Retrieves the name of the current project.
00075      *
00076      * @return std::string The name of the project.
00077      */
00078     std::string name();
00079 
00080     /**
00081      * @brief Get the path of the project
```

```
00082      *
00083      * Retrieves the path of the current project.
00084      *
00085      * @return std::string The path of the project.
00086      */
00087     std::string path();
00088
00089     /**
00090      * @brief Get the bit depth of the project
00091      *
00092      * Retrieves the bit depth of the current project.
00093      *
00094      * @return AE_ProjBitDepth The bit depth of the project.
00095      */
00096     AE_ProjBitDepth bitDepth();
00097
00098     /**
00099      * @brief Set the bit depth of the project
00100      *
00101      * Sets the bit depth of the current project.
00102      *
00103      * @param depth The bit depth to set.
00104      */
00105     void setBitDepth(AE_ProjBitDepth depth);
00106
00107     /**
00108      * @brief Save the project
00109      *
00110      * Saves the current project.
00111      */
00112     void save();
00113
00114     /**
00115      * @brief Save the project to a new path
00116      *
00117      * Saves the current project to the specified path.
00118      *
00119      * @param path The path to save the project to.
00120      */
00121     void saveAs(const std::string &path);
00122
00123     /**
00124      * @brief Check if the project is dirty
00125      *
00126      * Checks if the current project has unsaved changes.
00127      *
00128      * @return bool True if the project is dirty, false otherwise.
00129      */
00130     bool isDirty();
00131
00132   private:
00133     ProjectPtr m_proj; // Pointer to the After Effects project
00134     ProjSuite6
00135         m_suite; // Suite for accessing After Effects project functionality
00136
00137     /**
00138      * @brief Initialize the project
00139      *
00140      * Initializes the project by retrieving the active project from the suite.
00141      * If no project is active, an exception is thrown.
00142      *
00143      * @return ProjectPtr The initialized project pointer.
00144      * @throws AEException If no project is currently active.
00145      */
00146     inline ProjectPtr init()
00147     {
00148         try
00149         {
00150             ProjectPtr proj = m_suite.GetProjectByIndex(0);
00151             return proj;
00152         }
00153         catch (const AEException &e)
00154         {
00155             throw AEException(e);
00156         }
00157     }
00158 };
00159
00160 } // namespace AE
00161
00162 #endif // PROJECT_HPP
```

7.25 AEGP/Exception/Exception.hpp File Reference

A custom exception class derived from std::exception, for managing AE exceptions. Also includes a utility function for null checking.

```
#include <exception>
#include <string>
```

Classes

- class AEException

  A custom exception class derived from std::exception.

Functions

- template<typename T >
  void CheckNotNull (const T *ptr, const char *errorMessage)

  Utility function for null checking.

7.25.1 Detailed Description

A custom exception class derived from std::exception, for managing AE exceptions. Also includes a utility function for null checking.

Author

tjerf

Date

March 2024

7.25.2 Function Documentation

7.25.2.1 CheckNotNull()

```
template<typename T >
void CheckNotNull (
            const T * ptr,
            const char * errorMessage )
```

Utility function for null checking.

This function checks if a pointer is null and throws an AEException if it is.

Template Parameters

T	The type of the pointer.

Parameters

ptr	The pointer to check.
errorMessage	The error message to be associated with the exception.

7.26 Exception.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************//**
00002  * \file   Exception.hpp
00003  * \brief  A custom exception class derived from std::exception, for managing AE exceptions.
00004  * Also includes a utility function for null checking.
00005  *
00006  * \author tjerf
00007  * \date   March 2024
00008  *****************************************************************************/
00009
00010 #ifndef EXCEPTION_HPP
00011 #define EXCEPTION_HPP
00012
00013 #include <exception>
00014 #include <string>
00015
00016 /**
00017  * @class AEException
00018  * @brief A custom exception class derived from std::exception.
00019  *
00020  * This class represents an exception that can be thrown in the project. It
00021  * provides a way to encapsulate and propagate error messages.
00022  */
00023 class AEException : public std::exception
00024 {
00025   public:
00026     /**
00027      * @brief Constructs an AEException object with the given error message.
00028      *
00029      * @param message The error message associated with the exception.
00030      */
00031     AEException(const std::string &message) : m_message(message) {}
00032
00033     /**
00034      * @brief Constructs an AEException object with the given error message.
00035      *
00036      * @param message The error message associated with the exception.
00037      */
00038     AEException(const char *message) : m_message(message) {}
00039
00040     /**
00041      * @brief Returns the error message associated with the exception.
00042      *
00043      * @return const char* The error message.
00044      */
00045     virtual const char *what() const throw() { return m_message.c_str(); }
00046
00047   private:
00048     std::string m_message;
00049 };
00050
00051 /**
00052  * @brief Utility function for null checking.
00053  *
00054  * This function checks if a pointer is null and throws an AEException if it is.
00055  *
00056  * @tparam T The type of the pointer.
00057  * @param ptr The pointer to check.
00058  * @param errorMessage The error message to be associated with the exception.
00059  */
00060 template <typename T> void CheckNotNull(const T *ptr, const char *errorMessage)
00061 {
```

```
00062     if (ptr == nullptr)
00063     {
00064         throw AEException(errorMessage);
00065     }
00066 }
00067
00068 #endif // EXCEPTION_HPP
```

7.27 AEGP/Memory/AEAllocator.hpp File Reference

AEAllocator is a custom allocator that uses the After Effects' Memory Suite to allocate and deallocate memory in standard library containers.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AEAllocator< T >

  AEAllocator is a custom allocator that uses the After Effects' memory suites to allocate and deallocate memory.

Functions

- template<typename T , typename U >
  bool **operator==** (const AEAllocator< T > &, const AEAllocator< U > &)
- template<typename T , typename U >
  bool **operator!=** (const AEAllocator< T > &, const AEAllocator< U > &)

7.27.1 Detailed Description

AEAllocator is a custom allocator that uses the After Effects' Memory Suite to allocate and deallocate memory in standard library containers.

Author

tjerf

Date

March 2024

7.28 AEAllocator.hpp

Go to the documentation of this file.

```
00001 /*****************************************************************//**
00002  * \file   AEAllocator.hpp
00003  * \brief  AEAllocator is a custom allocator that uses the After Effects'
00004  * Memory Suite to allocate and deallocate memory in standard library containers.
00005  *
00006  * \author tjerf
00007  * \date   March 2024
00008  *********************************************************************/
00009 #ifndef AEMEMORY_H
00010 #define AEMEMORY_H
00011
00012 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00013
00014
00015 /**
00016  * \brief AEAllocator is a custom allocator that uses the After Effects' memory
00017  * suites to allocate and deallocate memory.
00018  *
00019  * Doing so allows the user to use the standard library containers with the
00020  * custom allocator, and the memory will be allocated and deallocated using AE's
00021  * memory, potentially improving performance and reducing memory fragmentation.
00022  *
00023  * \param T The type of the elements to be allocated.
00024  * \Usage The AEAllocator can be used with the standard library containers, such
00025  * as std::vector, std::list, and std::map. \Example The following example
00026  * demonstrates how to use the AEAllocator with a std::vector. #include
00027  * <AEMemory.h>
00028  *
00029  *     template <typename T>
00030  *     using vector = std::vector<T, AEAllocator<T>>;
00031  */
00032 template <typename T>
00033 class AEAllocator {
00034  public:
00035   // Define type aliases for the allocator traits.
00036   using value_type = T;
00037   using pointer = T*;
00038   using size_type = size_t;
00039
00040   // Define a default constructor for the allocator.
00041   AEAllocator() noexcept {}
00042
00043   // Define a copy constructor for the allocator.
00044   template <typename U>
00045   AEAllocator(const AEAllocator<U>&) noexcept {}
00046
00047   // Define the allocate function, which is used to allocate memory.
00048   pointer allocate(size_type n) {
00049     // Calculate the size of the memory to be allocated.
00050     size_t size = n * sizeof(T);
00051
00052     // Explicitly handle the conversion, adding a check to ensure no data loss.
00053     if (size > (std::numeric_limits<AEGP_MemSize>::max)()) {
00054       throw std::bad_alloc();  // Or any other mechanism to signal the error
00055     }
00056
00057     // Use static_cast to convert size to AEGP_MemSize now that we've checked
00058     // for overflow.
00059     AEGP_MemSize safeSize = static_cast<AEGP_MemSize>(size);
00060
00061     MemHandlePtr memHandle =
00062         MemorySuite1().NewMemHandle("AEAllocator", safeSize, AE_MemFlag::NONE);
00063
00064     if (!memHandle) {
00065       throw AEException("Failed to allocate memory handle.");
00066     }
00067
00068     void* ptr = nullptr;
00069     // Inside AEGPCustomAllocator::allocate
00070     try {
00071       ptr = MemorySuite1().LockMemHandle(memHandle);  // Assume this returns the
00072                                                       // locked pointer directly
00073     } catch (...) {
00074       MemorySuite1().FreeMemHandle(memHandle);
00075       throw;  // Re-throw the current exception
00076               // without having to catch a specific
00077               // one
00078     }
00079
00080     memHandleMap_[static_cast<pointer>(ptr)] = memHandle;
00081     return static_cast<pointer>(ptr);
00082   }
```

```
00083
00084   // Define the deallocate function, which is used
00085   // to deallocate memory.
00086   void deallocate(pointer p, size_type n) {
00087     // Find the memory handle associated with the
00088     // pointer in the memHandleMap.
00089     auto it = memHandleMap_.find(p);
00090
00091     // Check if the memory handle is found.
00092     if (it != memHandleMap_.end()) {
00093       // Get an instance of the Suites
00094       MemorySuite1().UnlockMemHandle(it->second);
00095       MemorySuite1().FreeMemHandle(it->second);
00096
00097       // Remove the memory handle from the
00098       // memHandleMap.
00099       memHandleMap_.erase(it);
00100     }
00101   }
00102
00103  private:
00104   std::unordered_map<pointer, MemHandlePtr>
00105       memHandleMap_;  // Store the memory handles
00106                       // in an unordered_map.
00107 };
00108
00109 // Define the equality operator for two AEGPCustomAllocator objects.
00110 template <typename T, typename U>
00111 bool operator==(const AEAllocator<T>&, const AEAllocator<U>&) {
00112   return true;
00113 };
00114
00115 // Define the inequality operator for two AEGPCustomAllocator objects.
00116 template <typename T, typename U>
00117 bool operator!=(const AEAllocator<T>&, const AEAllocator<U>&) {
00118   return false;
00119 };
00120
00121 #endif  // AEMEMORY_H
```

7.29 AEGP/Memory/AEMemory.hpp File Reference

A header file that defines various STL containers using a custom Allocator for After Effects Memory Management.

`#include "AETK/AEGP/Memory/AEAllocator.hpp"`

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

Typedefs

- template<typename T >
  using AE::vector = std::vector<T, AEAllocator<T>>

  Alias for std::vector using AEAllocator.
- template<typename Key , typename Value , typename Comparator = std::less<Key>>
  using AE::map

  Alias for std::map using AEAllocator.
- template<typename T >
  using AE::list = std::list<T, AEAllocator<T>>

  Alias for std::list using AEAllocator.
- template<typename Key , typename Hash = std::hash<Key>, typename KeyEqual = std::equal_to<Key>>
  using AE::unordered_map

  Alias for std::unordered_map using AEAllocator.

Functions

- template<typename T , typename... Args>
  std::unique_ptr< T > **AE::make_unique** (Args &&...args)
- template<typename T , typename... Args>
  std::shared_ptr< T > AE::make_shared (Args &&...args)

  Custom make_shared function that uses AEAllocator to create the object.

7.29.1 Detailed Description

A header file that defines various STL containers using a custom Allocator for After Effects Memory Management.

Author

tjerf

Date

March 2024

7.30 AEMemory.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************//**
00002  * \file   AEMemory.hpp
00003  * \brief  A header file that defines various STL containers using a custom
00004  * Allocator for After Effects Memory Management.
00005  *
00006  * \author tjerf
00007  * \date   March 2024
00008  *****************************************************************************/
00009
00010 #ifndef AEMEMORY_HPP
00011 #define AEMEMORY_HPP
00012
00013 #include "AETK/AEGP/Memory/AEAllocator.hpp"
00014
00015 /**
00016  * \namespace AE
00017  * \brief Namespace for various pre-defined STL containers using a custom
00018  * allocator.
00019  */
00020 namespace AE
00021 {
00022 template <typename T, typename... Args>
00023 std::unique_ptr<T> make_unique(Args &&...args)
00024 {
00025     // Assuming AEAllocator<T> can be default constructed or otherwise obtained
00026     AEAllocator<T> alloc;
00027
00028     // Using the rebind mechanism to get the correct allocator type
00029     using alloc_traits = std::allocator_traits<AEAllocator<T»;
00030     using rebind_alloc = typename alloc_traits::template rebind_alloc<T>;
00031     rebind_alloc rebound_alloc(alloc);
00032
00033     // Allocate and construct the object, returning a std::unique_ptr to manage
00034     // it
00035     return std::allocate_unique<T>(rebound_alloc, std::forward<Args>(args)...);
00036 }
00037 /**
00038  * \brief Custom make_shared function that uses AEAllocator to create the
00039  * object. \tparam T The type of the object to be created. \tparam Args The
00040  * types of the arguments to be passed to the constructor of the object. \param
00041  * args The arguments to be passed to the constructor of the object. \return
00042  * std::shared_ptr<T> A shared pointer to the created object.
00043  *
00044  * This function creates an object of type T using the AEAllocator<T> allocator
00045  * and returns a std::shared_ptr<T> to manage the object.
```

```
00046  */
00047 template <typename T, typename... Args>
00048 std::shared_ptr<T> make_shared(Args &&...args)
00049 {
00050     // Assuming AEAllocator<T> can be default constructed or otherwise obtained
00051     AEAllocator<T> alloc;
00052
00053     // Using the rebind mechanism to get the correct allocator type
00054     using alloc_traits = std::allocator_traits<AEAllocator<T»;
00055     using rebind_alloc = typename alloc_traits::template rebind_alloc<T>;
00056     rebind_alloc rebound_alloc(alloc);
00057
00058     // Allocate and construct the object, returning a std::shared_ptr to manage
00059     // it
00060     return std::allocate_shared<T>(rebound_alloc, std::forward<Args>(args)...);
00061 }
00062
00063 /**
00064  * \brief Alias for std::vector using AEAllocator.
00065  * \tparam T The type of the elements in the vector.
00066  */
00067 template <typename T> using vector = std::vector<T, AEAllocator<T»;
00068
00069 /**
00070  * \brief Alias for std::map using AEAllocator.
00071  * \tparam Key The type of the keys in the map.
00072  * \tparam Value The type of the values in the map.
00073  * \tparam Comparator The type of the comparator used to compare keys.
00074  */
00075 template <typename Key, typename Value, typename Comparator = std::less<Key»
00076 using map =
00077     std::map<Key, Value, Comparator, AEAllocator<std::pair<const Key, Value»>;
00078
00079 /**
00080  * \brief Alias for std::list using AEAllocator.
00081  * \tparam T The type of the elements in the list.
00082  */
00083 template <typename T> using list = std::list<T, AEAllocator<T»;
00084
00085 /**
00086  * \brief Alias for std::unordered_map using AEAllocator.
00087  * \tparam Key The type of the keys in the unordered map.
00088  * \tparam Hash The type of the hash function used to hash keys.
00089  * \tparam KeyEqual The type of the key equality function used to compare keys
00090  * for equality.
00091  */
00092 template <typename Key, typename Hash = std::hash<Key>,
00093           typename KeyEqual = std::equal_to<Key»
00094 using unordered_map =
00095     std::unordered_map<Key, AEAllocator<std::pair<const Key, Hash»>;
00096
00097 } // namespace AE
00098
00099 #endif // AEMEMORY_HPP
```

7.31 AEGP/Template/Plugin.hpp File Reference

Plugin interface template for Adobe After Effects plugin development.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class Command

  Abstract base class for creating commands within the plugin.
- class Plugin

  Represents the plugin, managing its commands and lifecycle.

Macros

- #define DECLARE_ENTRY(PluginType)

  Macro for defining the plugin's entry point function.

7.31.1 Detailed Description

Plugin interface template for Adobe After Effects plugin development.

This header defines the architecture for a plugin, including commands and their lifecycle management within the plugin. It's designed to be used within the Adobe After Effects SDK environment to create custom plugins.

Author

tjerf

Date

March 2024

7.31.2 Macro Definition Documentation

7.31.2.1 DECLARE_ENTRY

```
#define DECLARE_ENTRY(
                PluginType )
```

Value:

```
    extern "C" DllExport A_Err EntryPointFunc(                                  \
        struct SPBasicSuite *pica_basicP, A_long major_versionL,                \
        A_long minor_versionL, AEGP_PluginID aegp_plugin_id,                    \
        AEGP_GlobalRefcon *global_refconV)                                      \
    {                                                                           \
        SuiteManager::GetInstance().InitializeSuiteHandler(pica_basicP);        \
        return Plugin::EntryPointFunc<PluginType>(                              \
            pica_basicP, major_versionL, minor_versionL, aegp_plugin_id,        \
            global_refconV);                                                    \
    }
```

Macro for defining the plugin's entry point function.

Parameters

PluginType	The type of the plugin to be created.

7.32 Plugin.hpp

Go to the documentation of this file.

```
00001 /**
00002  * @file Plugin.hpp
00003  * @brief Plugin interface template for Adobe After Effects plugin development.
00004  *
00005  * This header defines the architecture for a plugin, including commands and
00006  * their lifecycle management within the plugin. It's designed to be used within
00007  * the Adobe After Effects SDK environment to create custom plugins.
00008  *
00009  * @author tjerf
00010  * @date March 2024
00011  */
00012
00013 #ifndef PLUGIN_HPP
00014 #define PLUGIN_HPP
00015
```

```
00016 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00017 /**
00018  * @class Command
00019  * @brief Abstract base class for creating commands within the plugin.
00020  *
00021  * This class defines the structure for commands that can be executed by the
00022  * plugin. Each command is associated with a specific action or behavior.
00023  */
00024 class Command
00025 {
00026   public:
00027     /**
00028      * Constructs a Command with a name, menu ID, and insertion order.
00029      * @param name The name of the command.
00030      * @param menuID The ID of the menu where the command will be inserted.
00031      * @param after_item Specifies the order of the command within the menu.
00032      * Defaults to INSERT_SORTED.
00033      */
00034     Command(std::string name, AE_MenuID menuID, int after_item = INSERT_SORTED)
00035         : m_name(name), m_commandSuite(CommandSuite1()), m_command(m_commandSuite.getUniqueCommand())
00036     {
00037         insertCommand(menuID, after_item);
00038     }
00039     /**
00040      * Virtual destructor for cleanup.
00041      */
00042     virtual ~Command() = default;
00043     /**
00044      * Executes the command's action. Must be implemented by derived classes.
00045      * This is where you'll execute the command's action or behavior.
00046      * You'll do whatever logic you'd like here-- AE related or not.
00047      */
00048     virtual void execute() = 0;
00049     /**
00050      * Updates the state or appearance of the menu item associated with this
00051      * command. Must be implemented by derived classes.
00052      *
00053      * This is used in the updateMenuHook to update the state of the menu item.
00054      * Use the helper functions to enable, disable, or check the menu item.
00055      */
00056     virtual void updateMenu() = 0;
00057     /**
00058      * Retrieves the name of the command.
00059      * @return The command's name.
00060      */
00061     std::string getName() const { return m_name; }
00062     /**
00063      * Retrieves the command's unique identifier.
00064      * @return The command's identifier.
00065      */
00066     int getCommand() const { return m_command; }
00067
00068     // Helper functions for command manipulation.
00069     inline void insertCommand(AE_MenuID menuID, int after_item = INSERT_SORTED)
00070     {
00071         m_commandSuite.insertMenuCommand(m_command, m_name.c_str(), menuID,
00072                                          after_item);
00073     }
00074     inline void setCommandName(std::string name)
00075     {
00076         m_commandSuite.setMenuCommandName(m_command, name.c_str());
00077     }
00078     inline void enableCommand(bool enable)
00079     {
00080         if (enable)
00081         {
00082             m_commandSuite.enableCommand(m_command);
00083         }
00084         else
00085         {
00086             m_commandSuite.disableCommand(m_command);
00087         }
00088     }
00089
00090     inline void checkCommand(bool check)
00091     {
00092         m_commandSuite.checkMarkMenuCommand(m_command, check);
00093     }
00094
00095   private:
00096     std::string m_name; ///< The command's name.
00097     int m_command;      ///< The command's unique identifier.
00098     CommandSuite1
00099         m_commandSuite; ///< Suite for command operations provided by AE SDK.
00100 };
00101
00102 /**
```

```
00103  * @class Plugin
00104  * @brief Represents the plugin, managing its commands and lifecycle.
00105  *
00106  * This class serves as the central management point for the plugin, handling
00107  * initialization, command registration, and event hooks.
00108  *
00109  * AE Refcons are ignored, as you can use maps to store data instead.
00110  */
00111 class Plugin
00112 {
00113   public:
00114     inline static Plugin *instance;
00115
00116     Plugin(struct SPBasicSuite *pica_basicP, /* » */
00117            AEGP_PluginID aegp_plugin_id,     /* » */
00118            AEGP_GlobalRefcon *global_refconV)
00119         : m_suiteManager(SuiteManager::GetInstance())
00120     {
00121         int id = static_cast<int>(std::hash<std::string>{}(typeid(*this).name()));
00122         AEGP_PluginID myID = id;
00123
00124         myID = aegp_plugin_id;
00125
00126         SuiteManager::GetInstance().SetPluginID(&myID);
00127
00128         instance = this;
00129     }
00130     /**
00131      * Virtual destructor for cleanup.
00132      */
00133     virtual ~Plugin() { instance = nullptr; }
00134
00135     // Lifecycle event handlers to be implemented by derived classes.//
00136     /**
00137     * @brief onInit Initializes the plugin and its commands.
00138      * Initializes the plugin and its commands.
00139      * Here, you will add commands to the plugin's command list, and then use the utility functions
00140      * to register your command hooks (if any).
00141      */
00142     virtual void onInit() = 0;
00143
00144     /**
00145      * Called when the plugin is being unloaded.
00146      * This will automatically clean up commands, its up to you to clean up anything else you need to.
00147      */
00148     virtual void onDeath() = 0;
00149
00150     /**
00151      * Called when the plugin is idle.
00152      * This is a good place to do any background processing or updating of the UI.
00153      * This is also where you would utilize the TaskManager to do background processing.
00154      */
00155     virtual void onIdle() = 0;
00156
00157     /**
00158      * Adds a command to the plugin's command list.
00159      * @param command A unique pointer to the Command object.
00160      */
00161     inline void addCommand(std::unique_ptr<Command> command)
00162     {
00163         m_commands.push_back(std::move(command));
00164     }
00165
00166     template <typename T>
00167     static A_Err EntryPointFunc(struct SPBasicSuite *pica_basicP,  /* » */
00168                                 A_long major_versionL,             /* » */
00169                                 A_long minor_versionL,             /* » */
00170                                 AEGP_PluginID aegp_plugin_id,      /* » */
00171                                 AEGP_GlobalRefcon *global_refconV) /* « */
00172     {
00173         Plugin *plugin = new T(pica_basicP, aegp_plugin_id, global_refconV);
00174         plugin->onInit();
00175         return A_Err_NONE;
00176     }
00177
00178     // Static hook methods for Adobe After Effects to invoke.   // Static hook
00179     // methods for Adobe After Effects to invoke.
00180     inline static A_Err CommandHook(AEGP_GlobalRefcon global_refcon,
00181                                     AEGP_CommandRefcon command_refcon,
00182                                     AEGP_Command command,
00183                                     AEGP_HookPriority hook_priority,
00184                                     A_Boolean already_handled,
00185                                     A_Boolean *handled)
00186     {
00187         if (instance)
00188         {
00189             for (auto &c : instance->m_commands)
```

```
00190             {
00191                 if (c->getCommand() == command)
00192                 {
00193                     c->execute();
00194                     *handled = TRUE;
00195                     return A_Err_NONE;
00196                 }
00197             }
00198         }
00199         *handled = TRUE;
00200         return A_Err_NONE;
00201     }
00202
00203     inline static A_Err UpdateMenuHook(AEGP_GlobalRefcon global_refcon,
00204                                        AEGP_UpdateMenuRefcon update_menu_refcon,
00205                                        AEGP_WindowType active_window)
00206     {
00207         if (instance)
00208         {
00209             for (auto &c : instance->m_commands)
00210             {
00211                 c->updateMenu();
00212             }
00213         }
00214         return A_Err_NONE;
00215     }
00216
00217     inline static A_Err DeathHook(AEGP_GlobalRefcon global_refcon,
00218                                   AEGP_DeathRefcon death_refcon)
00219     {
00220         if (instance)
00221         {
00222             instance->onDeath();
00223         }
00224         return A_Err_NONE;
00225     }
00226
00227     inline static A_Err IdleHook(AEGP_GlobalRefcon global_refcon,
00228                                  AEGP_IdleRefcon idle_refcon,
00229                                  A_long *max_sleepPL)
00230     {
00231         if (instance)
00232         {
00233             instance->onIdle();
00234         }
00235         return A_Err_NONE;
00236     }
00237
00238     // Methods to register the static hook methods with the Adobe After Effects
00239     // SDK.
00240     inline void registerCommandHook()
00241     {
00242         RegisterSuite5().registerCommandHook(*m_suiteManager.GetPluginID(),
00243                                              AEGP_Command_ALL,
00244                                              Plugin::CommandHook, NULL);
00245     }
00246
00247     inline void registerUpdateMenuHook()
00248     {
00249         RegisterSuite5().registerUpdateMenuHook(Plugin::UpdateMenuHook, NULL);
00250     }
00251
00252     inline void registerDeathHook( )
00253     {
00254         RegisterSuite5().registerDeathHook(Plugin::DeathHook, NULL);
00255     }
00256
00257     inline void registerIdleHook()
00258     {
00259         RegisterSuite5().registerIdleHook(Plugin::IdleHook, NULL);
00260     }
00261
00262
00263   private:
00264     SuiteManager &m_suiteManager;
00265     std::vector<std::unique_ptr<Command» m_commands; //use std, depending on preprocessor directives,
      will be either std:: or AE:: (custom allocated and owned by AE)
00266     inline void clearCommands()
00267     {
00268         m_commands.clear();
00269     }
00270 };
00271
00272 // Define a macro for setting up the plugin's entry point function.
00273 /**
00274  * @def DECLARE_ENTRY
00275  * @param PluginType The type of the plugin to be created.
```

```
00276  * @brief Macro for defining the plugin's entry point function.
00277  */
00278 #define DECLARE_ENTRY(PluginType)                                \
00279     extern "C" DllExport A_Err EntryPointFunc(                                  \
00280         struct SPBasicSuite *pica_basicP, A_long major_versionL,                \
00281         A_long minor_versionL, AEGP_PluginID aegp_plugin_id,                    \
00282         AEGP_GlobalRefcon *global_refconV)                                      \
00283     {                                                                           \
00284         SuiteManager::GetInstance().InitializeSuiteHandler(pica_basicP);        \
00285         return Plugin::EntryPointFunc<PluginType>(                              \
00286             pica_basicP, major_versionL, minor_versionL, aegp_plugin_id,        \
00287             global_refconV);                                                    \
00288     }
00289
00290
00291 #endif /* PLUGIN_HPP */
```

7.33 AEGP/Util/Context.hpp File Reference

File Containing Scoped "Context Managers" Currently only supports Scoped_Undo_Guard and Scoped_Quiet_↩ Guard, for scoping Undo Groups and Quiet Mode for error messages.

```
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- class AE::Scoped_Undo_Guard
- class AE::Scoped_Quiet_Guard
- class AE::Scoped_Error_Reporter

  A class that reports errors caught within its scope.

Namespaces

- namespace AE

  Namespace for various pre-defined STL containers using a custom allocator.

7.33.1 Detailed Description

File Containing Scoped "Context Managers" Currently only supports Scoped_Undo_Guard and Scoped_Quiet_↩ Guard, for scoping Undo Groups and Quiet Mode for error messages.

Author

tjerf

Date

March 2024

7.34 Context.hpp

Go to the documentation of this file.

```
00001 /************************************************************************//**
00002  * \file   Context.hpp
00003  * \brief  File Containing Scoped "Context Managers"
00004  * Currently only supports Scoped_Undo_Guard and Scoped_Quiet_Guard, for
00005  * scoping Undo Groups and Quiet Mode for error messages.
00006  *
00007  * \author tjerf
00008  * \date   March 2024
00009  ***************************************************************************/
00010
00011 #ifndef CONTEXT_HPP
00012 #define CONTEXT_HPP
00013
00014 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00015
00016 namespace AE
00017 {
00018 /**
00019  * @brief Scoped_Undo_Guard is a class that is used to start and end an undo
00020  * group
00021  *
00022  * The Scoped_Undo_Guard class is used to manage the scoping of an undo group.
00023  * It provides a convenient way to start and end an undo group by automatically
00024  * calling the corresponding functions from the UtilitySuite6 class.
00025  *
00026  * @example
00027  * void someFunction() {
00028  * {<---- start of scope
00029  * Scoped_Undo_Guard guard("someFunction");
00030  * // do some stuff
00031  * }<---- end of scope, undo group is ended automatically
00032  */
00033 class Scoped_Undo_Guard
00034 {
00035   public:
00036     /**
00037      * Constructs a Scoped_Undo_Guard object with the specified name.
00038      *
00039      * @param name The name of the undo group.
00040      */
00041     Scoped_Undo_Guard(std::string name)
00042     {
00043         UtilitySuite6().startUndoGroup(name);
00044     }
00045
00046     /**
00047      * Destructs the Scoped_Undo_Guard object and ends the undo group.
00048      */
00049     ~Scoped_Undo_Guard() { UtilitySuite6().endUndoGroup(); }
00050 };
00051
00052 /**
00053  * @brief Scoped_Quiet_Guard is a class that is used to quiet error messages
00054  *
00055  * The Scoped_Quiet_Guard class is used to manage the scoping of quiet mode for
00056  * error messages. It provides a convenient way to start and end quiet mode by
00057  * automatically calling the corresponding functions from the UtilitySuite6
00058  * class.
00059  *
00060  * @example
00061  * void someFunction() {
00062  * {<---- start of scope
00063  * Scoped_Quiet_Guard guard;
00064  * // do some stuff
00065  * }<---- end of scope, quiet mode is ended automatically
00066  */
00067 class Scoped_Quiet_Guard
00068 {
00069   public:
00070     /**
00071      * Constructs a Scoped_Quiet_Guard object and starts quiet mode for error
00072      * messages.
00073      */
00074     Scoped_Quiet_Guard() { UtilitySuite6().startQuietErrors(); }
00075
00076     /**
00077      * Destructs the Scoped_Quiet_Guard object and ends quiet mode for error
00078      * messages.
00079      */
00080     ~Scoped_Quiet_Guard() { UtilitySuite6().endQuietErrors(false); }
00081 };
00082 /**
```

```
00083  * @class Scoped_Error_Reporter
00084  *
00085  * @brief A class that reports errors caught within its scope.
00086  *
00087  * The Scoped_Error_Reporter class is responsible for catching and reporting
00088  * errors that occur within its scope. It provides a mechanism to re-throw
00089  * exceptions and handle them appropriately. If an exception is caught, it can
00090  * be reported as a standard or non-standard exception.
00091  */
00092 class Scoped_Error_Reporter
00093 {
00094   public:
00095     /**
00096      * @brief Default constructor for the Scoped_Error_Reporter class.
00097      */
00098     Scoped_Error_Reporter() = default;
00099
00100     /**
00101      * @brief Destructor for the Scoped_Error_Reporter class.
00102      *
00103      * The destructor attempts to re-throw any exception caught during the scope
00104      * of this object. If an exception is caught, it is handled by reporting the
00105      * error message. If no exception is caught, the destructor does nothing. If
00106      * an error occurs while handling the exception, an optional catch block
00107      * logs the error or takes other appropriate actions.
00108      */
00109     ~Scoped_Error_Reporter()
00110     {
00111         try
00112         {
00113             // Attempt to re-throw any exception caught during the scope of this
00114             // object
00115             if (std::current_exception())
00116             { // Checks if there's an active exception
00117                 try
00118                 {
00119                     throw; // Re-throws the caught exception to handle it
00120                 }
00121                 catch (const std::exception &e)
00122                 {
00123                     // Handle standard exceptions by reporting the error message
00124                     ReportError(e.what());
00125                 }
00126                 catch (...)
00127                 {
00128                     // Handle non-standard exceptions by reporting an unknown
00129                     // error message
00130                     ReportError("An unknown error occurred.");
00131                 }
00132             }
00133         }
00134         catch (...)
00135         {
00136             ReportError("An error occurred while handling an exception.");
00137         }
00138     }
00139
00140   private:
00141     /**
00142      * @brief Reports an error message.
00143      *
00144      * This function reports an error message by calling the reportInfoUnicode
00145      * function of the UtilitySuite6 class.
00146      *
00147      * @param errorMessage The error message to be reported.
00148      */
00149     inline void ReportError(const std::string &errorMessage)
00150     {
00151         UtilitySuite6().reportInfoUnicode(errorMessage);
00152     }
00153 };
00154
00155 } // namespace AE
00156
00157 #endif // CONTEXT_HPP
```

7.35 AEGP/Util/Image.hpp File Reference

A class for handling images.

```
#include <stb_image_write.h>
#include "AETK/AEGP/Core/Base/AEGeneral.hpp"
```

Classes

- struct UniformImage
- class Image

7.35.1 Detailed Description

A class for handling images.

Author

tjerf

Date

March 2024

7.36 Image.hpp

Go to the documentation of this file.

```
00001 /*************************************************************************//**
00002  * \file   Image.hpp
00003  * \brief  A class for handling images
00004  *
00005  * \author tjerf
00006  * \date   March 2024
00007  *****************************************************************************/
00008
00009 #ifndef IMAGE_HPP
00010 #define IMAGE_HPP
00011 #define STB_IMAGE_WRITE_IMPLEMENTATION
00012 #define _CRT_SECURE_NO_WARNINGS
00013 #include <stb_image_write.h>
00014
00015
00016 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00017
00018 // Include library headers conditionally
00019 #ifdef USE_OPENCV
00020 #include <opencv2/opencv.hpp>
00021 #endif
00022
00023 #ifdef USE_IMAGEMAGICK
00024 #include <Magick++.h>
00025 #endif
00026
00027
00028
00029 struct UniformImage
00030 {
00031     // Pixel data pointer. Using a void* allows for flexibility with different
00032     // data types.
00033     void *data;
00034
00035     // Image dimensions
00036     int width;
00037     int height;
00038
00039     // Depth per channel in bits (e.g., 8, 16, 32)
00040     // Since channels are always 4, we don't need a separate variable for it
00041     int bitDepth;
00042
```

```
00043     // Row pitch: The number of bytes from one row of pixels in memory to the
00044     // next row. This accounts for any padding at the end of each row.
00045     size_t rowPitch;
00046
00047     // Constructor for initializing the struct with default values
00048     UniformImage()
00049         : data(nullptr), width(0), height(0), bitDepth(0), rowPitch(0)
00050     {
00051     }
00052
00053     // Custom constructor for easy initialization
00054     UniformImage(void *pData, int w, int h, int depth, size_t pitch)
00055         : data(pData), width(w), height(h), bitDepth(depth), rowPitch(pitch)
00056     {
00057     }
00058
00059     // Destructor
00060     ~UniformImage()
00061     {
00062         // If the structure owns 'data', properly free it here
00063         // Consider the memory management strategy based on your use case
00064         data = nullptr;
00065     }
00066
00067     // Since channels are fixed to 4, you might add utility functions that
00068     // specifically handle 4-channel data Example: Converting to and from RGBA
00069     // format used by specific libraries
00070 };
00071
00072 class Image
00073 {//Format is always ARGB
00074   public:
00075     explicit Image(WorldPtr world) : mWorld(world) {}
00076
00077     inline UniformImage data(){
00078         std::tuple<A_long, A_long> size = WorldSuite3().getSize(mWorld);
00079         auto rowbytes = WorldSuite3().getRowBytes(mWorld);
00080         void *baseAddr = nullptr;
00081         int bitDepth = 8; // Default to 8, adjust based on actual data
00082         switch (WorldSuite3().getType(mWorld))
00083         {
00084         case AE_WorldType::NONE:
00085             break;
00086         case AE_WorldType::W8:
00087             baseAddr = WorldSuite3().getBaseAddr8(mWorld);
00088             break;
00089         case AE_WorldType::W16:
00090             baseAddr = WorldSuite3().getBaseAddr16(mWorld);
00091             bitDepth = 16;
00092             break;
00093         case AE_WorldType::W32:
00094             baseAddr = WorldSuite3().getBaseAddr32(mWorld);
00095             bitDepth = 32;
00096             break;
00097         }
00098         return UniformImage(baseAddr, std::get<0>(size), std::get<1>(size), bitDepth, rowbytes);
00099     }
00100 // functions
00101 #ifdef USE_OPENCV
00102     cv::Mat toFormat() {} // Implement conversion to cv::Mat
00103     UniformImage toUniformImage() {} // Implement conversion to UniformImage
00104 #endif
00105
00106 #ifdef USE_IMAGEMAGICK
00107     Magick::Image toFormat() {} // Implement conversion to Magick::Image
00108     UniformImage toUniformImage() {} // Implement conversion to UniformImage
00109 #endif
00110
00111         // New method to save the image using stb_image_write.h
00112     inline void saveImage(const std::string &filename, const std::string &format)
00113     {
00114         auto imageData = data(); // Assume this returns your UniformImage type
00115                                  // with image data
00116         int width = imageData.width;
00117         int height = imageData.height;
00118         int channels = 4; // Assuming ARGB format (4 channels)
00119
00120         // Convert ARGB to RGBA
00121         unsigned char *rgba = new unsigned char[width * height * channels];
00122         unsigned char *argb = static_cast<unsigned char *>(
00123             imageData.data); // Assuming imageData.data is your raw ARGB data
00124
00125         for (int i = 0; i < width * height; ++i)
00126         {
00127             rgba[i * 4 + 0] = argb[i * 4 + 1]; // R
00128             rgba[i * 4 + 1] = argb[i * 4 + 2]; // G
00129             rgba[i * 4 + 2] = argb[i * 4 + 3]; // B
```

```
00130             rgba[i * 4 + 3] =
00131                 argb[i * 4 +
00132                      0]; // A (moved from the first to the last position)
00133         }
00134
00135         // Save the image
00136         if (format == "png")
00137         {
00138             stbi_write_png(filename.c_str(), width, height, channels, rgba,
00139                            width * channels);
00140         }
00141         else if (format == "bmp")
00142         {
00143             stbi_write_bmp(filename.c_str(), width, height, channels, rgba);
00144         }
00145         else if (format == "tga")
00146         {
00147             stbi_write_tga(filename.c_str(), width, height, channels, rgba);
00148         }
00149
00150         // Cleanup
00151         delete[] rgba;
00152     }
00153   private:
00154     WorldPtr mWorld;
00155 };
00156
00157 #endif // IMAGE_HPP
```

7.37 AEGP/Util/Task.hpp File Reference

A class for creating and managing threads.

`#include "AETK/AEGP/Core/Base/AEGeneral.hpp"`

Classes

- class TaskScheduler

  Manages the scheduling and execution of tasks.

Functions

- template<typename ReturnType >
  std::future< ReturnType > ScheduleTask (std::function< ReturnType()> task, bool callIdle=TRUE)

  Schedules a task with a return value.
- void ScheduleTask (std::function< void()> task, bool callIdle=FALSE)

  Schedules a task with no return value.

7.37.1 Detailed Description

A class for creating and managing threads.

Task.hpp

Author

tjerf

Date

March 2024

7.37.2 Function Documentation

7.37.2.1 ScheduleTask() [1/2]

```
template<typename ReturnType >
std::future< ReturnType > ScheduleTask (
            std::function< ReturnType()> task,
            bool callIdle = TRUE )  [inline]
```

Schedules a task with a return value.

Template Parameters

ReturnType	The return type of the task.

Parameters

task	The task to be scheduled.
callIdle	Flag indicating whether to call idle routines for quicker response.

Returns

std::future<ReturnType> A future object representing the result of the task.

7.37.2.2 ScheduleTask() [2/2]

```
void ScheduleTask (
            std::function< void()> task,
            bool callIdle = FALSE )  [inline]
```

Schedules a task with no return value.

Parameters

task	The task to be scheduled.
callIdle	Flag indicating whether to call idle routines for quicker response.

7.38 Task.hpp

Go to the documentation of this file.

```
00001 /*****************************************************************/ /**
00002                                                                    * \file
00003                                                                    *Task.hpp
00004                                                                    * \brief  A
00005                                                                    *class for
00006                                                                    *creating
00007                                                                    *and
00008                                                                    *managing
00009                                                                    *threads
00010                                                                    *
00011                                                                    * \author
```

```
00012                                                                   *tjerf
00013                                                                   * \date
00014                                                                   *March 2024
00015 
      ***************************************************************************/
00016 #ifndef TASK_HPP
00017 #define TASK_HPP
00018 
00019 #include "AETK/AEGP/Core/Base/AEGeneral.hpp"
00020 
00021 /**
00022  * @class TaskScheduler
00023  * @brief Manages the scheduling and execution of tasks.
00024  */
00025 class TaskScheduler
00026 {
00027   public:
00028     /**
00029      * @brief Gets the singleton instance of the TaskScheduler.
00030      * @return TaskScheduler& The singleton instance.
00031      */
00032     static TaskScheduler &GetInstance()
00033     {
00034         static TaskScheduler instance;
00035         return instance;
00036     }
00037 
00038     /**
00039      * @brief Schedules a task with no return value.
00040      * @param task The task to be scheduled.
00041      * @param callIdle Flag indicating whether to call idle routines for quicker
00042      * response.
00043      */
00044     inline void ScheduleTask(std::function<void()> task, bool callIdle = TRUE)
00045     {
00046         std::lock_guard<std::mutex> lock(queueMutex);
00047         tasksQueue.push(std::move(task));
00048         if (callIdle)
00049         {
00050             UtilitySuite6().causeIdleRoutinesToBeCalled();
00051         }
00052     }
00053 
00054     /**
00055      * @brief Schedules a task with a return value.
00056      * @tparam ReturnType The return type of the task.
00057      * @param task The task to be scheduled.
00058      * @param callIdle Flag indicating whether to call idle routines for quicker
00059      * response.
00060      * @return std::future<ReturnType> A future object representing the result
00061      * of the task.
00062      */
00063     template <typename ReturnType>
00064     inline std::future<ReturnType>
00065     ScheduleTask(std::function<ReturnType()> task, bool callIdle = TRUE)
00066     {
00067         auto promise = std::make_shared<std::promise<ReturnType>>();
00068         auto future = promise->get_future();
00069 
00070         // Wrap the user's task to handle the promise/future mechanism
00071         std::function<void()> taskWrapper = [promise, task]() {
00072             try
00073             {
00074                 promise->set_value(task());
00075             }
00076             catch (...)
00077             {
00078                 try
00079                 {
00080                     // Attempt to re-throw the caught exception
00081                     throw;
00082                 }
00083                 catch (...)
00084                 {
00085                     // Store any exception in the promise
00086                     promise->set_exception(std::current_exception());
00087                 }
00088             }
00089         };
00090 
00091         ScheduleTask(taskWrapper, callIdle); // Schedule the wrapped task
00092         return future;
00093     }
00094 
00095     /**
00096      * @brief Executes the next scheduled task.
00097      */
```

```
00098     inline void ExecuteTask()
00099     {
00100         std::lock_guard<std::mutex> lock(queueMutex);
00101         if (!tasksQueue.empty())
00102         {
00103             auto task = tasksQueue.front();
00104             tasksQueue.pop();
00105             task(); // Execute the task
00106         }
00107     }
00108
00109   private:
00110     std::mutex queueMutex;
00111     std::queue<std::function<void()» tasksQueue;
00112 };
00113
00114 /**
00115  * @brief Schedules a task with a return value.
00116  * @tparam ReturnType The return type of the task.
00117  * @param task The task to be scheduled.
00118  * @param callIdle Flag indicating whether to call idle routines for quicker
00119  * response.
00120  * @return std::future<ReturnType> A future object representing the result of
00121  * the task.
00122  */
00123 template <typename ReturnType>
00124 inline std::future<ReturnType> ScheduleTask(std::function<ReturnType()> task,
00125                                             bool callIdle = TRUE)
00126 {
00127     return TaskScheduler::GetInstance().ScheduleTask(task, callIdle);
00128 }
00129
00130 /**
00131  * @brief Schedules a task with no return value.
00132  * @param task The task to be scheduled.
00133  * @param callIdle Flag indicating whether to call idle routines for quicker
00134  * response.
00135  */
00136 inline void ScheduleTask(std::function<void()> task, bool callIdle = FALSE)
00137 {
00138     TaskScheduler::GetInstance().ScheduleTask(task, callIdle);
00139 }
00140
00141 #endif // TASK_HPP
```

7.39 Effect/Core/Base/AEffect.hpp File Reference

AEffect class declaration.

```
#include "Headers/AE_Effect.h"
#include <vector>
```

Classes

- class Param
- class ParamConfig

7.39.1 Detailed Description

AEffect class declaration.

Author

tjerf

Date

March 2024

7.40 AEffect.hpp

Go to the documentation of this file.

```
00001 /*****************************************************************//**
00002  * \file   AEffect.hpp
00003  * \brief  AEffect class declaration
00004  *
00005  * \author tjerf
00006  * \date   March 2024
00007  *********************************************************************/
00008
00009 #ifndef AEFFECT_HPP
00010 #define AEFFECT_HPP
00011
00012 #include "Headers/AE_Effect.h"
00013 #include <vector>
00014
00015 class Param
00016 {
00017     static Param color(double r, double g, double b);
00018
00019 };
00020
00021 class ParamConfig
00022 {
00023     void setParam(Param param);
00024     void registerParams();
00025
00026   private:
00027     std::vector<Param> params;
00028 };
00029
00030 #endif // AEFFECT_HPP
```

7.41 Effect/Core/Base/AEffectGeneral.hpp File Reference

AEffectGeneral class declaration.

```
#include "Headers/AE_Effect.h"
#include "Headers/AE_EffectCB.h"
```

7.41.1 Detailed Description

AEffectGeneral class declaration.

Author

tjerf

Date

March 2024

7.42 AEffectGeneral.hpp

Go to the documentation of this file.

```
00001 /*****************************************************************//**
00002  * \file   AEffectGeneral.hpp
00003  * \brief  AEffectGeneral class declaration
00004  *
00005  * \author tjerf
00006  * \date   March 2024
00007  *********************************************************************/
00008 #ifndef AEFFECTGENERAL_HPP
00009 #define AEFFECTGENERAL_HPP
00010
00011 #include "Headers/AE_Effect.h"
00012 #include "Headers/AE_EffectCB.h"
00013
00014
00015 #endif //AEFFECTGENERAL_HPP
```

7.43 Effect/Core/Base/AEffectSuiteManager.hpp File Reference

Singleton class managing the After Effects suite handler and plugin ID.

```
#include "Util/AEGP_SuiteHandler.h"
#include "Headers/AE_Macros.h"
```

Classes

- class SuiteManager

  Singleton class managing the After Effects suite handler and plugin ID.

7.43.1 Detailed Description

Singleton class managing the After Effects suite handler and plugin ID.

SuiteManager.hpp

Author

tjerf

Date

March 2024

7.44 AEffectSuiteManager.hpp

Go to the documentation of this file.

```
00001 /**
00002  * \file
00003  *SuiteManager.hpp
00004  * \brief
00005  *Singleton
00006  *class
00007  *managing
00008  *the After
00009  *Effects
00010  *suite
00011  *handler
00012  *and plugin
00013  *ID.
00014  *
00015  * \author
00016  *tjerf
00017  * \date
00018  *March 2024
00019  *********************************************************************/
00020 #pragma once
00021
00022 #include "Util/AEGP_SuiteHandler.h"
00023 #include "Headers/AE_Macros.h"
00024 /*
00025  * File: SuiteManager.h
00026  * Description: Singleton class managing the After Effects suite handler and
00027  * plugin ID.
00028  *
00029  * Guidelines for Contributors:
00030  * 1. Singleton Pattern: Recognize that SuiteManager is a singleton and should
00031  * not be instantiated directly.
```

```
00032  * 2. Suite Handling: Understand how SuiteManager provides access to AE suites.
00033  * 3. No Alteration: Do not modify this file. It is crucial for the stable
00034  * operation of the entire plugin.
00035  */
00036
00037 /**
00038  * @class SuiteManager
00039  * @brief Singleton class managing the After Effects suite handler and plugin
00040  * ID.
00041  *
00042  * The SuiteManager class is responsible for managing the After Effects suite
00043  * handler and plugin ID. It follows the Singleton pattern to ensure that only
00044  * one instance of the class can exist. The class provides methods to initialize
00045  * the suite handler, get the suite handler, set the plugin ID, and get the
00046  * plugin ID.
00047  */
00048 class SuiteManager
00049 {
00050   public:
00051     /**
00052      * @brief Gets the singleton instance of SuiteManager.
00053      *
00054      * This method returns the singleton instance of the SuiteManager class.
00055      *
00056      * @return SuiteManager& The reference to the singleton instance of
00057      * SuiteManager.
00058      */
00059     static SuiteManager &GetInstance()
00060     {
00061         static SuiteManager instance;
00062         return instance;
00063     }
00064
00065     // Deleted copy constructor and assignment operator to ensure singleton
00066     SuiteManager(SuiteManager const &) = delete;
00067     void operator=(SuiteManager const &) = delete;
00068
00069     /**
00070      * @brief Initializes the suite handler.
00071      *
00072      * This method initializes the suite handler with the provided SPBasicSuite
00073      * pointer. It should be called before accessing any AE suites.
00074      *
00075      * @param pica_basicP The SPBasicSuite pointer.
00076      */
00077     void InitializeSuiteHandler(SPBasicSuite *pica_basicP)
00078     {
00079         if (!suitesInitialized)
00080         {
00081             suites = new AEGP_SuiteHandler(pica_basicP);
00082             suitesInitialized = true;
00083         }
00084     }
00085
00086     /**
00087      * @brief Gets the suite handler.
00088      *
00089      * This method returns a reference to the suite handler.
00090      *
00091      * @return AEGP_SuiteHandler& The reference to the suite handler.
00092      */
00093     AEGP_SuiteHandler &GetSuiteHandler() { return *suites; }
00094
00095     /**
00096      * @brief Sets the plugin ID.
00097      *
00098      * This method sets the plugin ID with the provided AEGP_PluginID pointer.
00099      *
00100      * @param pluginIDPtr The AEGP_PluginID pointer.
00101      */
00102     void SetPluginID(AEGP_PluginID *pluginIDPtr)
00103     {
00104         this->pluginIDPtr = pluginIDPtr;
00105     }
00106
00107     /**
00108      * @brief Gets the plugin ID.
00109      *
00110      * This method returns a constant pointer to the plugin ID.
00111      *
00112      * @return const AEGP_PluginID* The constant pointer to the plugin ID.
00113      */
00114     AEGP_PluginID *GetPluginID() const { return pluginIDPtr; }
00115
00116   private:
00117     /**
00118      * @brief Default constructor.
```

```
00119      *
00120      * The default constructor is private to prevent direct instantiation of the
00121      * SuiteManager class.
00122      */
00123     SuiteManager()
00124         : suites(nullptr), suitesInitialized(false), pluginIDPtr(nullptr)
00125     {
00126     }
00127
00128     AEGP_SuiteHandler *suites; /**< Pointer to the suite handler. */
00129     bool suitesInitialized; /**< Flag indicating if the suite handler has been
00130                                initialized. */
00131     AEGP_PluginID *pluginIDPtr; /**< Pointer to the plugin ID. */
00132 };
```

Chapter 8

Examples

8.1 C:/Users/tjerf/source/AETK/AETK/AEGP/Util/Context.hpp

Scoped_Undo_Guard is a class that is used to start and end an undo group.

Scoped_Undo_Guard is a class that is used to start and end an undo groupThe Scoped_Undo_Guard class is used to manage the scoping of an undo group. It provides a convenient way to start and end an undo group by automatically calling the corresponding functions from the UtilitySuite6 class.

void someFunction() { {<-— start of scope Scoped_Undo_Guard guard("someFunction"); // do some stuff }<-— end of scope, undo group is ended automatically

```
/*****************************************************************//**
 * \file   Context.hpp
 * \brief  File Containing Scoped "Context Managers"
 * Currently only supports Scoped_Undo_Guard and Scoped_Quiet_Guard, for
 * scoping Undo Groups and Quiet Mode for error messages.
 *
 * \author tjerf
 * \date   March 2024
 *********************************************************************/

#ifndef CONTEXT_HPP
#define CONTEXT_HPP

#include "AETK/AEGP/Core/Base/AEGeneral.hpp"

namespace AE
{
/**
 * @brief Scoped_Undo_Guard is a class that is used to start and end an undo
 * group
 *
 * The Scoped_Undo_Guard class is used to manage the scoping of an undo group.
 * It provides a convenient way to start and end an undo group by automatically
 * calling the corresponding functions from the UtilitySuite6 class.
 *
 * @example
 * void someFunction() {
 * {<---- start of scope
 * Scoped_Undo_Guard guard("someFunction");
 * // do some stuff
 * }<---- end of scope, undo group is ended automatically
 */
class Scoped_Undo_Guard
{
  public:
    /**
     * Constructs a Scoped_Undo_Guard object with the specified name.
     *
     * @param name The name of the undo group.
     */
    Scoped_Undo_Guard(std::string name)
    {
        UtilitySuite6().startUndoGroup(name);
    }
```

```
    /**
     * Destructs the Scoped_Undo_Guard object and ends the undo group.
     */
    ~Scoped_Undo_Guard() { UtilitySuite6().endUndoGroup(); }
};

/**
 * @brief Scoped_Quiet_Guard is a class that is used to quiet error messages
 *
 * The Scoped_Quiet_Guard class is used to manage the scoping of quiet mode for
 * error messages. It provides a convenient way to start and end quiet mode by
 * automatically calling the corresponding functions from the UtilitySuite6
 * class.
 *
 * @example
 * void someFunction() {
 * {<---- start of scope
 * Scoped_Quiet_Guard guard;
 * // do some stuff
 * }<---- end of scope, quiet mode is ended automatically
 */
class Scoped_Quiet_Guard
{
  public:
    /**
     * Constructs a Scoped_Quiet_Guard object and starts quiet mode for error
     * messages.
     */
    Scoped_Quiet_Guard() { UtilitySuite6().startQuietErrors(); }

    /**
     * Destructs the Scoped_Quiet_Guard object and ends quiet mode for error
     * messages.
     */
    ~Scoped_Quiet_Guard() { UtilitySuite6().endQuietErrors(false); }
};
/**
 * @class Scoped_Error_Reporter
 *
 * @brief A class that reports errors caught within its scope.
 *
 * The Scoped_Error_Reporter class is responsible for catching and reporting
 * errors that occur within its scope. It provides a mechanism to re-throw
 * exceptions and handle them appropriately. If an exception is caught, it can
 * be reported as a standard or non-standard exception.
 */
class Scoped_Error_Reporter
{
  public:
    /**
     * @brief Default constructor for the Scoped_Error_Reporter class.
     */
    Scoped_Error_Reporter() = default;

    /**
     * @brief Destructor for the Scoped_Error_Reporter class.
     *
     * The destructor attempts to re-throw any exception caught during the scope
     * of this object. If an exception is caught, it is handled by reporting the
     * error message. If no exception is caught, the destructor does nothing. If
     * an error occurs while handling the exception, an optional catch block
     * logs the error or takes other appropriate actions.
     */
    ~Scoped_Error_Reporter()
    {
        try
        {
            // Attempt to re-throw any exception caught during the scope of this
            // object
            if (std::current_exception())
            { // Checks if there's an active exception
                try
                {
                    throw; // Re-throws the caught exception to handle it
                }
                catch (const std::exception &e)
                {
                    // Handle standard exceptions by reporting the error message
                    ReportError(e.what());
                }
                catch (...)
                {
                    // Handle non-standard exceptions by reporting an unknown
                    // error message
                    ReportError("An unknown error occurred.");
                }
```

```
            }
        }
        catch (...)
        {
            ReportError("An error occurred while handling an exception.");
        }
    }

  private:
    /**
     * @brief Reports an error message.
     *
     * This function reports an error message by calling the reportInfoUnicode
     * function of the UtilitySuite6 class.
     *
     * @param errorMessage The error message to be reported.
     */
    inline void ReportError(const std::string &errorMessage)
    {
        UtilitySuite6().reportInfoUnicode(errorMessage);
    }
};

} // namespace AE

#endif // CONTEXT_HPP
```

Index