lib3mf

Release v2.3.1

3MF Consortium

Mar 18, 2024

CONTENTS

1 API documentation **3**
- 1.1 C++-language bindings . . . 3
  - 1.1.1 Minimal Example Project . . . 4
  - 1.1.2 General Information . . . 6
  - 1.1.3 API-Classes . . . 17
- 1.2 C-language bindings . . . 62
- 1.3 Python-language bindings . . . 62
- 1.4 Pascal-language bindings . . . 62
- 1.5 C#-language bindings . . . 62
- 1.6 Golang-language bindings . . . 62
- 1.7 NodeJS-language bindings . . . 62

2 Obtaining lib3mf **63**

3 Using lib3mf **65**

4 Meta Information **67**

5 Indices and tables **69**

Index **71**

Welcome! This is the documentation for lib3mf v2.3.1.

lib3mf is an implementation of the 3D Manufacturing Format file standard.

It provides 3MF reading and writing capabilities, as well as conversion and validation tools for input and output data. lib3mf runs on Windows, Linux and MacOS and offers a clean and easy-to-use API in various programming languages to speed up the development and keep integration costs at a minimum.

As 3MF shall become an universal 3D Printing standard, its quick adoption is very important. This library shall lower all barriers of adoption to any possible user, let it be software providers, hardware providers, service providers or middleware tools.

CHAPTER

ONE

API DOCUMENTATION

- *C++*
- *C*
- *Python*
- *Pascal*
- *C#*
- *Golang*
- *NodeJS*

1.1 C++-language bindings

This space describes the usage of lib3mf in a C++ host application.

The C++-language bindings come in two different flavors:

Cpp

CppDynamic

If you include the header `Cpp/lib3mf_implicit.hpp`, lib3mf will be loaded dynamically during *load-time* of your host application through your operating system's mechanism for loading libraries.

```
Lib3MF::PWrapper wrapper = CWrapper::loadLibrary();
```

The shared library file `lib3mf.*`. needs to reside in a path that your operating systems checks when loading libraries.

If you include the header `CppDynamic/lib3mf_dynamic.hpp`, Lib3MF will be loaded dynamically during *run-time* of your host application through an explicit call to

Linux

Mac OSX

Windows

```
Lib3MF::PWrapper wrapper = Lib3MF::CWrapper::loadLibrary("LibraryLocation/lib3mf.so");
```

```
Lib3MF::PWrapper wrapper = Lib3MF::CWrapper::loadLibrary("LibraryLocation/lib3mf.dylib");
```

```
Lib3MF::PWrapper wrapper = Lib3MF::CWrapper::loadLibrary("LibraryLocation/lib3mf.dll");
```

i.e. you need to explicitly provide the location of the shared library file `lib3mf.*`.

The `Lib3MF::PWrapper` object provides access to all functionality within lib3mf.

Both flavors of the C++-bindings are header-only which makes it extremly easy to include them into existing projects:

1.1.1 Minimal Example Project

Minimal application code:

Cpp

CppDynamic

```cpp
#include <iostream>
#include "lib3mf_implicit.hpp"


int main()
{
  try
  {
    auto wrapper = Lib3MF::CWrapper::loadLibrary();
    Lib3MF_uint32 nMajor, nMinor, nMicro;
    wrapper->GetLibraryVersion(nMajor, nMinor, nMicro);
    std::cout << "Lib3MF.Version = " << nMajor << "." << nMinor << "." << nMicro;
    std::string sPreReleaseInfo;
    if (wrapper->GetPrereleaseInformation(sPreReleaseInfo)) {
      std::cout << "-" << sPreReleaseInfo;
    }
    std::string sBuildInfo;
    if (wrapper->GetBuildInformation(sBuildInfo)) {
      std::cout << "+" << sBuildInfo;
    }
    std::cout << std::endl;
  }
  catch (std::exception &e)
  {
    std::cout << e.what() << std::endl;
    return 1;
  }
  return 0;
}
```

```cpp
#include <iostream>
#include "lib3mf_dynamic.hpp"


int main()
{
  try
  {
    std::string libpath = (""); // TODO: put the location of the Lib3MF-library file␣
↪here.
```

(continues on next page)

(continued from previous page)

```
    auto wrapper = Lib3MF::CWrapper::loadLibrary(libpath + "/lib3mf."); // TODO: add␣
↪correct suffix of the library
    Lib3MF_uint32 nMajor, nMinor, nMicro;
    wrapper->GetLibraryVersion(nMajor, nMinor, nMicro);
    std::cout << "Lib3MF.Version = " << nMajor << "." << nMinor << "." << nMicro;
    std::string sPreReleaseInfo;
    if (wrapper->GetPrereleaseInformation(sPreReleaseInfo)) {
      std::cout << "-" << sPreReleaseInfo;
    }
    std::string sBuildInfo;
    if (wrapper->GetBuildInformation(sBuildInfo)) {
      std::cout << "+" << sBuildInfo;
    }
    std::cout << std::endl;
  }
  catch (std::exception &e)
  {
    std::cout << e.what() << std::endl;
    return 1;
  }
  return 0;
}
```

CMakeLists.txt for minimal project:

Cpp

CppDynamic

```
cmake_minimum_required(VERSION 3.5)

set(CMAKE_CURRENT_BINDING_DIR ${CMAKE_CURRENT_SOURCE_DIR}/.)
project(Lib3MFExample_CPPImplicit)
set(CMAKE_CXX_STANDARD 11)
add_executable(Lib3MFExample_CPPImplicit "${CMAKE_CURRENT_SOURCE_DIR}/Lib3MF_example.cpp
↪")
find_library(LIB3MFLOCATION lib3mf "${CMAKE_CURRENT_SOURCE_DIR}/../../Implementations/*/
↪*/*")
target_link_libraries(Lib3MFExample_CPPImplicit ${LIB3MFLOCATION})
target_include_directories(Lib3MFExample_CPPImplicit PRIVATE "${CMAKE_CURRENT_BINDING_
↪DIR}")
```

```
cmake_minimum_required(VERSION 3.5)

set(CMAKE_CURRENT_BINDING_DIR ${CMAKE_CURRENT_SOURCE_DIR}/.)
project(Lib3MFExample_CPPDynamic)
set(CMAKE_CXX_STANDARD 11)
add_executable(Lib3MFExample_CPPDynamic "${CMAKE_CURRENT_SOURCE_DIR}/Lib3MF_example.cpp")
if (UNIX)
        target_link_libraries(Lib3MFExample_CPPDynamic ${CMAKE_DL_LIBS})
endif (UNIX)
target_include_directories(Lib3MFExample_CPPDynamic PRIVATE "${CMAKE_CURRENT_BINDING_DIR}
↪")
```

The examples in the Cpp/CppDynamic-folders of the binary SDK follow exactly this pattern.

The remainder of this space is an in-depth API-reference for the functionality of lib3mf.

1.1.2 General Information

The wrapper class CWrapper

class `Lib3MF::CWrapper`

All types of the 3MF Library reside in the namespace Lib3MF and all functionality of the 3MF Library resides in Lib3MF::CWrapper.

A suitable way to use Lib3MF::CWrapper is as a singleton.

void **GetLibraryVersion**(*Lib3MF_uint32* &nMajor, *Lib3MF_uint32* &nMinor, *Lib3MF_uint32* &nMicro)

retrieves the binary version of this library.

Parameters

- **nMajor** – returns the major version of this library
- **nMinor** – returns the minor version of this library
- **nMicro** – returns the micro version of this library

bool **GetPrereleaseInformation**(std::string &sPrereleaseInfo)

retrieves prerelease information of this library.

Returns
Does the library provide prerelease version?

Parameters
sPrereleaseInfo – retrieves prerelease information of this library.

bool **GetBuildInformation**(std::string &sBuildInformation)

retrieves build information of this library.

Returns
Does the library provide build version?

Parameters
sBuildInformation – retrieves build information of this library.

void **GetSpecificationVersion**(const std::string &sSpecificationURL, bool &bIsSupported, *Lib3MF_uint32* &nMajor, *Lib3MF_uint32* &nMinor, *Lib3MF_uint32* &nMicro)

retrieves whether a specification is supported, and if so, which version.

Parameters

- **sSpecificationURL** – URL of extension to check
- **bIsSupported** – returns whether this specification is supported
- **nMajor** – returns the major version of the extension (if IsSupported)
- **nMinor** – returns the minor version of the extension (if IsSupported)
- **nMicro** – returns the micro version of the extension (if IsSupported)

PModel **CreateModel**()

creates an empty model instance.

Returns
returns an empty model instance

void **Release**(*CBase* *pInstance)

releases shared ownership of an object instance

Parameters
pInstance – the object instance to release

void **Acquire**(*CBase* *pInstance)

acquires shared ownership of an object instance

Parameters
pInstance – the object instance to acquire

void **SetJournal**(const std::string &sJournalPath)

Sets the journal file path

Parameters
sJournalPath – File name of the journal file

bool **GetLastError**(*CBase* *pInstance, std::string &sLastErrorString)

Retrieves the last error string of an instance

Parameters

- **pInstance** – Object where the error occured.
- **sLastErrorString** – Last Error String

Returns
Returns if the instance has a last error.

void **RetrieveProgressMessage**(const eProgressIdentifier eTheProgressIdentifier, std::string &sProgressMessage)

Return an English text for a progress identifier.|Note: this is the only function you can call from your callback function.

Parameters

- **eTheProgressIdentifier** – the progress identifier that is passed to the callback function
- **sProgressMessage** – English text for the progress identifier

sColor **RGBAToColor**(const *Lib3MF_uint8* nRed, const *Lib3MF_uint8* nGreen, const *Lib3MF_uint8* nBlue, const *Lib3MF_uint8* nAlpha)

Creates a Color from uint8 RGBA values

Parameters

- **nRed** – Red value of color (0-255)
- **nGreen** – Green value of color (0-255)
- **nBlue** – Blue value of color (0-255)
- **nAlpha** – Alpha value of color (0-255)

Returns
Assembled color

sColor **FloatRGBAToColor**(const *Lib3MF_single* fRed, const *Lib3MF_single* fGreen, const *Lib3MF_single* fBlue, const *Lib3MF_single* fAlpha)

Creates a Color from uint8 RGBA values

Parameters

- **fRed** – Red value of color (0-1)
- **fGreen** – Green value of color (0-1)
- **fBlue** – Blue value of color (0-1)
- **fAlpha** – Alpha value of color (0-1)

Returns
Assembled color

void **ColorToRGBA**(const *sColor* &TheColor, *Lib3MF_uint8* &nRed, *Lib3MF_uint8* &nGreen, *Lib3MF_uint8* &nBlue, *Lib3MF_uint8* &nAlpha)

Calculates uint8-RGBA-values from a Color

Parameters

- **TheColor** – Color to handle
- **nRed** – Red value of color (0-255)
- **nGreen** – Green value of color (0-255)
- **nBlue** – Blue value of color (0-255)
- **nAlpha** – Alpha value of color (0-255)

void **ColorToFloatRGBA**(const *sColor* &TheColor, *Lib3MF_single* &fRed, *Lib3MF_single* &fGreen, *Lib3MF_single* &fBlue, *Lib3MF_single* &fAlpha)

Calculates float-RGBA-values from a Color

Parameters

- **TheColor** – Color to handle
- **fRed** – Red value of color (0-1)
- **fGreen** – Green value of color (0-1)
- **fBlue** – Blue value of color (0-1)
- **fAlpha** – Alpha value of color (0-1)

sTransform **GetIdentityTransform**()

Creates an identity transform

Returns
Transformation matrix.

sTransform **GetUniformScaleTransform**(const *Lib3MF_single* fFactor)

Creates a uniform scale transform

Parameters
fFactor – Factor in X, Y and Z

Returns
Transformation matrix.

sTransform **GetScaleTransform**(const *Lib3MF_single* fFactorX, const *Lib3MF_single* fFactorY, const *Lib3MF_single* fFactorZ)

Creates a scale transform

Parameters

- **fFactorX** – Factor in X
- **fFactorY** – Factor in Y
- **fFactorZ** – Factor in Z

Returns
Transformation matrix.

sTransform **GetTranslationTransform**(const *Lib3MF_single* fVectorX, const *Lib3MF_single* fVectorY, const *Lib3MF_single* fVectorZ)

Creates an translation transform

Parameters

- **fVectorX** – Translation in X
- **fVectorY** – Translation in Y
- **fVectorZ** – Translation in Z

Returns
Transformation matrix.

typedef std::shared_ptr<*CWrapper*> Lib3MF::**PWrapper**

Types used in the 3MF Library

Simple types

typedef uint8_t **Lib3MF_uint8**

typedef uint16_t **Lib3MF_uint16**

typedef uint32_t **Lib3MF_uint32**

typedef uint64_t **Lib3MF_uint64**

typedef int8_t **Lib3MF_int8**

typedef int16_t **Lib3MF_int16**

typedef int32_t **Lib3MF_int32**

typedef int64_t **Lib3MF_int64**

typedef float **Lib3MF_single**

typedef double **Lib3MF_double**

using **Lib3MF_pvoid** = void*

using **Lib3MFResult** = *Lib3MF_int32*

Enumerations

enum class **ePropertyType** : *Lib3MF_int32*

enumerator **NoPropertyType** = 0

enumerator **BaseMaterial** = 1

enumerator **TexCoord** = 2

enumerator **Colors** = 3

enumerator **Composite** = 4

enumerator **Multi** = 5

enum class **eSlicesMeshResolution** : *Lib3MF_int32*

enumerator **Fullres** = 0

enumerator **Lowres** = 1

enum class **eModelUnit** : *Lib3MF_int32*

enumerator **MicroMeter** = 0

enumerator **MilliMeter** = 1

enumerator **CentiMeter** = 2

enumerator **Inch** = 3

enumerator **Foot** = 4

enumerator **Meter** = 5

enum class **eObjectType** : *Lib3MF_int32*

enumerator **Other** = 0

enumerator **Model** = 1

enumerator **Support** = 2

enumerator **SolidSupport** = 3

enum class **eTextureType** : *Lib3MF_int32*

enumerator **Unknown** = 0

enumerator **PNG** = 1

enumerator **JPEG** = 2

enum class **eTextureTileStyle** : *Lib3MF_int32*

enumerator **Wrap** = 0

enumerator **Mirror** = 1

enumerator **Clamp** = 2

enumerator **NoTileStyle** = 3

enum class **eTextureFilter** : *Lib3MF_int32*

enumerator **Auto** = 0

enumerator **Linear** = 1

enumerator **Nearest** = 2

enum class **eBeamLatticeCapMode** : *Lib3MF_int32*

enumerator **Sphere** = 0

enumerator **HemiSphere** = 1

enumerator **Butt** = 2

enum class **eBeamLatticeClipMode** : *Lib3MF_int32*

enumerator **NoClipMode** = 0

enumerator **Inside** = 1

enumerator **Outside** = 2

enum class **eBeamLatticeBallMode** : *Lib3MF_int32*

enumerator **None** = 0

enumerator **Mixed** = 1

enumerator **All** = 2

enum class **eProgressIdentifier** : *Lib3MF_int32*

enumerator **QUERYCANCELED** = 0

enumerator **DONE** = 1

enumerator **CLEANUP** = 2

enumerator **READSTREAM** = 3

enumerator **EXTRACTOPCPACKAGE** = 4

enumerator **READNONROOTMODELS** = 5

enumerator **READROOTMODEL** = 6

enumerator **READRESOURCES** = 7

enumerator **READMESH** = 8

enumerator **READSLICES** = 9

enumerator **READBUILD** = 10

enumerator **READCUSTOMATTACHMENT** = 11

enumerator **READTEXTURETACHMENTS** = 12

enumerator **CREATEOPCPACKAGE** = 13

enumerator **WRITEMODELSTOSTREAM** = 14

enumerator **WRITEROOTMODEL** = 15

enumerator **WRITENONROOTMODELS** = 16

enumerator **WRITEATTACHMENTS** = 17

enumerator **WRITECONTENTTYPES** = 18

enumerator **WRITENOBJECTS** = 19

enumerator **WRITENODES** = 20

enumerator **WRITETRIANGLES** = 21

enumerator **WRITESLICES** = 22

enumerator **WRITEKEYSTORE** = 23

enum class **eBlendMethod** : *Lib3MF_int32*

enumerator **NoBlendMethod** = 0

enumerator **Mix** = 1

enumerator **Multiply** = 2

enum class **eEncryptionAlgorithm** : *Lib3MF_int32*

enumerator **AES256_GCM** = 1

enum class **eWrappingAlgorithm** : *Lib3MF_int32*

enumerator **RSA_OAEP** = 0

enum class **eMgfAlgorithm** : *Lib3MF_int32*

enumerator **MGF1_SHA1** = 160

enumerator **MGF1_SHA224** = 224

enumerator **MGF1_SHA256** = 256

enumerator **MGF1_SHA384** = 384

enumerator **MGF1_SHA512** = 512

enum class **eDigestMethod** : *Lib3MF_int32*

enumerator **SHA1** = 160

enumerator **SHA256** = 256

enum class **eCompression** : *Lib3MF_int32*

enumerator **NoCompression** = 0

enumerator **Deflate** = 1

Structs

All structs are defined as *packed*, i.e. with the

```
#pragma pack (1)
```

struct **sTriangle**

Lib3MF_uint32 **m_Indices**[3]

struct **sTriangleProperties**

Lib3MF_uint32 **m_ResourceID**

Lib3MF_uint32 **m_PropertyIDs**[3]

struct **sPosition**

Lib3MF_single **m_Coordinates**[3]

struct **sPosition2D**

Lib3MF_single **m_Coordinates**[2]

struct **sCompositeConstituent**

Lib3MF_uint32 **m_PropertyID**

Lib3MF_double **m_MixingRatio**

struct **sMultiPropertyLayer**

Lib3MF_uint32 **m_ResourceID**

eBlendMethod **m_TheBlendMethod**

struct **sTex2Coord**

Lib3MF_double **m_U**

Lib3MF_double **m_V**

struct **sTransform**

Lib3MF_single **m_Fields**[4][3]

struct **sBox**

Lib3MF_single **m_MinCoordinate**[3]

Lib3MF_single **m_MaxCoordinate**[3]

struct **sColor**

Lib3MF_uint8 **m_Red**

Lib3MF_uint8 **m_Green**

Lib3MF_uint8 **m_Blue**

Lib3MF_uint8 **m_Alpha**

struct **sBeam**

Lib3MF_uint32 **m_Indices**[2]

Lib3MF_double **m_Radii**[2]

eBeamLatticeCapMode **m_CapModes**[2]

struct **sBall**

Lib3MF_uint32 **m_Index**

Lib3MF_double **m_Radius**

Function types

using **ProgressCallback** = void (*)(bool*, *Lib3MF_double*, Lib3MF::eProgressIdentifier, *Lib3MF_pvoid*)

A callback function

Return
Returns whether the calculation should be aborted

Param dProgressValue
The value of the progress function: values in the interval [0,1] are progress; < mean no progress update

Param eProgressIdentifier
An identifier of progress

Param pUserData
Userdata that is passed to the callback function

using **WriteCallback** = void (*)(*Lib3MF_uint64*, *Lib3MF_uint64*, *Lib3MF_pvoid*)

Callback to call for writing a data chunk

Param nByteData
Pointer to the data to be written

Param nNumBytes
Number of bytes to write

Param pUserData
Userdata that is passed to the callback function

using **ReadCallback** = void (*)(*Lib3MF_uint64*, *Lib3MF_uint64*, *Lib3MF_pvoid*)

Callback to call for reading a data chunk

Param nByteData
Pointer to a buffer to read data into

Param nNumBytes
Number of bytes to read

Param pUserData
Userdata that is passed to the callback function

using **SeekCallback** = void (*)(*Lib3MF_uint64*, *Lib3MF_pvoid*)

Callback to call for seeking in the stream

Param nPosition
Position in the stream to move to

Param pUserData
Userdata that is passed to the callback function

using **RandomNumberCallback** = void (*)(*Lib3MF_uint64*, *Lib3MF_uint64*, *Lib3MF_pvoid*, *Lib3MF_uint64**)

Callback to generate random numbers

Param nByteData
Pointer to a buffer to read data into

Param nNumBytes
Size of available bytes in the buffer

Param pUserData
Userdata that is passed to the callback function

Return
Number of bytes generated when succeed. 0 or less if failed.

using **KeyWrappingCallback** = void (*)(Lib3MF_AccessRight, *Lib3MF_uint8**, *Lib3MF_uint8***, *Lib3MF_pvoid*, *Lib3MF_uint64**)

A callback used to wrap (encrypt) the content key available in keystore resource group

Param pKEKParams
The information about the parameters used used to wrap the key to the contents

Param nInBufferBufferSize
Buffer to the input value. When encrypting, this should be the plain key. When decrypting, this should be the key cipher.

Param nInBufferBufferSize
buffer of Buffer to the input value. When encrypting, this should be the plain key. When decrypting, this should be the key cipher.

Param nOutBufferBufferSize
Number of elements in buffer

Param pOutBufferNeededCount
will be filled with the count of the written elements, or needed buffer size.

Param pOutBufferBuffer
buffer of Buffer where the data will be placed. When encrypting, this will be the key cipher. When decrypting, this will be the plain key. When buffer is null, neededBytes contains the required bytes to run.

Param pUserData
Userdata that is passed to the callback function

Return
The needed/encrypted/decrypted bytes when succeed or zero when error.

using **ContentEncryptionCallback** = void (*)(Lib3MF_ContentEncryptionParams, *Lib3MF_uint8**, *Lib3MF_uint8***, *Lib3MF_pvoid*, *Lib3MF_uint64**)

A callback to encrypt/decrypt content called on each resource encrypted. This might be called several

times depending on content size. If Input is not available(either null or size is 0), clients must return the result of authenticated tag generation/validation.

Param pCEKParams
The params of the encryption process. Client must set/check AuthenticationTag when closing the encryption/decryption process.

Param nInputBufferSize
Buffer to the original data. In encrypting, this will be the plain data. If decrypting, this will be the cipher data

Param nInputBufferSize
buffer of Buffer to the original data. In encrypting, this will be the plain data. If decrypting, this will be the cipher data

Param nOutputBufferSize
Number of elements in buffer

Param pOutputNeededCount
will be filled with the count of the written elements, or needed buffer size.

Param pOutputBuffer
buffer of Buffer to hold the transformed data. When encrypting, this will be the cipher data. When decrypting, this shall be the plain data. If buffer is null, neededBytes return the necessary amount of bytes.

Param pUserData
Userdata that is passed to the callback function

Return
The needed/encrypted/decrypted/verified bytes when succeed or zero when error.

ELib3MFException: The standard exception class of the 3MF Library

Errors in the 3MF Library are reported as Exceptions. It is recommended to not throw these exceptions in your client code.

class `Lib3MF::ELib3MFException`

void `ELib3MFException::what()` const noexcept

Returns error message

Returns
the error message of this exception

Lib3MFResult `ELib3MFException::getErrorCode()` const noexcept

Returns error code

Returns
the error code of this exception

CInputVector: Adapter for passing arrays as input for functions

Several functions of the 3MF Library expect arrays of integral types or structs as input parameters. To not restrict the interface to, say, std::vector<type>, and to have a more abstract interface than a location in memory and the number of elements to input to a function the 3MF Library provides a templated adapter class to pass arrays as input for functions.

Usually, instances of CInputVector are generated anonymously (or even implicitly) in the call to a function that expects an input array.

template<typename **T**>
class `Lib3MF::CInputVector`

CInputVector(const std::vector<*T*> &vec)

Constructs of a CInputVector from a std::vector<T>

CInputVector(const *T* *in_data, size_t in_size)

Constructs of a CInputVector from a memory address and a given number of elements

const *T* **CInputVector*::data*() const

returns the start address of the data captured by this CInputVector

size_t *CInputVector*::**size**() const

returns the number of elements captured by this CInputVector

1.1.3 API-Classes

CAccessRight

class `Lib3MF::CAccessRight` : public *CBase*

PConsumer **GetConsumer**()

Gets the consumer associated with this access right

Returns
The consumer instance

eWrappingAlgorithm **GetWrappingAlgorithm**()

Gets the associated encryption algorithm

Returns
The algorithm used for the key in this accessright

eMgfAlgorithm **GetMgfAlgorithm**()

Gets the associated mask generation function algorithm

Returns
The MFG1 algorithm

eDigestMethod **GetDigestMethod**()

Gets the digest method assoicated

Returns
The digest method for this accessright

typedef std::shared_ptr<*CAccessRight*> `Lib3MF::PAccessRight`

Shared pointer to CAccessRight to easily allow reference counting.

CAttachment

class `Lib3MF::CAttachment` : public *CBase*

std::string **GetPath**()

Retrieves an attachment's package path. This function will be removed in a later release.

Returns
returns the attachment's package path string

void **SetPath**(const std::string &sPath)

Sets an attachment's package path. This function will be removed in a later release.

Parameters
sPath – new path of the attachment.

PPackagePart **PackagePart**()

Returns the PackagePart that is this attachment.

Returns
The PackagePart of this attachment.

std::string **GetRelationShipType**()

Retrieves an attachment's relationship type

Returns
returns the attachment's package relationship type string

void **SetRelationShipType**(const std::string &sPath)

Sets an attachment's relationship type.

Parameters
sPath – new relationship type string.

void **WriteToFile**(const std::string &sFileName)

Writes out the attachment as file.

Parameters
sFileName – file to write into.

void **ReadFromFile**(const std::string &sFileName)

Reads an attachment from a file.

Parameters
sFileName – file to read from.

Lib3MF_uint64 **GetStreamSize**()

Retrieves the size of the attachment stream

Returns
the stream size

void **WriteToBuffer**(std::vector<*Lib3MF_uint8*> &BufferBuffer)

Writes out the attachment into a buffer

Parameters
BufferBuffer – Buffer to write into

void **ReadFromBuffer**(const *CInputVector*<*Lib3MF_uint8*> &BufferBuffer)

Reads an attachment from a memory buffer

Parameters
BufferBuffer – Buffer to read from

typedef std::shared_ptr<*CAttachment*> **Lib3MF::PAttachment**

Shared pointer to CAttachment to easily allow reference counting.

CBase

class **Lib3MF::CBase**

typedef std::shared_ptr<*CBase*> **Lib3MF::PBase**

Shared pointer to CBase to easily allow reference counting.

CBaseMaterialGroup

class **Lib3MF::CBaseMaterialGroup** : public *CResource*

The BaseMaterialGroup corresponds to a basematerials-element within a 3MF document

Lib3MF_uint32 **GetCount**()

Retrieves the count of base materials in the material group.

Returns
returns the count of base materials.

void **GetAllPropertyIDs**(std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

returns all the PropertyIDs of all materials in this group

Parameters
PropertyIDsBuffer – PropertyID of the material in the material group.

Lib3MF_uint32 **AddMaterial**(const std::string &sName, const *sColor* &DisplayColor)

Adds a new material to the material group

Parameters

- **sName** – new name of the base material.
- **DisplayColor** – Display color of the material

Returns
returns new PropertyID of the new material in the material group.

void **RemoveMaterial**(const *Lib3MF_uint32* nPropertyID)

Removes a material from the material group.

Parameters
nPropertyID – PropertyID of the material in the material group.

std::string **GetName**(const *Lib3MF_uint32* nPropertyID)

Returns the base material's name

Parameters
nPropertyID – PropertyID of the material in the material group.

Returns
returns the name of the base material.

void **SetName**(const *Lib3MF_uint32* nPropertyID, const std::string &sName)

Sets a base material's name

Parameters

- **nPropertyID** – PropertyID of the material in the material group.
- **sName** – new name of the base material.

void **SetDisplayColor**(const *Lib3MF_uint32* nPropertyID, const *sColor* &TheColor)

Sets a base material's display color.

Parameters

- **nPropertyID** – PropertyID of the material in the material group.
- **TheColor** – The base material's display color

sColor **GetDisplayColor**(const *Lib3MF_uint32* nPropertyID)

Returns a base material's display color.

Parameters
nPropertyID – PropertyID of the material in the material group.

Returns
The base material's display color

typedef std::shared_ptr<*CBaseMaterialGroup*> **Lib3MF::PBaseMaterialGroup**

Shared pointer to CBaseMaterialGroup to easily allow reference counting.

CBaseMaterialGroupIterator

class **Lib3MF::CBaseMaterialGroupIterator** : public *CResourceIterator*

PBaseMaterialGroup **GetCurrentBaseMaterialGroup**()

Returns the MaterialGroup the iterator points at.

Returns
returns the BaseMaterialGroup instance.

typedef std::shared_ptr<*CBaseMaterialGroupIterator*> **Lib3MF::PBaseMaterialGroupIterator**

Shared pointer to CBaseMaterialGroupIterator to easily allow reference counting.

CBeamLattice

class **Lib3MF::CBeamLattice** : public *CBase*

Lib3MF_double **GetMinLength**()

Returns the minimal length of beams for the beamlattice.

Returns
minimal length of beams for the beamlattice

void **SetMinLength**(const *Lib3MF_double* dMinLength)

Sets the minimal length of beams for the beamlattice.

Parameters
dMinLength – minimal length of beams for the beamlattice

void **GetClipping**(*eBeamLatticeClipMode* &eClipMode, *Lib3MF_uint32* &nUniqueResourceID)

Returns the clipping mode and the clipping-mesh for the beamlattice of this mesh.

Parameters

- **eClipMode** – contains the clip mode of this mesh
- **nUniqueResourceID** – filled with the UniqueResourceID of the clipping mesh-object or an undefined value if pClipMode is MODELBEAMLATTICECLIPMODE_NONE

void **SetClipping**(const *eBeamLatticeClipMode* eClipMode, const *Lib3MF_uint32* nUniqueResourceID)

Sets the clipping mode and the clipping-mesh for the beamlattice of this mesh.

Parameters

- **eClipMode** – contains the clip mode of this mesh
- **nUniqueResourceID** – the UniqueResourceID of the clipping mesh-object. This mesh-object has to be defined before setting the Clipping.

bool **GetRepresentation**(*Lib3MF_uint32* &nUniqueResourceID)

Returns the representation-mesh for the beamlattice of this mesh.

Returns

flag whether the beamlattice has a representation mesh.

Parameters

nUniqueResourceID – filled with the UniqueResourceID of the clipping mesh-object.

void **SetRepresentation**(const *Lib3MF_uint32* nUniqueResourceID)

Sets the representation-mesh for the beamlattice of this mesh.

Parameters

nUniqueResourceID – the UniqueResourceID of the representation mesh-object. This mesh-object has to be defined before setting the representation.

void **GetBallOptions**(*eBeamLatticeBallMode* &eBallMode, *Lib3MF_double* &dBallRadius)

Returns the ball mode and the default ball radius for the beamlattice of this mesh.Returns the ball mode and the default ball radius for the beamlattice of this mesh.

param eBallMode
contains the ball mode of this mesh

param dBallRadius
default ball radius of balls for the beamlattice

typedef std::shared_ptr<*CBeamLattice*> **Lib3MF::PBeamLattice**

Shared pointer to CBeamLattice to easily allow reference counting.

CBeamSet

class **Lib3MF::CBeamSet** : public *CBase*

void **SetName**(const std::string &sName)

Sets a beamset's name string

Parameters

sName – new name of the beamset.

std::string **GetName**()

Retrieves a beamset's name string

Returns
returns the name of the beamset.

void **SetIdentifier**(const std::string &sIdentifier)

Sets a beamset's identifier string

Parameters
sIdentifier – new name of the beamset.

std::string **GetIdentifier**()

Retrieves a beamset's identifier string

Returns
returns the identifier of the beamset.

Lib3MF_uint32 **GetReferenceCount**()

Retrieves the reference count of a beamset

Returns
returns the reference count

void **SetReferences**(const *CInputVector*<*Lib3MF_uint32*> &ReferencesBuffer)

Sets the references of a beamset

Parameters
ReferencesBuffer – the new indices of all beams in this beamset

void **GetReferences**(std::vector<*Lib3MF_uint32*> &ReferencesBuffer)

Retrieves the references of a beamset

Parameters
ReferencesBuffer – retrieves the indices of all beams in this beamset

Lib3MF_uint32 **GetBallReferenceCount**()

Retrieves the ball reference count of a beamset

Returns
returns the ball reference count

void **SetBallReferences**(const *CInputVector*<*Lib3MF_uint32*> &BallReferencesBuffer)

Sets the ball references of a beamset

Parameters
BallReferencesBuffer – the new indices of all balls in this beamset

void **GetBallReferences**(std::vector<*Lib3MF_uint32*> &BallReferencesBuffer)

Retrieves the ball references of a beamset

Parameters
BallReferencesBuffer – retrieves the indices of all balls in this beamset

typedef std::shared_ptr<*CBeamSet*> **Lib3MF::PBeamSet**

Shared pointer to CBeamSet to easily allow reference counting.

CBuildItem

class `Lib3MF::CBuildItem` : public *CBase*

PObject **GetObjectResource**()

Retrieves the object resource associated to a build item

Returns
returns the associated resource instance

std::string **GetUUID**(bool &bHasUUID)

returns, whether a build item has a UUID and, if true, the build item's UUID

Parameters
bHasUUID – flag whether the build item has a UUID

Returns
the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

void **SetUUID**(const std::string &sUUID)

sets the build item's UUID

Parameters
sUUID – the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

Lib3MF_uint32 **GetObjectResourceID**()

Retrieves the object UniqueResourceID associated to a build item

Returns
returns the UniqueResourceID of the object

bool **HasObjectTransform**()

Checks, if a build item has a non-identity transformation matrix

Returns
returns true, if the transformation matrix is not the identity

sTransform **GetObjectTransform**()

Retrieves a build item's transformation matrix.

Returns
returns the transformation matrix

void **SetObjectTransform**(const *sTransform* &Transform)

Sets a build item's transformation matrix.

Parameters
Transform – new transformation matrix

std::string **GetPartNumber**()

Retrieves a build item's part number string

Returns
Returns a build item's part number string

void **SetPartNumber**(const std::string &sSetPartnumber)

Sets a build item's part number string

Parameters
sSetPartnumber – new part number string for referencing parts from the outside world

PMetaDataGroup **GetMetaDataGroup**()

Returns the metadatagroup of this build item

Returns
returns an Instance of the metadatagroup of this build item

sBox **GetOutbox**()

Returns the outbox of a build item

Returns
Outbox of this build item

typedef std::shared_ptr<*CBuildItem*> Lib3MF::**PBuildItem**

Shared pointer to CBuildItem to easily allow reference counting.

CBuildItemIterator

class Lib3MF::**CBuildItemIterator** : public *CBase*

bool **MoveNext**()

Iterates to the next build item in the list.

Returns
Iterates to the next build item in the list.

bool **MovePrevious**()

Iterates to the previous build item in the list.

Returns
Iterates to the previous build item in the list.

PBuildItem **GetCurrent**()

Returns the build item the iterator points at.

Returns
returns the build item instance.

PBuildItemIterator **Clone**()

Creates a new build item iterator with the same build item list.

Returns
returns the cloned Iterator instance

Lib3MF_uint64 **Count**()

Returns the number of build items the iterator captures.

Returns
returns the number of build items the iterator captures.

typedef std::shared_ptr<*CBuildItemIterator*> Lib3MF::**PBuildItemIterator**

Shared pointer to CBuildItemIterator to easily allow reference counting.

CColorGroup

class `Lib3MF::CColorGroup` : public *CResource*

Lib3MF_uint32 **GetCount**()

Retrieves the count of base materials in this Color Group.

Returns
returns the count of colors within this color group.

void **GetAllPropertyIDs**(std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

returns all the PropertyIDs of all colors within this group

Parameters
PropertyIDsBuffer – PropertyID of the color in the color group.

Lib3MF_uint32 **AddColor**(const *sColor* &TheColor)

Adds a new value.

Parameters
TheColor – The new color

Returns
PropertyID of the new color within this color group.

void **RemoveColor**(const *Lib3MF_uint32* nPropertyID)

Removes a color from the color group.

Parameters
nPropertyID – PropertyID of the color to be removed from the color group.

void **SetColor**(const *Lib3MF_uint32* nPropertyID, const *sColor* &TheColor)

Sets a color value.

Parameters

- **nPropertyID** – PropertyID of a color within this color group.
- **TheColor** – The color

sColor **GetColor**(const *Lib3MF_uint32* nPropertyID)

Sets a color value.

Parameters
nPropertyID – PropertyID of a color within this color group.

Returns
The color

typedef std::shared_ptr<*CColorGroup*> `Lib3MF::PColorGroup`

Shared pointer to CColorGroup to easily allow reference counting.

CColorGroupIterator

class `Lib3MF::CColorGroupIterator` : public *CResourceIterator*

PColorGroup **GetCurrentColorGroup**()

Returns the ColorGroup the iterator points at.

Returns
returns the ColorGroup instance.

typedef std::shared_ptr<*CColorGroupIterator*> `Lib3MF::PColorGroupIterator`

Shared pointer to CColorGroupIterator to easily allow reference counting.

CComponent

class `Lib3MF::CComponent` : public *CBase*

PObject **GetObjectResource**()

Returns the Resource Instance of the component.

Returns
filled with the Resource Instance.

Lib3MF_uint32 **GetObjectResourceID**()

Returns the UniqueResourceID of the component.

Returns
returns the UniqueResourceID.

std::string **GetUUID**(bool &bHasUUID)

returns, whether a component has a UUID and, if true, the component's UUID

Parameters
bHasUUID – flag whether the component has a UUID

Returns
the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

void **SetUUID**(const std::string &sUUID)

sets the component's UUID

Parameters
sUUID – the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

bool **HasTransform**()

Returns, if the component has a different transformation than the identity matrix

Returns
if true is returned, the transformation is not equal than the identity

sTransform **GetTransform**()

Returns the transformation matrix of the component.

Returns
filled with the component transformation matrix

void **SetTransform**(const *sTransform* &Transform)

Sets the transformation matrix of the component.

Parameters
Transform – new transformation matrix

typedef std::shared_ptr<*CComponent*> **Lib3MF::PComponent**

Shared pointer to CComponent to easily allow reference counting.

CComponentsObject

class **Lib3MF::CComponentsObject** : public *CObject*

PComponent **AddComponent**(*CObject* *pObjectResource, const *sTransform* &Transform)

Adds a new component to a components object.

Parameters

- **pObjectResource** – object to add as component. Must not lead to circular references!
- **Transform** – optional transform matrix for the component.

Returns
new component instance

PComponent **GetComponent**(const *Lib3MF_uint32* nIndex)

Retrieves a component from a component object.

Parameters
nIndex – index of the component to retrieve (0 to componentcount - 1)

Returns
component instance

Lib3MF_uint32 **GetComponentCount**()

Retrieves a component count of a component object.

Returns
returns the component count

typedef std::shared_ptr<*CComponentsObject*> **Lib3MF::PComponentsObject**

Shared pointer to CComponentsObject to easily allow reference counting.

CComponentsObjectIterator

class **Lib3MF::CComponentsObjectIterator** : public *CResourceIterator*

PComponentsObject **GetCurrentComponentsObject**()

Returns the ComponentsObject the iterator points at.

Returns
returns the ComponentsObject instance.

typedef std::shared_ptr<*CComponentsObjectIterator*> **Lib3MF::PComponentsObjectIterator**

Shared pointer to CComponentsObjectIterator to easily allow reference counting.

CCompositeMaterials

class `Lib3MF::CCompositeMaterials` : public *CResource*

Lib3MF_uint32 **GetCount**()

Retrieves the count of Composite-s in the CompositeMaterials.

Returns
returns the count of Composite-s

void **GetAllPropertyIDs**(std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

returns all the PropertyIDs of all Composite-Mixing Values in this CompositeMaterials

Parameters
PropertyIDsBuffer – PropertyID of the Composite-Mixing Values in the CompositeMaterials.

PBaseMaterialGroup **GetBaseMaterialGroup**()

Obtains the BaseMaterialGroup instance of this CompositeMaterials.

Returns
returns the BaseMaterialGroup instance of this CompositeMaterials

Lib3MF_uint32 **AddComposite**(const *CInputVector*<*sCompositeConstituent*> &CompositeBuffer)

Adds a new Composite-Mixing Values to the CompositeMaterials.

Parameters
CompositeBuffer – The Composite Constituents to be added as composite

Returns
returns new PropertyID of the new Composite in the CompositeMaterials.

void **RemoveComposite**(const *Lib3MF_uint32* nPropertyID)

Removes a Composite-Maxing Ratio from the CompositeMaterials.

Parameters
nPropertyID – PropertyID of the Composite-Mixing Values in the CompositeMaterials to be removed.

void **GetComposite**(const *Lib3MF_uint32* nPropertyID, std::vector<*sCompositeConstituent*> &CompositeBuffer)

Obtains a Composite-Maxing Ratio of this CompositeMaterials.

Parameters

- **nPropertyID** – the PropertyID of the Composite-Maxing Ratio in the CompositeMaterials.
- **CompositeBuffer** – The Composite-Mixing Values with the given PropertyID

typedef std::shared_ptr<*CCompositeMaterials*> `Lib3MF::PCompositeMaterials`

Shared pointer to CCompositeMaterials to easily allow reference counting.

CCompositeMaterialsIterator

class `Lib3MF::CCompositeMaterialsIterator` : public *CResourceIterator*

PCompositeMaterials **GetCurrentCompositeMaterials**()

Returns the CompositeMaterials the iterator points at.

Returns

returns the CompositeMaterials instance.

typedef std::shared_ptr<*CCompositeMaterialsIterator*> `Lib3MF::PCompositeMaterialsIterator`

Shared pointer to CCompositeMaterialsIterator to easily allow reference counting.

CConsumer

class `Lib3MF::CConsumer` : public *CBase*

std::string **GetConsumerID**()

Gets the consumerid

Returns

A unique identifier for the consumers

std::string **GetKeyID**()

Getts the keyid

Returns

The identifier for the key of this consumer

std::string **GetKeyValue**()

Gets the keyvalue associated with this consumer

Returns

The public key, when available, of this consumer

typedef std::shared_ptr<*CConsumer*> `Lib3MF::PConsumer`

Shared pointer to CConsumer to easily allow reference counting.

CContentEncryptionParams

class `Lib3MF::CContentEncryptionParams` : public *CBase*

eEncryptionAlgorithm **GetEncryptionAlgorithm**()

Returns the encryption method to be used in this encryption process

Returns

void **GetKey**(std::vector<*Lib3MF_uint8*> &ByteDataBuffer)

Gets the key for the resource associated

Parameters

ByteDataBuffer – Pointer to a buffer where to place the key.

void **GetInitializationVector**(std::vector<*Lib3MF_uint8*> &ByteDataBuffer)

Gets the IV data

Parameters

ByteDataBuffer – Pointer to a buffer where to place the data.

void **GetAuthenticationTag**(std::vector<*Lib3MF_uint8*> &ByteDataBuffer)

A handler descriptor that uniquely identifies the context of the resource. Each resource will be assigned a different value

Parameters
ByteDataBuffer – Pointer to a buffer where to place the data.

void **SetAuthenticationTag**(const *CInputVector*<*Lib3MF_uint8*> &ByteDataBuffer)

Sets the authentication tag

Parameters
ByteDataBuffer – The authentication tag size

void **GetAdditionalAuthenticationData**(std::vector<*Lib3MF_uint8*> &ByteDataBuffer)

A handler descriptor that uniquely identifies the context of the resource. Each resource will be assigned a different value

Parameters
ByteDataBuffer – Buffer where the data will be placed

Lib3MF_uint64 **GetDescriptor**()

A handler descriptor that uniquely identifies the context of the resource. Each resource will be assigned a different value

Returns

std::string **GetKeyUUID**()

Gets the resourcedatagroup keyuuid

Returns
The resourcedatagroup keyuuid that may be use to reference an external key

typedef std::shared_ptr<*CContentEncryptionParams*> **Lib3MF::PContentEncryptionParams**

Shared pointer to CContentEncryptionParams to easily allow reference counting.

CKeyStore

class **Lib3MF::CKeyStore** : public *CBase*

PConsumer **AddConsumer**(const std::string &sConsumerID, const std::string &sKeyID, const std::string &sKeyValue)

Adds a consumer to the keystore

Parameters

- **sConsumerID** – A unique identifier for the consumer
- **sKeyID** – The id of the key of the consumer
- **sKeyValue** – The public key for this consumer in PEM format

Returns
The consumer instance

Lib3MF_uint64 **GetConsumerCount**()

Gets the number of consumers in the keystore

Returns
The consumer count

PConsumer **GetConsumer**(const *Lib3MF_uint64* nConsumerIndex)

Get a consumer from the keystore

Parameters
nConsumerIndex – The index of the consumer

Returns
The consumer instance

void **RemoveConsumer**(*CConsumer* *pConsumer)

Removes a consumer from the keystore

Parameters
pConsumer – The consumer instance to remove

PConsumer **FindConsumer**(const std::string &sConsumerID)

Finds a consumer by ID

Parameters
sConsumerID – The ID of the consumer

Returns
The consumer instance

Lib3MF_uint64 **GetResourceDataGroupCount**()

Gets the number of resource data group in the keysore

Returns
The number of resource data available

PResourceDataGroup **AddResourceDataGroup**()

Adds a resource data group into the keystore.

Returns
The resource data group instance

PResourceDataGroup **GetResourceDataGroup**(const *Lib3MF_uint64* nResourceDataIndex)

Gets a resource data group

Parameters
nResourceDataIndex – The index of the resource data

Returns
The resource data group instance

void **RemoveResourceDataGroup**(*CResourceDataGroup* *pResourceDataGroup)

Removes a resource data group

Parameters
pResourceDataGroup – The resource data group instance

PResourceDataGroup **FindResourceDataGroup**(*CPackagePart* *pPartPath)

Finds a resource data group that contains a particular resourcedata

Parameters
pPartPath – The target path for the resourcedata hold by the resource data group

Returns
The data resource instance

PResourceData **AddResourceData**(*CResourceDataGroup* *pResourceDataGroup, *CPackagePart* *pPartPath, const eEncryptionAlgorithm eAlgorithm, const *eCompression* eCompression, const *CInputVector*<*Lib3MF_uint8*> &AdditionalAuthenticationDataBuffer)

Add resourcedata to resourcedatagroup element

Parameters

- **pResourceDataGroup** – The resource data group where to add this resource data
- **pPartPath** – The path of the part to be encrypted
- **eAlgorithm** – The encryption algorithm to be used to encrypt this resource
- **eCompression** – Whether compression should be used prior to encryption
- **AdditionalAuthenticationDataBuffer** – Additional data to be encrypted along the contents for better security

Returns
The data resource instance

void **RemoveResourceData**(*CResourceData* *pResourceData)

Removes a resource data

Parameters
pResourceData – The resource data to be removed

PResourceData **FindResourceData**(*CPackagePart* *pResourcePath)

Finds a resource data on this resource group

Parameters
pResourcePath – The target path for the resourcedata

Returns
The resource data instance

Lib3MF_uint64 **GetResourceDataCount**()

Gets the number of resource data in the keysore

Returns
The number of resource data available

PResourceData **GetResourceData**(const *Lib3MF_uint64* nResourceDataIndex)

Gets a resource data

Parameters
nResourceDataIndex – The index of the resource data

Returns
The data resource instance

std::string **GetUUID**(bool &bHasUUID)

Gets the keystore UUID

Parameters
bHasUUID – flag whether the keystore has a UUID

Returns
returns the keystore uuid.

void **SetUUID**(const std::string &sUUID)

Sets the keystore UUID

Parameters
sUUID – The new keystore uuid.

typedef std::shared_ptr<*CKeyStore*> **Lib3MF::PKeyStore**

Shared pointer to CKeyStore to easily allow reference counting.

CMeshObject

class **Lib3MF::CMeshObject** : public *CObject*

Lib3MF_uint32 **GetVertexCount**()

Returns the vertex count of a mesh object.

Returns
filled with the vertex count.

Lib3MF_uint32 **GetTriangleCount**()

Returns the triangle count of a mesh object.

Returns
filled with the triangle count.

sPosition **GetVertex**(const *Lib3MF_uint32* nIndex)

Returns the vertex count of a mesh object.

Parameters
nIndex – Index of the vertex (0 to vertexcount - 1)

Returns
filled with the vertex coordinates.

void **SetVertex**(const *Lib3MF_uint32* nIndex, const *sPosition* &Coordinates)

Sets the coordinates of a single vertex of a mesh object

Parameters

- **nIndex** – Index of the vertex (0 to vertexcount - 1)
- **Coordinates** – contains the vertex coordinates.

Lib3MF_uint32 **AddVertex**(const *sPosition* &Coordinates)

Adds a single vertex to a mesh object

Parameters
Coordinates – contains the vertex coordinates.

Returns
Index of the new vertex

void **GetVertices**(std::vector<*sPosition*> &VerticesBuffer)

Obtains all vertex positions of a mesh object

Parameters
VerticesBuffer – contains the vertex coordinates.

sTriangle **GetTriangle**(const *Lib3MF_uint32* nIndex)

Returns indices of a single triangle of a mesh object.

Parameters
nIndex – Index of the triangle (0 to trianglecount - 1)

Returns
filled with the triangle indices.

void **SetTriangle**(const *Lib3MF_uint32* nIndex, const *sTriangle* &Indices)

Sets the indices of a single triangle of a mesh object.

Parameters

- **nIndex** – Index of the triangle (0 to trianglecount - 1)
- **Indices** – contains the triangle indices.

Lib3MF_uint32 **AddTriangle**(const *sTriangle* &Indices)

Adds a single triangle to a mesh object

Parameters
Indices – contains the triangle indices.

Returns
Index of the new triangle

void **GetTriangleIndices**(std::vector<*sTriangle*> &IndicesBuffer)

Get all triangles of a mesh object

Parameters
IndicesBuffer – contains the triangle indices.

void **SetObjectLevelProperty**(const *Lib3MF_uint32* nUniqueResourceID, const *Lib3MF_uint32* nPropertyID)

Sets the property at the object-level of the mesh object.

Parameters

- **nUniqueResourceID** – the object-level Property UniqueResourceID.
- **nPropertyID** – the object-level PropertyID.

bool **GetObjectLevelProperty**(*Lib3MF_uint32* &nUniqueResourceID, *Lib3MF_uint32* &nPropertyID)

Gets the property at the object-level of the mesh object.

Parameters

- **nUniqueResourceID** – the object-level Property UniqueResourceID.
- **nPropertyID** – the object-level PropertyID.

Returns
Has an object-level property been specified?

void **SetTriangleProperties**(const *Lib3MF_uint32* nIndex, const *sTriangleProperties* &Properties)

Sets the properties of a single triangle of a mesh object.

Parameters

- **nIndex** – Index of the triangle (0 to trianglecount - 1)
- **Properties** – contains the triangle properties.

void **GetTriangleProperties**(const *Lib3MF_uint32* nIndex, *sTriangleProperties* &Property)

Gets the properties of a single triangle of a mesh object.

Parameters

- **nIndex** – Index of the triangle (0 to trianglecount - 1)
- **Property** – returns the triangle properties.

void **SetAllTriangleProperties**(const *CInputVector*<*sTriangleProperties*> &PropertiesArrayBuffer)

Sets the properties of all triangles of a mesh object. Sets the object level property to the first entry of the passed triangle properties, if not yet specified.

Parameters

PropertiesArrayBuffer – contains the triangle properties array. Must have trianglecount elements.

void **GetAllTriangleProperties**(std::vector<*sTriangleProperties*> &PropertiesArrayBuffer)

Gets the properties of all triangles of a mesh object.

Parameters

PropertiesArrayBuffer – returns the triangle properties array. Must have trianglecount elements.

void **ClearAllProperties**()

Clears all properties of this mesh object (triangle and object-level).

void **SetGeometry**(const *CInputVector*<*sPosition*> &VerticesBuffer, const *CInputVector*<*sTriangle*> &IndicesBuffer)

Set all triangles of a mesh object

Parameters

- **VerticesBuffer** – contains the positions.
- **IndicesBuffer** – contains the triangle indices.

bool **IsManifoldAndOriented**()

Retrieves, if an object describes a topologically oriented and manifold mesh, according to the core spec.

Returns

returns, if the object is oriented and manifold.

PBeamLattice **BeamLattice**()

Retrieves the BeamLattice within this MeshObject.

Returns

the BeamLattice within this MeshObject

typedef std::shared_ptr<*CMeshObject*> **Lib3MF::PMeshObject**

Shared pointer to CMeshObject to easily allow reference counting.

CMeshObjectIterator

class `Lib3MF::CMeshObjectIterator` : public *CResourceIterator*

PMeshObject **GetCurrentMeshObject**()

Returns the MeshObject the iterator points at.

Returns
returns the MeshObject instance.

typedef std::shared_ptr<*CMeshObjectIterator*> `Lib3MF::PMeshObjectIterator`

Shared pointer to CMeshObjectIterator to easily allow reference counting.

CMetaData

class `Lib3MF::CMetaData` : public *CBase*

std::string **GetNameSpace**()

returns the namespace URL of the metadata

Returns
the namespace URL of the metadata

void **SetNameSpace**(const std::string &sNameSpace)

sets a new namespace URL of the metadata

Parameters
sNameSpace – the new namespace URL of the metadata

std::string **GetName**()

returns the name of a metadata

Returns
the name of the metadata

void **SetName**(const std::string &sName)

sets a new name of a metadata

Parameters
sName – the new name of the metadata

std::string **GetKey**()

returns the (namespace+name) of a metadata

Returns
the key (namespace+name) of the metadata

bool **GetMustPreserve**()

returns, whether a metadata must be preserved

Returns
returns, whether a metadata must be preserved

void **SetMustPreserve**(const bool bMustPreserve)

sets whether a metadata must be preserved

Parameters
bMustPreserve – a new value whether a metadata must be preserved

std::string **GetType**()

returns the type of a metadata

Returns
the type of the metadata

void **SetType**(const std::string &sType)

sets a new type of a metadata. This must be a simple XML type

Parameters
sType – a new type of the metadata

std::string **GetValue**()

returns the value of the metadata

Returns
the value of the metadata

void **SetValue**(const std::string &sValue)

sets a new value of the metadata

Parameters
sValue – a new value of the metadata

typedef std::shared_ptr<*CMetaData*> **Lib3MF::PMetaData**

Shared pointer to CMetaData to easily allow reference counting.

CMetaDataGroup

class **Lib3MF::CMetaDataGroup** : public *CBase*

Lib3MF_uint32 **GetMetaDataCount**()

returns the number of metadata in this metadatagroup

Returns
returns the number metadata

PMetaData **GetMetaData**(const *Lib3MF_uint32* nIndex)

returns a metadata value within this metadatagroup

Parameters
nIndex – Index of the Metadata.

Returns
an instance of the metadata

PMetaData **GetMetaDataByKey**(const std::string &sNameSpace, const std::string &sName)

returns a metadata value within this metadatagroup

Parameters

- **sNameSpace** – the namespace of the metadata
- **sName** – the name of the Metadata

Returns
an instance of the metadata

void **RemoveMetaDataByIndex**(const *Lib3MF_uint32* nIndex)

removes metadata by index from the model.

Parameters
nIndex – Index of the metadata to remove

void **RemoveMetaData**(*CMetaData* *pTheMetaData)

removes metadata from the model.

Parameters
pTheMetaData – The metadata to remove

PMetaData **AddMetaData**(const std::string &sNameSpace, const std::string &sName, const std::string &sValue, const std::string &sType, const bool bMustPreserve)

adds a new metadata to this metadatagroup

Parameters

- **sNameSpace** – the namespace of the metadata
- **sName** – the name of the metadata
- **sValue** – the value of the metadata
- **sType** – the type of the metadata
- **bMustPreserve** – shuold the metadata be preserved

Returns
a new instance of the metadata

typedef std::shared_ptr<*CMetaDataGroup*> **Lib3MF::PMetaDataGroup**

Shared pointer to CMetaDataGroup to easily allow reference counting.

CModel

class **Lib3MF::CModel** : public *CBase*

PPackagePart **RootModelPart**()

Returns the PackagePart within the OPC package that holds the root model.

Returns
the PackagePart within the OPC package that holds the model-file

PPackagePart **FindOrCreatePackagePart**(const std::string &sAbsolutePath)

Returns a new PackagePart for use within the OPC package.

Parameters
sAbsolutePath – the absolute Path (physical location) within the OPC package

Returns
the new PackagePart within the OPC package

void **SetUnit**(const *eModelUnit* eUnit)

sets the units of a model.

Parameters
eUnit – Unit enum value for the model unit

eModelUnit **GetUnit**()

returns the units of a model.

Returns
Unit enum value for the model unit

std::string **GetLanguage**()

retrieves the language of a model

Returns
language identifier

void **SetLanguage**(const std::string &sLanguage)

sets the language of a model

Parameters
sLanguage – language identifier

PWriter **QueryWriter**(const std::string &sWriterClass)

creates a model writer instance for a specific file type

Parameters
sWriterClass – string identifier for the file type

Returns
string identifier for the file type

PReader **QueryReader**(const std::string &sReaderClass)

creates a model reader instance for a specific file type

Parameters
sReaderClass – string identifier for the file type

Returns
string identifier for the file type

PTexture2D **GetTexture2DByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model texture by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the texture2d instance

ePropertyType **GetPropertyTypeByID**(const *Lib3MF_uint32* nUniqueResourceID)

returns a Property's type

Parameters
nUniqueResourceID – Resource ID of the Property to Query

Returns
returns a Property's type

PBaseMaterialGroup **GetBaseMaterialGroupByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model base material group by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the BaseMaterialGroup instance

PTexture2DGroup **GetTexture2DGroupByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model texture2d group by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the Texture2DGroup instance

PCompositeMaterials **GetCompositeMaterialsByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model CompositeMaterials by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the CompositeMaterials instance

PMultiPropertyGroup **GetMultiPropertyGroupByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model MultiPropertyGroup by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the MultiPropertyGroup instance

PMeshObject **GetMeshObjectByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a mesh object by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the mesh object instance

PComponentsObject **GetComponentsObjectByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a components object by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the components object instance

PColorGroup **GetColorGroupByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model color group by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the ColorGroup instance

PSliceStack **GetSliceStackByID**(const *Lib3MF_uint32* nUniqueResourceID)

finds a model slicestack by its UniqueResourceID

Parameters
nUniqueResourceID – UniqueResourceID

Returns
returns the slicestack instance

std::string **GetBuildUUID**(bool &bHasUUID)

returns, whether a build has a UUID and, if true, the build's UUID

Parameters
bHasUUID – flag whether the build has a UUID

Returns
the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

void **SetBuildUUID**(const std::string &sUUID)

sets the build's UUID

Parameters
sUUID – the UUID as string of the form 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'

PBuildItemIterator **GetBuildItems**()

creates a build item iterator instance with all build items.

Returns
returns the iterator instance.

sBox **GetOutbox**()

Returns the outbox of a Model

Returns
Outbox of this Model

PResourceIterator **GetResources**()

creates a resource iterator instance with all resources.

Returns
returns the iterator instance.

PObjectIterator **GetObjects**()

creates a resource iterator instance with all object resources.

Returns
returns the iterator instance.

PMeshObjectIterator **GetMeshObjects**()

creates a resource iterator instance with all mesh object resources.

Returns
returns the iterator instance.

PComponentsObjectIterator **GetComponentsObjects**()

creates a resource iterator instance with all components object resources.

Returns
returns the iterator instance.

PTexture2DIterator **GetTexture2Ds**()

creates a Texture2DIterator instance with all texture2d resources.

Returns
returns the iterator instance.

PBaseMaterialGroupIterator **GetBaseMaterialGroups**()

creates a BaseMaterialGroupIterator instance with all base material resources.

Returns
returns the iterator instance.

PColorGroupIterator **GetColorGroups**()

creates a ColorGroupIterator instance with all ColorGroup resources.

Returns
returns the iterator instance.

PTexture2DGroupIterator **GetTexture2DGroups**()

creates a Texture2DGroupIterator instance with all base material resources.

Returns
returns the iterator instance.

PCompositeMaterialsIterator **GetCompositeMaterials**()

creates a CompositeMaterialsIterator instance with all CompositeMaterials resources.

Returns
returns the iterator instance.

PMultiPropertyGroupIterator **GetMultiPropertyGroups**()

creates a MultiPropertyGroupsIterator instance with all MultiPropertyGroup resources.

Returns
returns the iterator instance.

PSliceStackIterator **GetSliceStacks**()

creates a resource iterator instance with all slice stack resources.

Returns
returns the iterator instance.

PModel **MergeToModel**()

Merges all components and objects which are referenced by a build item into a mesh. The memory is duplicated and a new model is created.

Returns
returns the merged model instance

PMeshObject **AddMeshObject**()

adds an empty mesh object to the model.

Returns
returns the mesh object instance

PComponentsObject **AddComponentsObject**()

adds an empty component object to the model.

Returns
returns the components object instance

PSliceStack **AddSliceStack**(const *Lib3MF_double* dZBottom)

creates a new model slicestack by its id

Parameters
dZBottom – Bottom Z value of the slicestack

Returns
returns the new slicestack instance

PTexture2D **AddTexture2DFromAttachment**(*CAttachment* *pTextureAttachment)

adds a texture2d resource to the model. Its path is given by that of an existing attachment.

Parameters
pTextureAttachment – attachment containing the image data.

Returns
returns the new texture instance.

PBaseMaterialGroup **AddBaseMaterialGroup**()

adds an empty BaseMaterialGroup resource to the model.

Returns
returns the new base material instance.

PColorGroup **AddColorGroup**()

adds an empty ColorGroup resource to the model.

Returns
returns the new ColorGroup instance.

PTexture2DGroup **AddTexture2DGroup**(*CTexture2D* *pTexture2DInstance)

adds an empty Texture2DGroup resource to the model.

Parameters
pTexture2DInstance – The texture2D instance of the created Texture2DGroup.

Returns
returns the new Texture2DGroup instance.

PCompositeMaterials **AddCompositeMaterials**(*CBaseMaterialGroup* *pBaseMaterialGroupInstance)

adds an empty CompositeMaterials resource to the model.

Parameters
pBaseMaterialGroupInstance – The BaseMaterialGroup instance of the created CompositeMaterials.

Returns
returns the new CompositeMaterials instance.

PMultiPropertyGroup **AddMultiPropertyGroup**()

adds an empty MultiPropertyGroup resource to the model.

Returns
returns the new MultiPropertyGroup instance.

PBuildItem **AddBuildItem**(*CObject* *pObject, const *sTransform* &Transform)

adds a build item to the model.

Parameters

- **pObject** – Object instance.
- **Transform** – Transformation matrix.

Returns
returns the build item instance.

void **RemoveBuildItem**(*CBuildItem* *pBuildItemInstance)

removes a build item from the model

Parameters
pBuildItemInstance – Build item to remove.

PMetaDataGroup **GetMetaDataGroup**()

Returns the metadata of the model as MetaDataGroup

Returns
returns an Instance of the metadatagroup of the model

PAttachment **AddAttachment**(const std::string &sURI, const std::string &sRelationShipType)

adds an attachment stream to the model. The OPC part will be related to the model stream with a certain relationship type.

Parameters

- **sURI** – Path of the attachment
- **sRelationShipType** – Relationship type of the attachment

Returns
Instance of the attachment object

void **RemoveAttachment**(*CAttachment* *pAttachmentInstance)

Removes attachment from the model.

Parameters
pAttachmentInstance – Attachment instance to remove

PAttachment **GetAttachment**(const *Lib3MF_uint32* nIndex)

retrieves an attachment stream object from the model..

Parameters
nIndex – Index of the attachment stream

Returns
Instance of the attachment object

PAttachment **FindAttachment**(const std::string &sURI)

retrieves an attachment stream object from the model.

Parameters
sURI – Path URI in the package

Returns
Instance of the attachment object

Lib3MF_uint32 **GetAttachmentCount**()

retrieves the number of attachments of the model.

Returns
Returns the number of attachments.

bool **HasPackageThumbnailAttachment**()

Retrieve whether the OPC package contains a package thumbnail.

Returns
returns whether the OPC package contains a package thumbnail

PAttachment **CreatePackageThumbnailAttachment**()

Create a new or the existing package thumbnail for the OPC package.

Returns
Instance of a new or the existing thumbnailattachment object.

PAttachment **GetPackageThumbnailAttachment**()

Get the attachment to the OPC package containing the package thumbnail.

Returns
Instance of the thumbnailattachment object or NULL.

void **RemovePackageThumbnailAttachment**()

Remove the attachment to the OPC package containing the package thumbnail.

void **AddCustomContentType**(const std::string &sExtension, const std::string &sContentType)

Adds a new Content Type to the model.

Parameters

- **sExtension** – File Extension
- **sContentType** – Content Type Identifier

void **RemoveCustomContentType**(const std::string &sExtension)

Removes a custom Content Type from the model (UTF8 version).

Parameters
sExtension – File Extension

void **SetRandomNumberCallback**(const *RandomNumberCallback* pTheCallback, const *Lib3MF_pvoid* pUserData)

Sets the random number generator callback for use in the library

Parameters

- **pTheCallback** – The callback used to generate random numbers
- **pUserData** – Userdata to be passed to the callback function

PKeyStore **GetKeyStore**()

Gets the keystore associated with this model

Returns
The package keystore

typedef std::shared_ptr<*CModel*> **Lib3MF::PModel**

Shared pointer to CModel to easily allow reference counting.

CMultiPropertyGroup

class **Lib3MF::CMultiPropertyGroup** : public *CResource*

Lib3MF_uint32 **GetCount**()

Retrieves the count of MultiProperty-s in the MultiPropertyGroup.

Returns
returns the count of MultiProperty-s

void **GetAllPropertyIDs**(std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

returns all the PropertyIDs of all MultiProperty-s in this MultiPropertyGroup

Parameters
PropertyIDsBuffer – PropertyID of the MultiProperty-s in the MultiPropertyGroup.

Lib3MF_uint32 **AddMultiProperty**(const *CInputVector*<*Lib3MF_uint32*> &PropertyIDsBuffer)

Adds a new MultiProperty to the MultiPropertyGroup.

Parameters
PropertyIDsBuffer – The PropertyIDs of the new MultiProperty.

Returns
returns the PropertyID of the new MultiProperty in the MultiPropertyGroup.

void **SetMultiProperty**(const *Lib3MF_uint32* nPropertyID, const *CInputVector*<*Lib3MF_uint32*> &PropertyIDsBuffer)

Sets the PropertyIDs of a MultiProperty.

Parameters

- **nPropertyID** – the PropertyID of the MultiProperty to be changed.
- **PropertyIDsBuffer** – The new PropertyIDs of the MultiProperty

void **GetMultiProperty**(const *Lib3MF_uint32* nPropertyID, std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

Obtains the PropertyIDs of a MultiProperty.

Parameters

- **nPropertyID** – the PropertyID of the MultiProperty to be queried.
- **PropertyIDsBuffer** – The PropertyIDs of the MultiProperty

void **RemoveMultiProperty**(const *Lib3MF_uint32* nPropertyID)

Removes a MultiProperty from this MultiPropertyGroup.

Parameters
nPropertyID – the PropertyID of the MultiProperty to be removed.

Lib3MF_uint32 **GetLayerCount**()

Retrieves the number of layers of this MultiPropertyGroup.

Returns
returns the number of layers

Lib3MF_uint32 **AddLayer**(const *sMultiPropertyLayer* &TheLayer)

Adds a MultiPropertyLayer to this MultiPropertyGroup.

Parameters
TheLayer – The MultiPropertyLayer to add to this MultiPropertyGroup

Returns
returns the index of this MultiPropertyLayer

sMultiPropertyLayer **GetLayer**(const *Lib3MF_uint32* nLayerIndex)

Obtains a MultiPropertyLayer of this MultiPropertyGroup.

Parameters
nLayerIndex – The Index of the MultiPropertyLayer queried

Returns
The MultiPropertyLayer with index LayerIndex within MultiPropertyGroup

void **RemoveLayer**(const *Lib3MF_uint32* nLayerIndex)

Removes a MultiPropertyLayer from this MultiPropertyGroup.

Parameters
nLayerIndex – The Index of the MultiPropertyLayer to be removed

typedef std::shared_ptr<*CMultiPropertyGroup*> **Lib3MF::PMultiPropertyGroup**
Shared pointer to CMultiPropertyGroup to easily allow reference counting.

CMultiPropertyGroupIterator

class **Lib3MF::CMultiPropertyGroupIterator** : public *CResourceIterator*

PMultiPropertyGroup **GetCurrentMultiPropertyGroup**()
Returns the MultiPropertyGroup the iterator points at.

Returns
returns the MultiPropertyGroup instance.

typedef std::shared_ptr<*CMultiPropertyGroupIterator*> **Lib3MF::PMultiPropertyGroupIterator**
Shared pointer to CMultiPropertyGroupIterator to easily allow reference counting.

CObject

class **Lib3MF::CObject** : public *CResource*

eObjectType **GetType**()
Retrieves an object's type

Returns
returns object type enum.

void **SetType**(const *eObjectType* eObjectType)
Sets an object's type

Parameters
eObjectType – object type enum.

std::string **GetName**()
Retrieves an object's name

Returns
returns object name.

void **SetName**(const std::string &sName)
Sets an object's name string

Parameters
sName – new object name.

std::string **GetPartNumber**()
Retrieves an object's part number

Returns
returns object part number.

void **SetPartNumber**(const std::string &sPartNumber)
Sets an objects partnumber string

Parameters
sPartNumber – new object part number.

bool **IsMeshObject**()

Retrieves, if an object is a mesh object

Returns
returns, whether the object is a mesh object

bool **IsComponentsObject**()

Retrieves, if an object is a components object

Returns
returns, whether the object is a components object

bool **IsValid**()

Retrieves, if the object is valid according to the core spec. For mesh objects, we distinguish between the type attribute of the object:In case of object type other, this always means false.In case of object type model or solidsupport, this means, if the mesh suffices all requirements of the core spec chapter 4.1.In case of object type support or surface, this always means true.A component objects is valid if and only if it contains at least one component and all child components are valid objects.

Returns
returns whether the object is a valid object description

void **SetAttachmentAsThumbnail**(*CAttachment* *pAttachment)

Use an existing attachment as thumbnail for this object

Parameters
pAttachment – Instance of a new or the existing thumbnailattachment object.

PAttachment **GetThumbnailAttachment**()

Get the attachment containing the object thumbnail.

Returns
Instance of the thumbnailattachment object or NULL.

void **ClearThumbnailAttachment**()

Clears the attachment. The attachment instance is not removed from the package.

sBox **GetOutbox**()

Returns the outbox of a build item

Returns
Outbox of this build item

std::string **GetUUID**(bool &bHasUUID)

Retrieves an object's uuid string (see production extension specification)

Parameters
bHasUUID – flag whether the build item has a UUID

Returns
returns object uuid.

void **SetUUID**(const std::string &sUUID)

Sets a build object's uuid string (see production extension specification)

Parameters
sUUID – new object uuid string.

PMetaDataGroup **GetMetaDataGroup**()

Returns the metadatagroup of this object

Returns
returns an Instance of the metadatagroup of this object

void **SetSlicesMeshResolution**(const *eSlicesMeshResolution* eMeshResolution)

set the meshresolution of the mesh object

Parameters
eMeshResolution – meshresolution of this object

eSlicesMeshResolution **GetSlicesMeshResolution**()

get the meshresolution of the mesh object

Returns
meshresolution of this object

bool **HasSlices**(const bool bRecursive)

returns whether the Object has a slice stack. If Recursive is true, also checks whether any references object has a slice stack

Parameters
bRecursive – check also all referenced objects?

Returns
does the object have a slice stack?

void **ClearSliceStack**()

unlinks the attached slicestack from this object. If no slice stack is attached, do noting.

PSliceStack **GetSliceStack**()

get the Slicestack attached to the object

Returns
returns the slicestack instance

void **AssignSliceStack**(*CSliceStack* *pSliceStackInstance)

assigns a slicestack to the object

Parameters
pSliceStackInstance – the new slice stack of this Object

typedef std::shared_ptr<*CObject*> Lib3MF::**PObject**

Shared pointer to CObject to easily allow reference counting.

CObjectIterator

class Lib3MF::**CObjectIterator** : public *CResourceIterator*

PObject **GetCurrentObject**()

Returns the Object the iterator points at.

Returns
returns the Object instance.

typedef std::shared_ptr<*CObjectIterator*> Lib3MF::**PObjectIterator**

Shared pointer to CObjectIterator to easily allow reference counting.

CPackagePart

class `Lib3MF::CPackagePart` : public *CBase*

std::string **GetPath**()

Returns the absolute path of this PackagePart.

Returns
Returns the absolute path of this PackagePart

void **SetPath**(const std::string &sPath)

Sets the absolute path of this PackagePart.

Parameters
sPath – Sets the absolute path of this PackagePart.

typedef std::shared_ptr<*CPackagePart*> `Lib3MF::PPackagePart`

Shared pointer to CPackagePart to easily allow reference counting.

CReader

class `Lib3MF::CReader` : public *CBase*

void **ReadFromFile**(const std::string &sFilename)

Reads a model from a file. The file type is specified by the Model Reader class

Parameters
sFilename – Filename to read from

void **ReadFromBuffer**(const *CInputVector*<*Lib3MF_uint8*> &BufferBuffer)

Reads a model from a memory buffer.

Parameters
BufferBuffer – Buffer to read from

void **ReadFromCallback**(const *ReadCallback* pTheReadCallback, const *Lib3MF_uint64* nStreamSize, const *SeekCallback* pTheSeekCallback, const *Lib3MF_pvoid* pUserData)

Reads a model and from the data provided by a callback function

Parameters

- **pTheReadCallback** – Callback to call for reading a data chunk
- **nStreamSize** – number of bytes the callback returns
- **pTheSeekCallback** – Callback to call for seeking in the stream.
- **pUserData** – Userdata that is passed to the callback function

void **SetProgressCallback**(const *ProgressCallback* pProgressCallback, const *Lib3MF_pvoid* pUserData)

Set the progress callback for calls to this writer

Parameters

- **pProgressCallback** – pointer to the callback function.
- **pUserData** – pointer to arbitrary user data that is passed without modification to the callback.

void **AddRelationToRead**(const std::string &sRelationShipType)

Adds a relationship type which shall be read as attachment in memory while loading

Parameters
sRelationShipType – String of the relationship type

void **RemoveRelationToRead**(const std::string &sRelationShipType)

Removes a relationship type which shall be read as attachment in memory while loading

Parameters
sRelationShipType – String of the relationship type

void **SetStrictModeActive**(const bool bStrictModeActive)

Activates (deactivates) the strict mode of the reader.

Parameters
bStrictModeActive – flag whether strict mode is active or not.

bool **GetStrictModeActive**()

Queries whether the strict mode of the reader is active or not

Returns
returns flag whether strict mode is active or not.

std::string **GetWarning**(const *Lib3MF_uint32* nIndex, *Lib3MF_uint32* &nErrorCode)

Returns Warning and Error Information of the read process

Parameters

- **nIndex** – Index of the Warning. Valid values are 0 to WarningCount - 1
- **nErrorCode** – filled with the error code of the warning

Returns
the message of the warning

Lib3MF_uint32 **GetWarningCount**()

Returns Warning and Error Count of the read process

Returns
filled with the count of the occurred warnings.

void **AddKeyWrappingCallback**(const std::string &sConsumerID, const *KeyWrappingCallback* pTheCallback, const *Lib3MF_pvoid* pUserData)

Registers a callback to deal with key wrapping mechanism from keystore

Parameters

- **sConsumerID** – The ConsumerID to register for
- **pTheCallback** – The callback used to decrypt data key
- **pUserData** – Userdata that is passed to the callback function

void **SetContentEncryptionCallback**(const *ContentEncryptionCallback* pTheCallback, const *Lib3MF_pvoid* pUserData)

Registers a callback to deal with encryption of content

Parameters

- **pTheCallback** – The callback used to encrypt content
- **pUserData** – Userdata that is passed to the callback function

typedef std::shared_ptr<*CReader*> **Lib3MF::PReader**

Shared pointer to CReader to easily allow reference counting.

CResource

class **Lib3MF::CResource** : public *CBase*

Lib3MF_uint32 **GetResourceID**()

Retrieves the unique id of this resource within a package. This function will be removed in a later release in favor of GetUniqueResourceID

Returns
Retrieves the unique id of this resource within a package.

Lib3MF_uint32 **GetUniqueResourceID**()

Retrieves the unique id of this resource within a package.

Returns
Retrieves the unique id of this resource within a package.

PPackagePart **PackagePart**()

Returns the PackagePart within which this resource resides

Returns
the PackagePart within which this resource resides.

void **SetPackagePart**(*CPackagePart* *pPackagePart)

Sets the new PackagePart within which this resource resides

Parameters
pPackagePart – the new PackagePart within which this resource resides.

Lib3MF_uint32 **GetModelResourceID**()

Retrieves the id of this resource within a model.

Returns
Retrieves the id of this resource within a model.

typedef std::shared_ptr<*CResource*> **Lib3MF::PResource**

Shared pointer to CResource to easily allow reference counting.

CResourceData

class **Lib3MF::CResourceData** : public *CBase*

PPackagePart **GetPath**()

Gets the encrypted part path

Returns
The part path

eEncryptionAlgorithm **GetEncryptionAlgorithm**()

Gets the encryption algorithm used to encrypt this ResourceData

Returns
The encryption algorithm

eCompression **GetCompression**()

Tells whether this ResourceData is compressed or not

Returns
The compression method

void **GetAdditionalAuthenticationData**(std::vector<*Lib3MF_uint8*> &ByteDataBuffer)

Tells whether this ResourceData is compressed or not

Parameters
ByteDataBuffer – The compression method

typedef std::shared_ptr<*CResourceData*> **Lib3MF::PResourceData**

Shared pointer to CResourceData to easily allow reference counting.

CResourceDataGroup

class **Lib3MF::CResourceDataGroup** : public *CBase*

std::string **GetKeyUUID**()

Sets the resourcedatagroup keyuuid

Returns
The new resourcedatagroup keyuuid.

PAccessRight **AddAccessRight**(*CConsumer* *pConsumer, const *eWrappingAlgorithm* eWrappingAlgorithm, const *eMgfAlgorithm* eMgfAlgorithm, const *eDigestMethod* eDigestMethod)

Add accessright to resourcedatagroup element

Parameters

- **pConsumer** – The Consumer reference
- **eWrappingAlgorithm** – The key wrapping algorithm to be used
- **eMgfAlgorithm** – The mask generation function to be used
- **eDigestMethod** – The digest mechanism to be used

Returns
The acess right instance

PAccessRight **FindAccessRightByConsumer**(*CConsumer* *pConsumer)

Finds the AccessRight associated with a Consumer

Parameters
pConsumer – The Consumer instance

Returns
The AcessRight instance

void **RemoveAccessRight**(*CConsumer* *pConsumer)

Removes access from a Consumer on this resource data group

Parameters
pConsumer – The Consumer instance

typedef std::shared_ptr<*CResourceDataGroup*> **Lib3MF::PResourceDataGroup**

Shared pointer to CResourceDataGroup to easily allow reference counting.

CResourceIterator

class `Lib3MF::CResourceIterator` : public *CBase*

bool **MoveNext**()

Iterates to the next resource in the list.

Returns
Iterates to the next resource in the list.

bool **MovePrevious**()

Iterates to the previous resource in the list.

Returns
Iterates to the previous resource in the list.

PResource **GetCurrent**()

Returns the resource the iterator points at.

Returns
returns the resource instance.

PResourceIterator **Clone**()

Creates a new resource iterator with the same resource list.

Returns
returns the cloned Iterator instance

Lib3MF_uint64 **Count**()

Returns the number of resoucres the iterator captures.

Returns
returns the number of resoucres the iterator captures.

typedef std::shared_ptr<*CResourceIterator*> `Lib3MF::PResourceIterator`

Shared pointer to CResourceIterator to easily allow reference counting.

CSlice

class `Lib3MF::CSlice` : public *CBase*

void **SetVertices**(const *CInputVector*<*sPosition2D*> &VerticesBuffer)

Set all vertices of a slice. All polygons will be cleared.

Parameters
VerticesBuffer – contains the positions.

void **GetVertices**(std::vector<*sPosition2D*> &VerticesBuffer)

Get all vertices of a slice

Parameters
VerticesBuffer – contains the positions.

Lib3MF_uint64 **GetVertexCount**()

Get the number of vertices in a slice

Returns
the number of vertices in the slice

Lib3MF_uint64 **AddPolygon**(const *CInputVector*<*Lib3MF_uint32*> &IndicesBuffer)

Add a new polygon to this slice

Parameters
IndicesBuffer – the new indices of the new polygon

Returns
the index of the new polygon

Lib3MF_uint64 **GetPolygonCount**()

Get the number of polygons in the slice

Returns
the number of polygons in the slice

void **SetPolygonIndices**(const *Lib3MF_uint64* nIndex, const *CInputVector*<*Lib3MF_uint32*> &IndicesBuffer)

Set all indices of a polygon

Parameters

- **nIndex** – the index of the polygon to manipulate
- **IndicesBuffer** – the new indices of the index-th polygon

void **GetPolygonIndices**(const *Lib3MF_uint64* nIndex, std::vector<*Lib3MF_uint32*> &IndicesBuffer)

Get all vertices of a slice

Parameters

- **nIndex** – the index of the polygon to manipulate
- **IndicesBuffer** – the indices of the index-th polygon

Lib3MF_uint64 **GetPolygonIndexCount**(const *Lib3MF_uint64* nIndex)

Get the number of vertices in a slice

Parameters
nIndex – the index of the polygon to manipulate

Returns
the number of indices of the index-th polygon

Lib3MF_double **GetZTop**()

Get the upper Z-Coordinate of this slice.

Returns
the upper Z-Coordinate of this slice

typedef std::shared_ptr<*CSlice*> **Lib3MF::PSlice**

Shared pointer to CSlice to easily allow reference counting.

CSliceStack

class `Lib3MF::CSliceStack` : public *CResource*

Lib3MF_double **GetBottomZ**()

Get the lower Z-Coordinate of the slice stack.

Returns
the lower Z-Coordinate the slice stack

Lib3MF_uint64 **GetSliceCount**()

Returns the number of slices

Returns
the number of slices

PSlice **GetSlice**(const *Lib3MF_uint64* nSliceIndex)

Query a slice from the slice stack

Parameters
nSliceIndex – the index of the slice

Returns
the Slice instance

PSlice **AddSlice**(const *Lib3MF_double* dZTop)

Returns the number of slices

Parameters
dZTop – upper Z coordinate of the slice

Returns
a new Slice instance

Lib3MF_uint64 **GetSliceRefCount**()

Returns the number of slice refs

Returns
the number of slicereferences

void **AddSliceStackReference**(*CSliceStack* *pTheSliceStack)

Adds another existing slicestack as sliceref in this slicestack

Parameters
pTheSliceStack – the slicestack to use as sliceref

PSliceStack **GetSliceStackReference**(const *Lib3MF_uint64* nSliceRefIndex)

Adds another existing slicestack as sliceref in this slicestack

Parameters
nSliceRefIndex – the index of the slice ref

Returns
the slicestack that is used as sliceref

void **CollapseSliceReferences**()

Removes the indirection of slices via slice-refs, i.e. creates the slices of all slice refs of this SliceStack as actual slices of this SliceStack. All previously existing slices or slicerefs will be removed.

void **SetOwnPath**(const std::string &sPath)

Sets the package path where this Slice should be stored. Input an empty string to reset the path

Parameters
sPath – the package path where this Slice should be stored

std::string **GetOwnPath**()

Obtains the package path where this Slice should be stored. Returns an empty string if the slicestack is stored within the root model.

Returns
the package path where this Slice will be stored

typedef std::shared_ptr<*CSliceStack*> **Lib3MF::PSliceStack**

Shared pointer to CSliceStack to easily allow reference counting.

CSliceStackIterator

class **Lib3MF::CSliceStackIterator** : public *CResourceIterator*

PSliceStack **GetCurrentSliceStack**()

Returns the SliceStack the iterator points at.

Returns
returns the SliceStack instance.

typedef std::shared_ptr<*CSliceStackIterator*> **Lib3MF::PSliceStackIterator**

Shared pointer to CSliceStackIterator to easily allow reference counting.

CTexture2D

class **Lib3MF::CTexture2D** : public *CResource*

PAttachment **GetAttachment**()

Retrieves the attachment located at the path of the texture.

Returns
attachment that holds the texture's image information.

void **SetAttachment**(*CAttachment* *pAttachment)

Sets the texture's package path to the path of the attachment.

Parameters
pAttachment – attachment that holds the texture's image information.

eTextureType **GetContentType**()

Retrieves a texture's content type.

Returns
returns content type enum.

void **SetContentType**(const *eTextureType* eContentType)

Retrieves a texture's content type.

Parameters
eContentType – new Content Type

void **GetTileStyleUV**(*eTextureTileStyle* &eTileStyleU, *eTextureTileStyle* &eTileStyleV)

Retrieves a texture's tilestyle type.

Parameters

- **eTileStyleU** – returns tilestyle type enum.
- **eTileStyleV** – returns tilestyle type enum.

void **SetTileStyleUV**(const *eTextureTileStyle* eTileStyleU, const *eTextureTileStyle* eTileStyleV)

Sets a texture's tilestyle type.

Parameters

- **eTileStyleU** – new tilestyle type enum.
- **eTileStyleV** – new tilestyle type enum.

eTextureFilter **GetFilter**()

Retrieves a texture's filter type.

Returns
returns filter type enum.

void **SetFilter**(const *eTextureFilter* eFilter)

Sets a texture's filter type.

Parameters
eFilter – sets new filter type enum.

typedef std::shared_ptr<*CTexture2D*> **Lib3MF::PTexture2D**

Shared pointer to CTexture2D to easily allow reference counting.

CTexture2DGroup

class **Lib3MF::CTexture2DGroup** : public *CResource*

Lib3MF_uint32 **GetCount**()

Retrieves the count of tex2coords in the Texture2DGroup.

Returns
returns the count of tex2coords.

void **GetAllPropertyIDs**(std::vector<*Lib3MF_uint32*> &PropertyIDsBuffer)

returns all the PropertyIDs of all tex2coords in this Texture2DGroup

Parameters
PropertyIDsBuffer – PropertyID of the tex2coords in the Texture2DGroup.

Lib3MF_uint32 **AddTex2Coord**(const *sTex2Coord* &UVCoordinate)

Adds a new tex2coord to the Texture2DGroup

Parameters
UVCoordinate – The u/v-coordinate within the texture, horizontally right/vertically up from the origin in the lower left of the texture.

Returns
returns new PropertyID of the new tex2coord in the Texture2DGroup.

sTex2Coord **GetTex2Coord**(const *Lib3MF_uint32* nPropertyID)

Obtains a tex2coord to the Texture2DGroup

Parameters
nPropertyID – the PropertyID of the tex2coord in the Texture2DGroup.

Returns
The u/v-coordinate within the texture, horizontally right/vertically up from the origin in the lower left of the texture.

void **RemoveTex2Coord**(const *Lib3MF_uint32* nPropertyID)

Removes a tex2coords from the Texture2DGroup.

Parameters
nPropertyID – PropertyID of the tex2coords in the Texture2DGroup.

PTexture2D **GetTexture2D**()

Obtains the texture2D instance of this group.

Returns
the texture2D instance of this group.

typedef std::shared_ptr<*CTexture2DGroup*> **Lib3MF::PTexture2DGroup**

Shared pointer to CTexture2DGroup to easily allow reference counting.

CTexture2DGroupIterator

class **Lib3MF::CTexture2DGroupIterator** : public *CResourceIterator*

PTexture2DGroup **GetCurrentTexture2DGroup**()

Returns the Texture2DGroup the iterator points at.

Returns
returns the Texture2DGroup instance.

typedef std::shared_ptr<*CTexture2DGroupIterator*> **Lib3MF::PTexture2DGroupIterator**

Shared pointer to CTexture2DGroupIterator to easily allow reference counting.

CTexture2DIterator

class **Lib3MF::CTexture2DIterator** : public *CResourceIterator*

PTexture2D **GetCurrentTexture2D**()

Returns the Texture2D the iterator points at.

Returns
returns the Texture2D instance.

typedef std::shared_ptr<*CTexture2DIterator*> **Lib3MF::PTexture2DIterator**

Shared pointer to CTexture2DIterator to easily allow reference counting.

CWriter

class `Lib3MF::CWriter` : public *CBase*

void **WriteToFile**(const std::string &sFilename)

Writes out the model as file. The file type is specified by the Model Writer class.

Parameters
sFilename – Filename to write into

Lib3MF_uint64 **GetStreamSize**()

Retrieves the size of the full 3MF file stream.

Returns
the stream size

void **WriteToBuffer**(std::vector<*Lib3MF_uint8*> &BufferBuffer)

Writes out the 3MF file into a memory buffer

Parameters
BufferBuffer – buffer to write into

void **WriteToCallback**(const *WriteCallback* pTheWriteCallback, const *SeekCallback* pTheSeekCallback, const *Lib3MF_pvoid* pUserData)

Writes out the model and passes the data to a provided callback function. The file type is specified by the Model Writer class.

Parameters

- **pTheWriteCallback** – Callback to call for writing a data chunk
- **pTheSeekCallback** – Callback to call for seeking in the stream
- **pUserData** – Userdata that is passed to the callback function

void **SetProgressCallback**(const *ProgressCallback* pProgressCallback, const *Lib3MF_pvoid* pUserData)

Set the progress callback for calls to this writer

Parameters

- **pProgressCallback** – pointer to the callback function.
- **pUserData** – pointer to arbitrary user data that is passed without modification to the callback.

Lib3MF_uint32 **GetDecimalPrecision**()

Returns the number of digits after the decimal point to be written in each vertex coordinate-value.

Returns
The number of digits to be written in each vertex coordinate-value after the decimal point.

void **SetDecimalPrecision**(const *Lib3MF_uint32* nDecimalPrecision)

Sets the number of digits after the decimal point to be written in each vertex coordinate-value.

Parameters
nDecimalPrecision – The number of digits to be written in each vertex coordinate-value after the decimal point.

void **SetStrictModeActive**(const bool bStrictModeActive)

Activates (deactivates) the strict mode of the reader.

Parameters
bStrictModeActive – flag whether strict mode is active or not.

bool **GetStrictModeActive**()

Queries whether the strict mode of the reader is active or not

Returns
returns flag whether strict mode is active or not.

std::string **GetWarning**(const *Lib3MF_uint32* nIndex, *Lib3MF_uint32* &nErrorCode)

Returns Warning and Error Information of the read process

Parameters

- **nIndex** – Index of the Warning. Valid values are 0 to WarningCount - 1
- **nErrorCode** – filled with the error code of the warning

Returns
the message of the warning

Lib3MF_uint32 **GetWarningCount**()

Returns Warning and Error Count of the read process

Returns
filled with the count of the occurred warnings.

void **AddKeyWrappingCallback**(const std::string &sConsumerID, const *KeyWrappingCallback* pTheCallback, const *Lib3MF_pvoid* pUserData)

Registers a callback to deal with data key encryption/decryption from keystore

Parameters

- **sConsumerID** – The ConsumerID to register for
- **pTheCallback** – The callback to be callede for wrapping and encryption key
- **pUserData** – Userdata that is passed to the callback function

void **SetContentEncryptionCallback**(const *ContentEncryptionCallback* pTheCallback, const *Lib3MF_pvoid* pUserData)

Registers a callback to deal with encryption of content

Parameters

- **pTheCallback** – The callback used to encrypt content
- **pUserData** – Userdata that is passed to the callback function

typedef std::shared_ptr<*CWriter*> Lib3MF::**PWriter**

Shared pointer to CWriter to easily allow reference counting.

1.2 C-language bindings

This space describes the usage of lib3mf in a C host application.

TODO

1.3 Python-language bindings

TODO

1.4 Pascal-language bindings

TODO

1.5 C#-language bindings

This space describes the usage of lib3mf in a C# host application.

TODO

1.6 Golang-language bindings

TODO

1.7 NodeJS-language bindings

TODO

CHAPTER

TWO

OBTAINING LIB3MF

Simply download the precompiled binary SDK https://github.com/3MFConsortium/lib3mf/releases.

CHAPTER

THREE

USING LIB3MF

Allthough the different language bindings are kept as similar as possible, the usage of lib3mf still depends your programming language. You are best-off starting with one of the examples distributed in the SDK (https://github.com/3MFConsortium/lib3mf/releases).

In addition, the home pages for each language binding give detailed instructions on how to use them.

CHAPTER

FOUR

META INFORMATION

source/license

Reporting Bugs

The 3MF Consortium

Specification of the 3MF format

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

INDEX

C

ContentEncryptionCallback (*C++ type*), 15

E

eBeamLatticeBallMode (*C++ enum*), 11
eBeamLatticeBallMode::All (*C++ enumerator*), 11
eBeamLatticeBallMode::eBlendMethod (*C++ enum*), 12
eBeamLatticeBallMode::eBlendMethod::Mix (*C++ enumerator*), 12
eBeamLatticeBallMode::eBlendMethod::Multiply (*C++ enumerator*), 12
eBeamLatticeBallMode::eBlendMethod::NoBlendMethod (*C++ enumerator*), 12
eBeamLatticeBallMode::eCompression (*C++ enum*), 12
eBeamLatticeBallMode::eCompression::Deflate (*C++ enumerator*), 12
eBeamLatticeBallMode::eCompression::NoCompression (*C++ enumerator*), 12
eBeamLatticeBallMode::eDigestMethod (*C++ enum*), 12
eBeamLatticeBallMode::eDigestMethod::SHA1 (*C++ enumerator*), 12
eBeamLatticeBallMode::eDigestMethod::SHA256 (*C++ enumerator*), 12
eBeamLatticeBallMode::eEncryptionAlgorithm (*C++ enum*), 12
eBeamLatticeBallMode::eEncryptionAlgorithm::AES256_GCM (*C++ enumerator*), 12
eBeamLatticeBallMode::eMgfAlgorithm (*C++ enum*), 12
eBeamLatticeBallMode::eMgfAlgorithm::MGF1_SHA1 (*C++ enumerator*), 12
eBeamLatticeBallMode::eMgfAlgorithm::MGF1_SHA224 (*C++ enumerator*), 12
eBeamLatticeBallMode::eMgfAlgorithm::MGF1_SHA256 (*C++ enumerator*), 12
eBeamLatticeBallMode::eMgfAlgorithm::MGF1_SHA384 (*C++ enumerator*), 12
eBeamLatticeBallMode::eMgfAlgorithm::MGF1_SHA512 (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier (*C++ enum*), 11
eBeamLatticeBallMode::eProgressIdentifier::CLEANUP (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::CREATEOPCPACKAG (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::DONE (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::EXTRACTOPCPACKA (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::QUERYCANCELED (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READBUILD (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READCUSTOMATTAC (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READMESH (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READNONROOTMODE (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READRESOURCES (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READROOTMODEL (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READSLICES (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READSTREAM (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::READTEXTURETACH (*C++ enumerator*), 11
eBeamLatticeBallMode::eProgressIdentifier::WRITEATTACHMENT (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITECONTENTTYP (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITEKEYSTORE (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITEMODELSTOST (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITENOBJECTS (*C++ enumerator*), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITENODES (*C++ enumerator*), 12

eBeamLatticeBallMode::eProgressIdentifier::WRITEMODELS (C++ enumerator), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITEROOTMODEL (C++ enumerator), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITETEXTURES (C++ enumerator), 12
eBeamLatticeBallMode::eProgressIdentifier::WRITETRIANGLES (C++ enumerator), 12
eBeamLatticeBallMode::eWrappingAlgorithm (C++ enum), 12
eBeamLatticeBallMode::eWrappingAlgorithm::RSA_OAEP (C++ enumerator), 12
eBeamLatticeBallMode::Mixed (C++ enumerator), 11
eBeamLatticeBallMode::None (C++ enumerator), 11
eBeamLatticeCapMode (C++ enum), 11
eBeamLatticeCapMode::Butt (C++ enumerator), 11
eBeamLatticeCapMode::HemiSphere (C++ enumerator), 11
eBeamLatticeCapMode::Sphere (C++ enumerator), 11
eBeamLatticeClipMode (C++ enum), 11
eBeamLatticeClipMode::Inside (C++ enumerator), 11
eBeamLatticeClipMode::NoClipMode (C++ enumerator), 11
eBeamLatticeClipMode::Outside (C++ enumerator), 11
eModelUnit (C++ enum), 10
eModelUnit::CentiMeter (C++ enumerator), 10
eModelUnit::Foot (C++ enumerator), 10
eModelUnit::Inch (C++ enumerator), 10
eModelUnit::Meter (C++ enumerator), 10
eModelUnit::MicroMeter (C++ enumerator), 10
eModelUnit::MilliMeter (C++ enumerator), 10
eObjectType (C++ enum), 10
eObjectType::Model (C++ enumerator), 10
eObjectType::Other (C++ enumerator), 10
eObjectType::SolidSupport (C++ enumerator), 10
eObjectType::Support (C++ enumerator), 10
ePropertyType (C++ enum), 10
ePropertyType::BaseMaterial (C++ enumerator), 10
ePropertyType::Colors (C++ enumerator), 10
ePropertyType::Composite (C++ enumerator), 10
ePropertyType::Multi (C++ enumerator), 10
ePropertyType::NoPropertyType (C++ enumerator), 10
ePropertyType::TexCoord (C++ enumerator), 10
eSlicesMeshResolution (C++ enum), 10
eSlicesMeshResolution::Fullres (C++ enumerator), 10
eSlicesMeshResolution::Lowres (C++ enumerator), 10
eTextureClipMode (C++ enum), 11
eTextureClipMode::Auto (C++ enumerator), 11
eTextureClipMode::Clamp (C++ enumerator), 11
eTextureClipMode::Mirror (C++ enumerator), 11
eTextureClipMode::NoClipMode (C++ enumerator), 11
eTextureClipMode::Wrap (C++ enumerator), 11
eTextureFilter (C++ enum), 11
eTextureFilter::Auto (C++ enumerator), 11
eTextureFilter::Linear (C++ enumerator), 11
eTextureFilter::Nearest (C++ enumerator), 11
eTileStyle (C++ enum), 10
eTileStyle::Clamp (C++ enumerator), 10
eTileStyle::Mirror (C++ enumerator), 10
eTileStyle::NoTileStyle (C++ enumerator), 11
eTileStyle::Wrap (C++ enumerator), 10
eTextureType (C++ enum), 10
eTextureType::JPEG (C++ enumerator), 10
eTextureType::PNG (C++ enumerator), 10
eTextureType::Unknown (C++ enumerator), 10

K

KeyWrappingCallback (C++ type), 15

L

Lib3MF::CAccessRight (C++ class), 17
Lib3MF::CAccessRight::GetConsumer (C++ function), 17
Lib3MF::CAccessRight::GetDigestMethod (C++ function), 17
Lib3MF::CAccessRight::GetMgfAlgorithm (C++ function), 17
Lib3MF::CAccessRight::GetWrappingAlgorithm (C++ function), 17
Lib3MF::CAttachment (C++ class), 18
Lib3MF::CAttachment::GetPath (C++ function), 18
Lib3MF::CAttachment::GetRelationShipType (C++ function), 18
Lib3MF::CAttachment::GetStreamSize (C++ function), 18
Lib3MF::CAttachment::PackagePart (C++ function), 18
Lib3MF::CAttachment::ReadFromBuffer (C++ function), 18
Lib3MF::CAttachment::ReadFromFile (C++ function), 18
Lib3MF::CAttachment::SetPath (C++ function), 18
Lib3MF::CAttachment::SetRelationShipType (C++ function), 18
Lib3MF::CAttachment::WriteToBuffer (C++ function), 18
Lib3MF::CAttachment::WriteToFile (C++ function), 18
Lib3MF::CBase (C++ class), 19
Lib3MF::CBaseMaterialGroup (C++ class), 19
Lib3MF::CBaseMaterialGroup::AddMaterial (C++ function), 19
Lib3MF::CBaseMaterialGroup::GetAllPropertyIDs (C++ function), 19

Lib3MF::CBaseMaterialGroup::GetCount (C++ function), 19
Lib3MF::CBaseMaterialGroup::GetDisplayColor (C++ function), 20
Lib3MF::CBaseMaterialGroup::GetName (C++ function), 19
Lib3MF::CBaseMaterialGroup::RemoveMaterial (C++ function), 19
Lib3MF::CBaseMaterialGroup::SetDisplayColor (C++ function), 20
Lib3MF::CBaseMaterialGroup::SetName (C++ function), 19
Lib3MF::CBaseMaterialGroupIterator (C++ class), 20
Lib3MF::CBaseMaterialGroupIterator::GetCurrentBaseMaterialGroup (C++ function), 20
Lib3MF::CBeamLattice (C++ class), 20
Lib3MF::CBeamLattice::GetBallOptions (C++ function), 21
Lib3MF::CBeamLattice::GetClipping (C++ function), 20
Lib3MF::CBeamLattice::GetMinLength (C++ function), 20
Lib3MF::CBeamLattice::GetRepresentation (C++ function), 21
Lib3MF::CBeamLattice::SetClipping (C++ function), 21
Lib3MF::CBeamLattice::SetMinLength (C++ function), 20
Lib3MF::CBeamLattice::SetRepresentation (C++ function), 21
Lib3MF::CBeamSet (C++ class), 21
Lib3MF::CBeamSet::GetBallReferenceCount (C++ function), 22
Lib3MF::CBeamSet::GetBallReferences (C++ function), 22
Lib3MF::CBeamSet::GetIdentifier (C++ function), 22
Lib3MF::CBeamSet::GetName (C++ function), 21
Lib3MF::CBeamSet::GetReferenceCount (C++ function), 22
Lib3MF::CBeamSet::GetReferences (C++ function), 22
Lib3MF::CBeamSet::SetBallReferences (C++ function), 22
Lib3MF::CBeamSet::SetIdentifier (C++ function), 22
Lib3MF::CBeamSet::SetName (C++ function), 21
Lib3MF::CBeamSet::SetReferences (C++ function), 22
Lib3MF::CBuildItem (C++ class), 23
Lib3MF::CBuildItem::GetMetaDataGroup (C++ function), 23
Lib3MF::CBuildItem::GetObjectResource (C++ function), 23
Lib3MF::CBuildItem::GetObjectResourceID (C++ function), 23
Lib3MF::CBuildItem::GetObjectTransform (C++ function), 23
Lib3MF::CBuildItem::GetOutbox (C++ function), 24
Lib3MF::CBuildItem::GetPartNumber (C++ function), 23
Lib3MF::CBuildItem::GetUUID (C++ function), 23
Lib3MF::CBuildItem::HasObjectTransform (C++ function), 23
Lib3MF::CBuildItem::SetObjectTransform (C++ function), 23
Lib3MF::CBuildItem::SetPartNumber (C++ function), 23
Lib3MF::CBuildItem::SetUUID (C++ function), 23
Lib3MF::CBuildItemIterator (C++ class), 24
Lib3MF::CBuildItemIterator::Clone (C++ function), 24
Lib3MF::CBuildItemIterator::Count (C++ function), 24
Lib3MF::CBuildItemIterator::GetCurrent (C++ function), 24
Lib3MF::CBuildItemIterator::MoveNext (C++ function), 24
Lib3MF::CBuildItemIterator::MovePrevious (C++ function), 24
Lib3MF::CColorGroup (C++ class), 25
Lib3MF::CColorGroup::AddColor (C++ function), 25
Lib3MF::CColorGroup::GetAllPropertyIDs (C++ function), 25
Lib3MF::CColorGroup::GetColor (C++ function), 25
Lib3MF::CColorGroup::GetCount (C++ function), 25
Lib3MF::CColorGroup::RemoveColor (C++ function), 25
Lib3MF::CColorGroup::SetColor (C++ function), 25
Lib3MF::CColorGroupIterator (C++ class), 26
Lib3MF::CColorGroupIterator::GetCurrentColorGroup (C++ function), 26
Lib3MF::CComponent (C++ class), 26
Lib3MF::CComponent::GetObjectResource (C++ function), 26
Lib3MF::CComponent::GetObjectResourceID (C++ function), 26
Lib3MF::CComponent::GetTransform (C++ function), 26
Lib3MF::CComponent::GetUUID (C++ function), 26
Lib3MF::CComponent::HasTransform (C++ function), 26
Lib3MF::CComponent::SetTransform (C++ function), 26
Lib3MF::CComponent::SetUUID (C++ function), 26
Lib3MF::CComponentsObject (C++ class), 27
Lib3MF::CComponentsObject::AddComponent

(C++ function), 27
Lib3MF::CComponentsObject::GetComponent (C++ function), 27
Lib3MF::CComponentsObject::GetComponentCount (C++ function), 27
Lib3MF::CComponentsObjectIterator (C++ class), 27
Lib3MF::CComponentsObjectIterator::GetCurrentComponentObject (C++ function), 27
Lib3MF::CCompositeMaterials (C++ class), 28
Lib3MF::CCompositeMaterials::AddComposite (C++ function), 28
Lib3MF::CCompositeMaterials::GetAllPropertyIDs (C++ function), 28
Lib3MF::CCompositeMaterials::GetBaseMaterialGroup (C++ function), 28
Lib3MF::CCompositeMaterials::GetComposite (C++ function), 28
Lib3MF::CCompositeMaterials::GetCount (C++ function), 28
Lib3MF::CCompositeMaterials::RemoveComposite (C++ function), 28
Lib3MF::CCompositeMaterialsIterator (C++ class), 29
Lib3MF::CCompositeMaterialsIterator::GetCurrentCompositeMaterials (C++ function), 29
Lib3MF::CConsumer (C++ class), 29
Lib3MF::CConsumer::GetConsumerID (C++ function), 29
Lib3MF::CConsumer::GetKeyID (C++ function), 29
Lib3MF::CConsumer::GetKeyValue (C++ function), 29
Lib3MF::CContentEncryptionParams (C++ class), 29
Lib3MF::CContentEncryptionParams::GetAdditionalAuthenticationData (C++ function), 30
Lib3MF::CContentEncryptionParams::GetAuthenticationTag (C++ function), 29
Lib3MF::CContentEncryptionParams::GetDescriptor (C++ function), 30
Lib3MF::CContentEncryptionParams::GetEncryptionAlgorithm (C++ function), 29
Lib3MF::CContentEncryptionParams::GetInitializationVector (C++ function), 29
Lib3MF::CContentEncryptionParams::GetKey (C++ function), 29
Lib3MF::CContentEncryptionParams::GetKeyUUID (C++ function), 30
Lib3MF::CContentEncryptionParams::SetAuthenticationTag (C++ function), 30
Lib3MF::CInputVector (C++ class), 17
Lib3MF::CInputVector::CInputVector (C++ function), 17
Lib3MF::CInputVector::CInputVector::data (C++ function), 17
Lib3MF::CInputVector::CInputVector::size (C++ function), 17
Lib3MF::CKeyStore (C++ class), 30
Lib3MF::CKeyStore::AddConsumer (C++ function), 30
Lib3MF::CKeyStore::AddResourceData (C++ function), 31
Lib3MF::CKeyStore::AddResourceDataGroup (C++ function), 31
Lib3MF::CKeyStore::FindConsumer (C++ function), 31
Lib3MF::CKeyStore::FindResourceData (C++ function), 32
Lib3MF::CKeyStore::FindResourceDataGroup (C++ function), 31
Lib3MF::CKeyStore::GetConsumer (C++ function), 30
Lib3MF::CKeyStore::GetConsumerCount (C++ function), 30
Lib3MF::CKeyStore::GetResourceData (C++ function), 32
Lib3MF::CKeyStore::GetResourceDataCount (C++ function), 32
Lib3MF::CKeyStore::GetResourceDataGroup (C++ function), 31
Lib3MF::CKeyStore::GetResourceDataGroupCount (C++ function), 31
Lib3MF::CKeyStore::GetUUID (C++ function), 32
Lib3MF::CKeyStore::RemoveConsumer (C++ function), 31
Lib3MF::CKeyStore::RemoveResourceData (C++ function), 32
Lib3MF::CKeyStore::RemoveResourceDataGroup (C++ function), 31
Lib3MF::CKeyStore::SetUUID (C++ function), 32
Lib3MF::CMeshObject (C++ class), 33
Lib3MF::CMeshObject::AddTriangle (C++ function), 34
Lib3MF::CMeshObject::AddVertex (C++ function), 33
Lib3MF::CMeshObject::BeamLattice (C++ function), 35
Lib3MF::CMeshObject::ClearAllProperties (C++ function), 35
Lib3MF::CMeshObject::GetAllTriangleProperties (C++ function), 35
Lib3MF::CMeshObject::GetObjectLevelProperty (C++ function), 34
Lib3MF::CMeshObject::GetTriangle (C++ function), 33
Lib3MF::CMeshObject::GetTriangleCount (C++ function), 33
Lib3MF::CMeshObject::GetTriangleIndices

(C++ function), 34
Lib3MF::CMeshObject::GetTriangleProperties (C++ function), 34
Lib3MF::CMeshObject::GetVertex (C++ function), 33
Lib3MF::CMeshObject::GetVertexCount (C++ function), 33
Lib3MF::CMeshObject::GetVertices (C++ function), 33
Lib3MF::CMeshObject::IsManifoldAndOriented (C++ function), 35
Lib3MF::CMeshObject::SetAllTriangleProperties (C++ function), 35
Lib3MF::CMeshObject::SetGeometry (C++ function), 35
Lib3MF::CMeshObject::SetObjectLevelProperty (C++ function), 34
Lib3MF::CMeshObject::SetTriangle (C++ function), 34
Lib3MF::CMeshObject::SetTriangleProperties (C++ function), 34
Lib3MF::CMeshObject::SetVertex (C++ function), 33
Lib3MF::CMeshObjectIterator (C++ class), 36
Lib3MF::CMeshObjectIterator::GetCurrentMeshObject (C++ function), 36
Lib3MF::CMetaData (C++ class), 36
Lib3MF::CMetaData::GetKey (C++ function), 36
Lib3MF::CMetaData::GetMustPreserve (C++ function), 36
Lib3MF::CMetaData::GetName (C++ function), 36
Lib3MF::CMetaData::GetNameSpace (C++ function), 36
Lib3MF::CMetaData::GetType (C++ function), 36
Lib3MF::CMetaData::GetValue (C++ function), 37
Lib3MF::CMetaData::SetMustPreserve (C++ function), 36
Lib3MF::CMetaData::SetName (C++ function), 36
Lib3MF::CMetaData::SetNameSpace (C++ function), 36
Lib3MF::CMetaData::SetType (C++ function), 37
Lib3MF::CMetaData::SetValue (C++ function), 37
Lib3MF::CMetaDataGroup (C++ class), 37
Lib3MF::CMetaDataGroup::AddMetaData (C++ function), 38
Lib3MF::CMetaDataGroup::GetMetaData (C++ function), 37
Lib3MF::CMetaDataGroup::GetMetaDataByKey (C++ function), 37
Lib3MF::CMetaDataGroup::GetMetaDataCount (C++ function), 37
Lib3MF::CMetaDataGroup::RemoveMetaData (C++ function), 38
Lib3MF::CMetaDataGroup::RemoveMetaDataByIndex (C++ function), 37
Lib3MF::CModel (C++ class), 38
Lib3MF::CModel::AddAttachment (C++ function), 44
Lib3MF::CModel::AddBaseMaterialGroup (C++ function), 43
Lib3MF::CModel::AddBuildItem (C++ function), 43
Lib3MF::CModel::AddColorGroup (C++ function), 43
Lib3MF::CModel::AddComponentsObject (C++ function), 42
Lib3MF::CModel::AddCompositeMaterials (C++ function), 43
Lib3MF::CModel::AddCustomContentType (C++ function), 45
Lib3MF::CModel::AddMeshObject (C++ function), 42
Lib3MF::CModel::AddMultiPropertyGroup (C++ function), 43
Lib3MF::CModel::AddSliceStack (C++ function), 42
Lib3MF::CModel::AddTexture2DFromAttachment (C++ function), 42
Lib3MF::CModel::AddTexture2DGroup (C++ function), 43
Lib3MF::CModel::CreatePackageThumbnailAttachment (C++ function), 44
Lib3MF::CModel::FindAttachment (C++ function), 44
Lib3MF::CModel::FindOrCreatePackagePart (C++ function), 38
Lib3MF::CModel::GetAttachment (C++ function), 44
Lib3MF::CModel::GetAttachmentCount (C++ function), 44
Lib3MF::CModel::GetBaseMaterialGroupByID (C++ function), 39
Lib3MF::CModel::GetBaseMaterialGroups (C++ function), 41
Lib3MF::CModel::GetBuildItems (C++ function), 41
Lib3MF::CModel::GetBuildUUID (C++ function), 40
Lib3MF::CModel::GetColorGroupByID (C++ function), 40
Lib3MF::CModel::GetColorGroups (C++ function), 41
Lib3MF::CModel::GetComponentsObjectByID (C++ function), 40
Lib3MF::CModel::GetComponentsObjects (C++ function), 41
Lib3MF::CModel::GetCompositeMaterials (C++ function), 42
Lib3MF::CModel::GetCompositeMaterialsByID (C++ function), 40
Lib3MF::CModel::GetKeyStore (C++ function), 45
Lib3MF::CModel::GetLanguage (C++ function), 39
Lib3MF::CModel::GetMeshObjectByID (C++ function), 40
Lib3MF::CModel::GetMeshObjects (C++ function), 41

Lib3MF::CModel::GetMetaDataGroup (C++ function), 43
Lib3MF::CModel::GetMultiPropertyGroupByID (C++ function), 40
Lib3MF::CModel::GetMultiPropertyGroups (C++ function), 42
Lib3MF::CModel::GetObjects (C++ function), 41
Lib3MF::CModel::GetOutbox (C++ function), 41
Lib3MF::CModel::GetPackageThumbnailAttachment (C++ function), 44
Lib3MF::CModel::GetPropertyTypeByID (C++ function), 39
Lib3MF::CModel::GetResources (C++ function), 41
Lib3MF::CModel::GetSliceStackByID (C++ function), 40
Lib3MF::CModel::GetSliceStacks (C++ function), 42
Lib3MF::CModel::GetTexture2DByID (C++ function), 39
Lib3MF::CModel::GetTexture2DGroupByID (C++ function), 39
Lib3MF::CModel::GetTexture2DGroups (C++ function), 42
Lib3MF::CModel::GetTexture2Ds (C++ function), 41
Lib3MF::CModel::GetUnit (C++ function), 38
Lib3MF::CModel::HasPackageThumbnailAttachment (C++ function), 44
Lib3MF::CModel::MergeToModel (C++ function), 42
Lib3MF::CModel::QueryReader (C++ function), 39
Lib3MF::CModel::QueryWriter (C++ function), 39
Lib3MF::CModel::RemoveAttachment (C++ function), 44
Lib3MF::CModel::RemoveBuildItem (C++ function), 43
Lib3MF::CModel::RemoveCustomContentType (C++ function), 45
Lib3MF::CModel::RemovePackageThumbnailAttachment (C++ function), 45
Lib3MF::CModel::RootModelPart (C++ function), 38
Lib3MF::CModel::SetBuildUUID (C++ function), 41
Lib3MF::CModel::SetLanguage (C++ function), 39
Lib3MF::CModel::SetRandomNumberCallback (C++ function), 45
Lib3MF::CModel::SetUnit (C++ function), 38
Lib3MF::CMultiPropertyGroup (C++ class), 45
Lib3MF::CMultiPropertyGroup::AddLayer (C++ function), 46
Lib3MF::CMultiPropertyGroup::AddMultiProperty (C++ function), 45
Lib3MF::CMultiPropertyGroup::GetAllPropertyIDs (C++ function), 45
Lib3MF::CMultiPropertyGroup::GetCount (C++ function), 45
Lib3MF::CMultiPropertyGroup::GetLayer (C++ function), 46
Lib3MF::CMultiPropertyGroup::GetLayerCount (C++ function), 46
Lib3MF::CMultiPropertyGroup::GetMultiProperty (C++ function), 46
Lib3MF::CMultiPropertyGroup::RemoveLayer (C++ function), 46
Lib3MF::CMultiPropertyGroup::RemoveMultiProperty (C++ function), 46
Lib3MF::CMultiPropertyGroup::SetMultiProperty (C++ function), 46
Lib3MF::CMultiPropertyGroupIterator (C++ class), 47
Lib3MF::CMultiPropertyGroupIterator::GetCurrentMultiProper (C++ function), 47
Lib3MF::CObject (C++ class), 47
Lib3MF::CObject::AssignSliceStack (C++ function), 49
Lib3MF::CObject::ClearSliceStack (C++ function), 49
Lib3MF::CObject::ClearThumbnailAttachment (C++ function), 48
Lib3MF::CObject::GetMetaDataGroup (C++ function), 48
Lib3MF::CObject::GetName (C++ function), 47
Lib3MF::CObject::GetOutbox (C++ function), 48
Lib3MF::CObject::GetPartNumber (C++ function), 47
Lib3MF::CObject::GetSlicesMeshResolution (C++ function), 49
Lib3MF::CObject::GetSliceStack (C++ function), 49
Lib3MF::CObject::GetThumbnailAttachment (C++ function), 48
Lib3MF::CObject::GetType (C++ function), 47
Lib3MF::CObject::GetUUID (C++ function), 48
Lib3MF::CObject::HasSlices (C++ function), 49
Lib3MF::CObject::IsComponentsObject (C++ function), 48
Lib3MF::CObject::IsMeshObject (C++ function), 47
Lib3MF::CObject::IsValid (C++ function), 48
Lib3MF::CObject::SetAttachmentAsThumbnail (C++ function), 48
Lib3MF::CObject::SetName (C++ function), 47
Lib3MF::CObject::SetPartNumber (C++ function), 47
Lib3MF::CObject::SetSlicesMeshResolution (C++ function), 49
Lib3MF::CObject::SetType (C++ function), 47
Lib3MF::CObject::SetUUID (C++ function), 48
Lib3MF::CObjectIterator (C++ class), 49
Lib3MF::CObjectIterator::GetCurrentObject (C++ function), 49
Lib3MF::CPackagePart (C++ class), 50

Lib3MF::CPackagePart::GetPath (*C++ function*), 50
Lib3MF::CPackagePart::SetPath (*C++ function*), 50
Lib3MF::CReader (*C++ class*), 50
Lib3MF::CReader::AddKeyWrappingCallback (*C++ function*), 51
Lib3MF::CReader::AddRelationToRead (*C++ function*), 50
Lib3MF::CReader::GetStrictModeActive (*C++ function*), 51
Lib3MF::CReader::GetWarning (*C++ function*), 51
Lib3MF::CReader::GetWarningCount (*C++ function*), 51
Lib3MF::CReader::ReadFromBuffer (*C++ function*), 50
Lib3MF::CReader::ReadFromCallback (*C++ function*), 50
Lib3MF::CReader::ReadFromFile (*C++ function*), 50
Lib3MF::CReader::RemoveRelationToRead (*C++ function*), 51
Lib3MF::CReader::SetContentEncryptionCallback (*C++ function*), 51
Lib3MF::CReader::SetProgressCallback (*C++ function*), 50
Lib3MF::CReader::SetStrictModeActive (*C++ function*), 51
Lib3MF::CResource (*C++ class*), 52
Lib3MF::CResource::GetModelResourceID (*C++ function*), 52
Lib3MF::CResource::GetResourceID (*C++ function*), 52
Lib3MF::CResource::GetUniqueResourceID (*C++ function*), 52
Lib3MF::CResource::PackagePart (*C++ function*), 52
Lib3MF::CResource::SetPackagePart (*C++ function*), 52
Lib3MF::CResourceData (*C++ class*), 52
Lib3MF::CResourceData::GetAdditionalAuthenticationData (*C++ function*), 53
Lib3MF::CResourceData::GetCompression (*C++ function*), 52
Lib3MF::CResourceData::GetEncryptionAlgorithm (*C++ function*), 52
Lib3MF::CResourceData::GetPath (*C++ function*), 52
Lib3MF::CResourceDataGroup (*C++ class*), 53
Lib3MF::CResourceDataGroup::AddAccessRight (*C++ function*), 53
Lib3MF::CResourceDataGroup::FindAccessRightByConsumer (*C++ function*), 53
Lib3MF::CResourceDataGroup::GetKeyUUID (*C++ function*), 53
Lib3MF::CResourceDataGroup::RemoveAccessRight (*C++ function*), 53
Lib3MF::CResourceIterator (*C++ class*), 54
Lib3MF::CResourceIterator::Clone (*C++ function*), 54
Lib3MF::CResourceIterator::Count (*C++ function*), 54
Lib3MF::CResourceIterator::GetCurrent (*C++ function*), 54
Lib3MF::CResourceIterator::MoveNext (*C++ function*), 54
Lib3MF::CResourceIterator::MovePrevious (*C++ function*), 54
Lib3MF::CSlice (*C++ class*), 54
Lib3MF::CSlice::AddPolygon (*C++ function*), 54
Lib3MF::CSlice::GetPolygonCount (*C++ function*), 55
Lib3MF::CSlice::GetPolygonIndexCount (*C++ function*), 55
Lib3MF::CSlice::GetPolygonIndices (*C++ function*), 55
Lib3MF::CSlice::GetVertexCount (*C++ function*), 54
Lib3MF::CSlice::GetVertices (*C++ function*), 54
Lib3MF::CSlice::GetZTop (*C++ function*), 55
Lib3MF::CSlice::SetPolygonIndices (*C++ function*), 55
Lib3MF::CSlice::SetVertices (*C++ function*), 54
Lib3MF::CSliceStack (*C++ class*), 56
Lib3MF::CSliceStack::AddSlice (*C++ function*), 56
Lib3MF::CSliceStack::AddSliceStackReference (*C++ function*), 56
Lib3MF::CSliceStack::CollapseSliceReferences (*C++ function*), 56
Lib3MF::CSliceStack::GetBottomZ (*C++ function*), 56
Lib3MF::CSliceStack::GetOwnPath (*C++ function*), 57
Lib3MF::CSliceStack::GetSlice (*C++ function*), 56
Lib3MF::CSliceStack::GetSliceCount (*C++ function*), 56
Lib3MF::CSliceStack::GetSliceRefCount (*C++ function*), 56
Lib3MF::CSliceStack::GetSliceStackReference (*C++ function*), 56
Lib3MF::CSliceStack::SetOwnPath (*C++ function*), 56
Lib3MF::CSliceStackIterator (*C++ class*), 57
Lib3MF::CSliceStackIterator::GetCurrentSliceStack (*C++ function*), 57
Lib3MF::CTexture2D (*C++ class*), 57
Lib3MF::CTexture2D::GetAttachment (*C++ function*), 57
Lib3MF::CTexture2D::GetContentType (*C++ function*), 57
Lib3MF::CTexture2D::GetFilter (*C++ function*), 58

Lib3MF::CTexture2D::GetTileStyleUV (C++ function), 57
Lib3MF::CTexture2D::SetAttachment (C++ function), 57
Lib3MF::CTexture2D::SetContentType (C++ function), 57
Lib3MF::CTexture2D::SetFilter (C++ function), 58
Lib3MF::CTexture2D::SetTileStyleUV (C++ function), 58
Lib3MF::CTexture2DGroup (C++ class), 58
Lib3MF::CTexture2DGroup::AddTex2Coord (C++ function), 58
Lib3MF::CTexture2DGroup::GetAllPropertyIDs (C++ function), 58
Lib3MF::CTexture2DGroup::GetCount (C++ function), 58
Lib3MF::CTexture2DGroup::GetTex2Coord (C++ function), 58
Lib3MF::CTexture2DGroup::GetTexture2D (C++ function), 59
Lib3MF::CTexture2DGroup::RemoveTex2Coord (C++ function), 59
Lib3MF::CTexture2DGroupIterator (C++ class), 59
Lib3MF::CTexture2DGroupIterator::GetCurrentTexture2DGroup (C++ function), 59
Lib3MF::CTexture2DIterator (C++ class), 59
Lib3MF::CTexture2DIterator::GetCurrentTexture2D (C++ function), 59
Lib3MF::CWrapper (C++ class), 6
Lib3MF::CWrapper::Acquire (C++ function), 7
Lib3MF::CWrapper::ColorToFloatRGBA (C++ function), 8
Lib3MF::CWrapper::ColorToRGBA (C++ function), 8
Lib3MF::CWrapper::CreateModel (C++ function), 6
Lib3MF::CWrapper::FloatRGBAToColor (C++ function), 7
Lib3MF::CWrapper::GetBuildInformation (C++ function), 6
Lib3MF::CWrapper::GetIdentityTransform (C++ function), 8
Lib3MF::CWrapper::GetLastError (C++ function), 7
Lib3MF::CWrapper::GetLibraryVersion (C++ function), 6
Lib3MF::CWrapper::GetPrereleaseInformation (C++ function), 6
Lib3MF::CWrapper::GetScaleTransform (C++ function), 8
Lib3MF::CWrapper::GetSpecificationVersion (C++ function), 6
Lib3MF::CWrapper::GetTranslationTransform (C++ function), 9
Lib3MF::CWrapper::GetUniformScaleTransform (C++ function), 8
Lib3MF::CWrapper::Release (C++ function), 7
Lib3MF::CWrapper::RetrieveProgressMessage (C++ function), 7
Lib3MF::CWrapper::RGBAToColor (C++ function), 7
Lib3MF::CWrapper::SetJournal (C++ function), 7
Lib3MF::CWriter (C++ class), 60
Lib3MF::CWriter::AddKeyWrappingCallback (C++ function), 61
Lib3MF::CWriter::GetDecimalPrecision (C++ function), 60
Lib3MF::CWriter::GetStreamSize (C++ function), 60
Lib3MF::CWriter::GetStrictModeActive (C++ function), 61
Lib3MF::CWriter::GetWarning (C++ function), 61
Lib3MF::CWriter::GetWarningCount (C++ function), 61
Lib3MF::CWriter::SetContentEncryptionCallback (C++ function), 61
Lib3MF::CWriter::SetDecimalPrecision (C++ function), 60
Lib3MF::CWriter::SetProgressCallback (C++ function), 60
Lib3MF::CWriter::SetStrictModeActive (C++ function), 60
Lib3MF::CWriter::WriteToBuffer (C++ function), 60
Lib3MF::CWriter::WriteToCallback (C++ function), 60
Lib3MF::CWriter::WriteToFile (C++ function), 60
Lib3MF::ELib3MFException (C++ class), 16
Lib3MF::ELib3MFException::ELib3MFException::getErrorCode (C++ function), 16
Lib3MF::ELib3MFException::ELib3MFException::what (C++ function), 16
Lib3MF::PAccessRight (C++ type), 17
Lib3MF::PAttachment (C++ type), 19
Lib3MF::PBase (C++ type), 19
Lib3MF::PBaseMaterialGroup (C++ type), 20
Lib3MF::PBaseMaterialGroupIterator (C++ type), 20
Lib3MF::PBeamLattice (C++ type), 21
Lib3MF::PBeamSet (C++ type), 22
Lib3MF::PBuildItem (C++ type), 24
Lib3MF::PBuildItemIterator (C++ type), 24
Lib3MF::PColorGroup (C++ type), 25
Lib3MF::PColorGroupIterator (C++ type), 26
Lib3MF::PComponent (C++ type), 27
Lib3MF::PComponentsObject (C++ type), 27
Lib3MF::PComponentsObjectIterator (C++ type), 27
Lib3MF::PCompositeMaterials (C++ type), 28
Lib3MF::PCompositeMaterialsIterator (C++ type), 29
Lib3MF::PConsumer (C++ type), 29

Lib3MF::PContentEncryptionParams (*C++ type*), 30
Lib3MF::PKeyStore (*C++ type*), 33
Lib3MF::PMeshObject (*C++ type*), 35
Lib3MF::PMeshObjectIterator (*C++ type*), 36
Lib3MF::PMetaData (*C++ type*), 37
Lib3MF::PMetaDataGroup (*C++ type*), 38
Lib3MF::PModel (*C++ type*), 45
Lib3MF::PMultiPropertyGroup (*C++ type*), 47
Lib3MF::PMultiPropertyGroupIterator (*C++ type*), 47
Lib3MF::PObject (*C++ type*), 49
Lib3MF::PObjectIterator (*C++ type*), 49
Lib3MF::PPackagePart (*C++ type*), 50
Lib3MF::PReader (*C++ type*), 51
Lib3MF::PResource (*C++ type*), 52
Lib3MF::PResourceData (*C++ type*), 53
Lib3MF::PResourceDataGroup (*C++ type*), 53
Lib3MF::PResourceIterator (*C++ type*), 54
Lib3MF::PSlice (*C++ type*), 55
Lib3MF::PSliceStack (*C++ type*), 57
Lib3MF::PSliceStackIterator (*C++ type*), 57
Lib3MF::PTexture2D (*C++ type*), 58
Lib3MF::PTexture2DGroup (*C++ type*), 59
Lib3MF::PTexture2DGroupIterator (*C++ type*), 59
Lib3MF::PTexture2DIterator (*C++ type*), 59
Lib3MF::PWrapper (*C++ type*), 9
Lib3MF::PWriter (*C++ type*), 61
Lib3MF_double (*C++ type*), 9
Lib3MF_int16 (*C++ type*), 9
Lib3MF_int32 (*C++ type*), 9
Lib3MF_int64 (*C++ type*), 9
Lib3MF_int8 (*C++ type*), 9
Lib3MF_pvoid (*C++ type*), 9
Lib3MF_single (*C++ type*), 9
Lib3MF_uint16 (*C++ type*), 9
Lib3MF_uint32 (*C++ type*), 9
Lib3MF_uint64 (*C++ type*), 9
Lib3MF_uint8 (*C++ type*), 9
Lib3MFResult (*C++ type*), 9

P

ProgressCallback (*C++ type*), 14

R

RandomNumberCallback (*C++ type*), 15
ReadCallback (*C++ type*), 14

S

sBall (*C++ struct*), 14
sBall::m_Index (*C++ member*), 14
sBall::m_Radius (*C++ member*), 14
sBeam (*C++ struct*), 14
sBeam::m_CapModes (*C++ member*), 14
sBeam::m_Indices (*C++ member*), 14
sBeam::m_Radii (*C++ member*), 14
sBox (*C++ struct*), 13
sBox::m_MaxCoordinate (*C++ member*), 13
sBox::m_MinCoordinate (*C++ member*), 13
sColor (*C++ struct*), 13
sColor::m_Alpha (*C++ member*), 13
sColor::m_Blue (*C++ member*), 13
sColor::m_Green (*C++ member*), 13
sColor::m_Red (*C++ member*), 13
sCompositeConstituent (*C++ struct*), 13
sCompositeConstituent::m_MixingRatio (*C++ member*), 13
sCompositeConstituent::m_PropertyID (*C++ member*), 13
SeekCallback (*C++ type*), 14
sMultiPropertyLayer (*C++ struct*), 13
sMultiPropertyLayer::m_ResourceID (*C++ member*), 13
sMultiPropertyLayer::m_TheBlendMethod (*C++ member*), 13
sPosition (*C++ struct*), 13
sPosition2D (*C++ struct*), 13
sPosition2D::m_Coordinates (*C++ member*), 13
sPosition::m_Coordinates (*C++ member*), 13
sTex2Coord (*C++ struct*), 13
sTex2Coord::m_U (*C++ member*), 13
sTex2Coord::m_V (*C++ member*), 13
sTransform (*C++ struct*), 13
sTransform::m_Fields (*C++ member*), 13
sTriangle (*C++ struct*), 13
sTriangle::m_Indices (*C++ member*), 13
sTriangleProperties (*C++ struct*), 13
sTriangleProperties::m_PropertyIDs (*C++ member*), 13
sTriangleProperties::m_ResourceID (*C++ member*), 13

W

WriteCallback (*C++ type*), 14