Welcome to the CINEMA 4D SDK!

There are four components to our SDK:

- C++
  This SDK offers the most powerful toolset. It is the right choice if you need speed, full access to all parts of the application and want to create complex projects.
  For C++ development you need tools like Microsoft Visual Studio (2005 or higher) and Apple Xcode (3.2.3 or higher). Your code needs to be compiled for both OS X and Windows platforms.
  The C++ SDK can be found in C4D's plugins directory. The documentation is available at www.plugincafe.com.

- COFFEE
  COFFEE is C4D's integrated programming/scripting language. COFFEE is an easy-to-use language, but has a much more restricted feature set than C++. COFFEE runs instantaneously on any available platform.
  The COFFEE documentation is available at www.plugincafe.com. No additional tools are needed.

- Python
  Python is C4D's second programming/scripting language. Python offers much more access than COFFEE, but not as much as C++. Python runs instantaneously on any available platform. Python has a lot of internet resources available that make it the first choice for anything database or web-related.
  The Python documentation is available at www.plugincafe.com. No additional tools are needed.

- Melange
  Melange is a C++ library that can be linked to your own application. It allows you to build, load and save C4D files – without the need for a C4D installation.
  If a C4D installation is present you can also render using melange. The rendering will be done in the background as if it was a native part of your application.
  Melange and the Melange documentation are available at www.plugincafe.com.

# C++ Transition to R12

When you first try to compile your existing code for R12 you'll notice lots of compile errors as there have been massive changes to the SDK.

With R12 there was no possibility for old plugins to run without recompiling, as the internal C4D architecture completely changed from single to double precision IEEE floats. So any regular floating point value is now 8 bytes instead of 4 bytes in size.

We took the opportunity to clean up the SDK, structure it better and make it easier and safer to use. That's why you'll notice a lot of renamed symbols that unfortunately require your code to be adapted.

The following guide will help you to quickly make progress. We've divided it into three sections:

***Chapter 1: Getting your code to compile***
This talks about the necessary steps so your code will compile without any errors or warnings.

***Chapter 2: Getting your code to work***
Your code now compiles but there are still a couple things that don't working right. This chapter outlines the pitfalls and problems that may arise.

***Chapter 3: Using new safety features***
C4D offers several new methods and tools to make your code safer. This overview shows how to effectively use it.

# Chapter 1 - Getting your code to compile

## 1.1 Name changes

C4D's SDK has undergone a lot of renamings so that structures and calls are more consistent and carry more logical names.
The following pages list the most common used symbols. We suggest you do a "Find in Files" search with "Match case" and "Match whole word" enabled – that way symbols can be spotted and renamed in an efficient manner.

Try to start out with the following symbols:

| ***Old Name*** | ***New Name*** |
|---|---|
| PluginShader | BaseShader |
| PluginTag | BaseTag |
| PluginVideoPost | BaseVideoPost |
| PluginObject | BaseObject |
| PluginMaterial | BaseMaterial |
| PluginSceneHook | BaseSceneHook |
| PluginSceneLoader | BaseSceneLoader |
| PluginSceneSaver | BaseSceneSaver |
| AlphaBitmap | BaseBitmap |
| LayerSetSelection | LayerSet |
| STRING | CSTRING |
| GetWorldColor | GetViewColor |
| Iadaptive | SPLINEOBJECT_INTERPOLATION_ADAPTIVE |
| C4DPL_BITMAPFILTER | PLUGINTYPE_BITMAPFILTER |
| LockAndWait | Lock |
| LockNoWait | AttemptLock |
| UnLock | Unlock |
| Handle3D | DrawHandle |
| Line3D | DrawLine |
| Circle3D | DrawCircle |
| GetVFlags | GetBuildFlags |
| StringToLong | ToLong |
| StringToReal | ToReal |
| Thermite | SPLINETYPE_BEZIER |
| Tlinear | SPLINETYPE_LINEAR |
| Tbspline | SPLINETYPE_BSPLINE |
| HCLONE_ASPOLY | HIERARCHYCLONEFLAGS_ASPOLY |
| VFLAG_ISOPARM | BUILDFLAGS_ISOPARM |
| IMAGE_NOMEM | IMAGERESULT_OUTOFMEMORY |
| IMAGE_DISKERROR | IMAGERESULT_FILEERROR |
| IMAGE_OK | IMAGERESULT_OK |
| GE_READ | FILEOPEN_READ |
| GE_WRITE | FILEOPEN_WRITE |
| FILE_NODIALOG | FILEDIALOG_NONE |
| FILE_DIALOG | FILEDIALOG_ANY |
| FILE_IGNOREOPEN | FILEDIALOG_IGNOREOPEN |
| GE_MOTOROLA | BYTEORDER_MOTOROLA |
| GE_INTEL | BYTEORDER_INTEL |
| MODE_RGB | COLORMODE_RGB |

| ***Old Name*** | ***New Name*** |
|---|---|
| CHANNEL_BUMP_SUPPORT | SHADERINFO_BUMP_SUPPORT |
| CHANNEL_ALPHA_SUPPORT | SHADERINFO_ALPHA_SUPPORT |
| CHANNEL_TRANSFORM | SHADERINFO_TRANSFORM |
| SHADER_REFLECTANCE | VOLUMEINFO_REFLECTION |
| SHADER_TRANSPARENCY | VOLUMEINFO_TRANSPARENCY |
| SHADER_ALPHA | VOLUMEINFO_ALPHA |
| SHADER_CHANGENORMAL | VOLUMEINFO_CHANGENORMAL |
| SHADER_DISPLACE | VOLUMEINFO_DISPLACEMENT |
| SHADER_VOLUMETRIC | VOLUMEINFO_VOLUMETRIC |
| SHADER_MIPSAT | VOLUMEINFO_MIPSAT |
| SHADER_TRANSFORM | VOLUMEINFO_TRANSFORM |
| LOAD_OK | INITRENDERRESULT_OK |
| LOAD_NOMEM | INITRENDERRESULT_OUTOFMEMORY |
| UNDO_CHANGE | UNDOTYPE_CHANGE |
| UNDO_CHANGE_NOCHILDS | UNDOTYPE_CHANGE_NOCHILDREN |
| UNDO_CHANGE_SMALL | UNDOTYPE_CHANGE_SMALL |
| UNDO_CHANGE_SELECTION | UNDOTYPE_CHANGE_SELECTION |
| UNDO_NEW | UNDOTYPE_NEW |
| UNDO_DELETE | UNDOTYPE_DELETE |
| MOUSEDRAG_NOMOVE | MOUSEDRAGFLAGS_NOMOVE |
| MOUSEDRAG_CONTINUE | MOUSEDRAGRESULT_CONTINUE |
| MODIFY_EDGESELECTION | MODELINGCOMMANDMODE_EDGESELECTION |
| MODIFY_POLYGONSELECTION | MODELINGCOMMANDMODE_POLYGONSELECTION |
| MODIFY_POINTSELECTION | MODELINGCOMMANDMODE_POINTSELECTION |
| MODELINGCOMMANDFLAG_CREATEUNDO | MODELINGCOMMANDFLAGS_CREATEUNDO |
| DA_ONLY_ACTIVE_VIEW | DRAWFLAGS_ONLY_ACTIVE_VIEW |
| DA_NO_THREAD | DRAWFLAGS_NO_THREAD |
| DA_NO_ANIMATION | DRAWFLAGS_NO_ANIMATION |
| DA_NO_EXPRESSIONS | DRAWFLAGS_NO_EXPRESSIONS |
| DA_FORCEFULLREDRAW | DRAWFLAGS_FORCEFULLREDRAW |
| VP_RENDER | VIDEOPOSTCALL_RENDER |
| VP_INNER | VIDEOPOSTCALL_INNER |
| St7bit | STRINGENCODING_7BIT |
| StXbit | STRINGENCODING_XBIT |
| DIRTY_DATA | DIRTYFLAGS_DATA |
| DIRTY_MATRIX | DIRTYFLAGS_MATRIX |
| COPY_NO_HIERARCHY | COPYFLAGS_NO_HIERARCHY |
| COPY_NO_ANIMATION | COPYFLAGS_NO_ANIMATION |
| COPY_NO_BITS | COPYFLAGS_NO_BITS |
| MPThread | MPThreadPool |
| GetPos | GetRelPos (*) |
| SetPos | SetRelPos (*) |
| GetScale | GetRelScale (*) |
| SetScale | SetRelScale (*) |
| GetRot | GetRelRot (*) |
| SetRot | SetRelRot (*) |

(*) more on this later – you need to make a choice here between the Rel or Abs version. To get your code to compile you can temporarily pick the Rel version, but later need to check which is the correct one to fully support the new “frozen transformation” feature of C4D

### 1.2 Plugin registration

In the past most registration functions were available with identical name, but 2-3 different parameter lists. This has now been consolidated – let's take a look e.g. at RegisterObjectPlugin:

***Old***

```
Bool RegisterObjectPlugin(LONG id, const String &str, LONG info,
DataAllocator *g, const String &description, BaseBitmap *icon, LONG
disklevel, void *emulation=NULL);

Bool RegisterObjectPlugin(LONG id, const String &str, LONG info,
DataAllocator *g, const String &description, String icon, LONG
disklevel);

Bool RegisterObjectPlugin(LONG id, const String &str, LONG info,
DataAllocator *g, const String &description, LONG disklevel);
```

***New***

```
Bool RegisterObjectPlugin(LONG id, const String &str, LONG info,
DataAllocator *g, const String &description, BaseBitmap *icon, LONG
disklevel);
```

To load icons for registration from disk use `AutoBitmap("mybitmap.xxx")`, e.g.

```
result =
RegisterTagPlugin(ID_HAIR_STYLING_EXAMPLE,GeLoadString(IDS_HAIR_STYLING_
EXAMPLE), TAG_MULTIPLE|TAG_VISIBLE, HairStylingTag::Alloc,
"Thairsdkstyling", AutoBitmap("hairstyling.tif"),0);
```

Please keep in mind though that once you have lots of icons this is highly inefficient and increases C4D's startup times. In that case just create a single bitmap containing all images at once and register those using

```
AutoAlloc<BaseBitmap> single_bitmap;
if (!single_bitmap || single_bitmap->Init(bitmapfilepath)!=IMAGERESULT_OK)
  return FALSE;
RegisterIcon(plugin_id_1,single_bitmap,x,y,w,h,ICONFLAG_COPY);
RegisterIcon(plugin_id_2,single_bitmap,x,y,w,h,ICONFLAG_COPY);
…
```

### 1.3 Constants and Flags

C4D's SDK now uses enumerations instead of defines whereever possible. As the C++ 0x standard is not widely recognized we've introduced our own typechecks to make sure that no erroneous constants are passed to any function.

Lets' take e.g. a look at the enumeration `MOUSEDRAGFLAGS`:

***Old***

```
#define MOUSEDRAG_DONTHIDEMOUSE           (1<<0)
#define MOUSEDRAG_NOMOVE                  (1<<1)
#define MOUSEDRAG_EVERYPACKET             (1<<2)
#define MOUSEDRAG_COMPENSATEVIEWPORTORG   (1<<3)
```

***New***

```
enum MOUSEDRAGFLAGS
{
  MOUSEDRAGFLAGS_0                        = 0,
  MOUSEDRAGFLAGS_DONTHIDEMOUSE            = (1<<0),
  MOUSEDRAGFLAGS_NOMOVE                   = (1<<1),
  MOUSEDRAGFLAGS_EVERYPACKET              = (1<<2),
  MOUSEDRAGFLAGS_COMPENSATEVIEWPORTORG    = (1<<3),
  MOUSEDRAGFLAGS_AIRBRUSH                 = (1<<4)
} ENUM_END_FLAGS(MOUSEDRAGFLAGS);
```

Now, if you e.g. call

```
void MouseDragStart(LONG button,Real mx,Real my,MOUSEDRAGFLAGS flag);
```

the following code will result in a compile error:

```
MouseDragStart(0,0.0,0.0,0);
```

C4D expects you to pass a proper member of the flag set – so the correct code would be:

```
MouseDragStart(0,0.0,0.0,MOUSEDRAGFLAGS_0);
```

Any flag set always contains the null element which gets the name of the flag set with the suffix "_0".

The compiler will also force you to change the following code:

```
LONG myflags = MOUSEDRAGFLAGS_DONTHIDEMOUSE|(1<<0);
```

Instead you need to write:

```
MOUSEDRAGFLAGS myflags =
  MOUSEDRAGFLAGS_DONTHIDEMOUSE|MOUSEDRAGFLAGS_NOMOVE;
```

If you encounter compile errors quickly look up the original definition of a function, constant or flag set and check if the name has changed or if it now needs an enumeration instead of a LONG value.

### 1.4 Member functions

Unfortunately not all compilers detect changes in virtual overrides (and there have been quite a few). This means that your code might not be called without you noticing that there was a change. We have enabled errors/warnings whereever possible, but only some compilers support this properly. To make sure that there are no problems please check for the following occurences in your code:

*Any classes derived from NodeData (Objects, Tags, SceneHooks etc.):*

```
(virtual) Bool CopyTo(NodeData *dest, GeListNode *snode, GeListNode
*dnode, LONG flags, AliasTrans *trn);

must be changed to

(virtual) Bool CopyTo(NodeData *dest, GeListNode *snode, GeListNode
*dnode, COPYFLAGS flags, AliasTrans *trn);
```

```
(virtual) LONG GetBranchInfo(GeListNode *node, BranchInfo *info, LONG max,
ULONG flags);

must be changed to

(virtual) LONG GetBranchInfo(GeListNode *node, BranchInfo *info, LONG max,
GETBRANCHINFO flags);
```

```
(virtual) Bool GetDDescription(GeListNode *node, Description
*description,LONG &flags);

must be changed to

(virtual) Bool GetDDescription(GeListNode *node, Description *description,
DESCFLAGS_DESC &flags);
```

```
(virtual) Bool GetDParameter(GeListNode *node, const DescID &id,GeData
&t_data,LONG &flags);

must be changed to

(virtual) Bool GetDParameter(GeListNode *node, const DescID &id,GeData
&t_data,DESCFLAGS_GET &flags);
```

```
(virtual) Bool SetDParameter(GeListNode *node, const DescID &id,const
GeData &t_data,LONG &flags);

must be changed to

(virtual) Bool SetDParameter(GeListNode *node, const DescID &id,const
GeData &t_data,DESCFLAGS_SET &flags);
```

```
(virtual) Bool GetDEnabling(GeListNode *node, const DescID &id,const
GeData &t_data,LONG flags,const BaseContainer *itemdesc);

must be changed to

(virtual) Bool GetDEnabling(GeListNode *node, const DescID &id,const
GeData &t_data,DESCFLAGS_ENABLE flags,const BaseContainer *itemdesc);
```

*Any classes derived from ObjectData / EffectorData:*

```
(virtual) Bool Draw(BaseObject *op, LONG type, BaseDraw *bd, BaseDrawHelp
*bh);

must be changed to

(virtual) DRAWRESULT Draw(BaseObject *op, DRAWPASS type, BaseDraw *bd,
BaseDrawHelp *bh);
```

```
(virtual) LONG DetectHandle(BaseObject *op, BaseDraw *bd, LONG x, LONG y,
LONG qualifier);

must be changed to

(virtual) LONG DetectHandle(BaseObject *op, BaseDraw *bd, LONG x, LONG y,
QUALIFIER qualifier);
```

```
(virtual) Bool MoveHandle(BaseObject *op, BaseObject *undo, const Matrix
&tm, LONG hit_id, LONG qualifier);

must be changed to

(virtual) Bool MoveHandle(BaseObject *op, BaseObject *undo, const Matrix
&tm, LONG hit_id, QUALIFIER qualifier);
```

```
(virtual) LONG Execute(BaseObject *op, BaseDocument *doc, BaseThread *bt,
LONG priority, LONG flags);

must be changed to

(virtual) EXECUTIONRESULT Execute(BaseObject *op, BaseDocument *doc,
BaseThread *bt, LONG priority, EXECUTIONFLAGS flags);
```

*Any classes derived from TagData:*

```
(virtual) LONG Execute(BaseTag *tag, BaseDocument *doc, BaseObject *op,
BaseThread *bt, LONG priority, LONG flags);

must be changed to

(virtual) EXECUTIONRESULT Execute(BaseTag *tag, BaseDocument *doc,
BaseObject *op, BaseThread *bt, LONG priority, EXECUTIONFLAGS flags);
```

*Any classes derived from CustomDataTypeClass:*

```
(virtual) Bool _GetDescription(const CustomDataType *data,Description
&desc,LONG &flags,const BaseContainer &parentdescription,DescID
*singledescid)

must be changed to

(virtual) Bool _GetDescription(const CustomDataType *data,Description
&desc,DESCFLAGS_DESC &flags,const BaseContainer &parentdescription,DescID
*singledescid)
```

*Any classes derived from CTrackData:*

```
(virtual) Bool KeyGetDDescription(CTrack *track, CKey *node, Description
*description,LONG &flags);

must be changed to

(virtual) Bool KeyGetDDescription(CTrack *track, CKey *node, Description
*description,DESCFLAGS_DESC &flags);
```

```
(virtual) Bool KeyGetDEnabling(CTrack *track, CKey *node, const DescID
&id,const GeData &t_data,LONG flags,const BaseContainer *itemdesc);

must be changed to

(virtual) Bool KeyGetDEnabling(CTrack *track, CKey *node, const DescID
&id,const GeData &t_data,DESCFLAGS_ENABLE flags,const BaseContainer
*itemdesc);
```

```
(virtual) Bool KeyGetDParameter(CTrack *track, CKey *node, const DescID
&id,GeData &t_data,LONG &flags);

must be changed to

(virtual) Bool KeyGetDParameter(CTrack *track, CKey *node, const DescID
&id,GeData &t_data,DESCFLAGS_GET &flags);
```

```
(virtual) Bool KeySetDParameter(CTrack *track, CKey *node, const DescID
&id,const GeData &t_data,LONG &flags);

must be changed to

(virtual) Bool KeySetDParameter(CTrack *track, CKey *node, const DescID
&id,const GeData &t_data,DESCFLAGS_SET &flags);
```

*Any classes derived from BitmapLoaderData:*

```
(virtual) LONG Load(const Filename &name, BaseBitmap *bm, LONG frame);

must be changed to

(virtual) IMAGERESULT Load(const Filename &name, BaseBitmap *bm, LONG
frame);
```

*(more changes in members LoadAnimated and ExtractSound)*

*Any classes derived from BitmapSaverData:*

```
(virtual) LONG Save(const Filename &name, BaseBitmap *bm, BaseContainer
*data, LONG savebits);

must be changed to

(virtual) IMAGERESULT Save(const Filename &name, BaseBitmap *bm,
BaseContainer *data, SAVEBIT savebits);
```

*(more changes in members Open, Write and AddSound)*

*Any classes derived from ShaderData:*

```
(virtual) LONG InitRender(BaseShader *chn, InitRenderStruct *irs);

must be changed to

(virtual) INITRENDERRESULT InitRender(BaseShader *sh, const
InitRenderStruct &irs);
```

```
(virtual) LONG GetRenderInfo(BaseShader *sh);

must be changed to

(virtual) SHADERINFO GetRenderInfo(BaseShader *sh);
```

*Any classes derived from VideoPostData:*

```
(virtual) LONG GetRenderInfo(BaseVideoPost *node);

must be changed to

(virtual) VIDEOPOSTINFO GetRenderInfo(BaseVideoPost *node);
```

```
(virtual) LONG Execute(BaseVideoPost *node, VideoPostStruct *vps);

must be changed to

(virtual) RENDERRESULT Execute(BaseVideoPost *node, VideoPostStruct *vps);
```

*(more changes in member GetGlInfo)*

*Any classes derived from MaterialData:*

```
(virtual) LONG GetRenderInfo(BaseMaterial *node);

must be changed to

(virtual) VOLUMEINFO GetRenderInfo(BaseMaterial *node);
```

```
(virtual) LONG InitRender(BaseMaterial *mat, InitRenderStruct *irs);

must be changed to

(virtual) INITRENDERRESULT InitRender(BaseMaterial *mat, const
InitRenderStruct &irs);
```

*(more changes in members InitCalculation and InitGLImage)*

*Any classes derived from ToolData:*

```
(virtual) LONG Draw(BaseDocument *doc, BaseContainer &data, BaseDraw *bd,
BaseDrawHelp *bh, BaseThread *bt,LONG flags);

must be changed to

(virtual) TOOLDRAW Draw(BaseDocument *doc, BaseContainer &data, BaseDraw
*bd, BaseDrawHelp *bh, BaseThread *bt,TOOLDRAWFLAGS flags);
```

*(more changes in member InitDisplayControl)*

*Any classes derived from SceneLoaderData:*

```
(virtual) LONG Load(BaseSceneLoader *node, const Filename &name,
BaseDocument *doc, LONG filterflags, String *error, BaseThread *thread);

must be changed to

(virtual) FILEERROR Load(BaseSceneLoader *node, const Filename &name,
BaseDocument *doc, SCENEFILTER filterflags);
```

*Any classes derived from SceneSaverData:*

```
(virtual) LONG Save(BaseSceneSaver *node, const Filename &name,
BaseDocument *doc, LONG filterflags, String *error, BaseThread *thread);

must be changed to

(virtual) FILEERROR Save(BaseSceneSaver *node, const Filename &name,
BaseDocument *doc, SCENEFILTER filterflags);
```

*Any classes derived from SceneHookData:*

```
(virtual) LONG Execute(BaseObject* op, BaseDocument* doc, BaseThread* bt,
LONG priority, LONG flags);

must be changed to

(virtual) EXECUTIONRESULT Execute(BaseObject *op, BaseDocument *doc,
BaseThread *bt, LONG priority, EXECUTIONFLAGS flags);
```

```
(virtual) LONG InitSceneHook  (BaseSceneHook *node, BaseDocument *doc,
BaseThread *bt);

must be changed to

(virtual) EXECUTIONRESULT InitSceneHook(BaseSceneHook *node, BaseDocument
*doc, BaseThread *bt);
```

*Any classes derived from FalloffData:*

```
(virtual) Bool Draw(const FalloffDataData &data, LONG drawpass, BaseDraw
*bd, BaseDrawHelp *bh);

must be changed to

(virtual) DRAWRESULT Draw(const FalloffDataData &data, DRAWPASS type,
BaseDraw *bd, BaseDrawHelp *bh);
```

### 1.5 Floating point numbers

The following code won't compile anymore:

```
Real r=5.0;
r=FCut(r,0.0f,8.0f);
```

As C4D now runs with double precision part the compiler will report "2 overloads have similar conversions".
This is correct as part of this statement is double precision (the variable r) and part is single precision (0.0f and 8.0f). Try to remove any trailing 'f''s from your floating-point numbers and use double precision constants instead:

```
Real r=5;
r=FCut(r,0.0,8.0);
```

As a benefit your code will actually run faster as conversions from double to single or vice versa can be costly performance-wise.

The old macros SV() / LV() for vector precision conversion and SM() / LM() for matrix precision conversion have been changed to member functions.

***Old***

```
LVector v;
SVector sv = SV(v);
```

***New***

```
LVector v;
SVector sv = v.ToSV();
```

Any vector (`SVector/LVector/Vector`) now has the members

`ToSV` (converts to type `SVector`)
`ToRV` (converts to type `Vector`)
`ToLV` (converts to type `LVector`)

and the same for any matrix

`ToSM` (converts to type `SMatrix`)
`ToRM` (converts to type `Matrix`)
`ToLM` (converts to type `LMatrix`)

### 1.6 Variable Tags

`GetDataAddressR` and `GetDataAddressW` are no longer members of VariableTags. Instead you have to retrieve the correct type of tag, e.g. `PointTag` or `VertexMapTag`. Those tags contain the forementioned member functions, but already with the correct return type – e.g. `Vector*` for the `PointTag` or `SReal*` for `VertexMapTag`

***Old***

```
pVTag=(VariableTag*)bc->GetLink(HAIR_DEFORMER_COMB_Y,doc,Tvertexmap);
if (pVTag && pVTag->GetObject()==rData.pObject)
  pCombY=(Real*)pVTag->GetDataAddressR();
```

***New***

```
VTag=(VertexMapTag*)bc->GetLink(HAIR_DEFORMER_COMB_Y,doc,Tvertexmap);
if (pVTag && pVTag->GetObject()==rData.pObject)
  pCombY=pVTag->GetDataAddressR();
```

Please notice that the red marked typecast is no longer necessary. You should remove all those typecasts from your code – as the compiler will automatically notify you of severe mistakes.
In this example the old code casted the result to `Real*`, which in R11.5 was defined as a single precision floating-point number. Now, in R12 `Real` is defined as double precision floating-point number, but `GetDataAddressR()` returns `SReal*`. If this is not corrected, your code will crash instantly and overwrite memory.

### 1.7 Frequent Changes

*GeDialog*

`GeDialog::Open` now needs to be passed `DLG_TYPE` instead of type `Bool` as first parameter.

`Open(FALSE,…)` becomes `Open(DLG_TYPE_MODAL,…)`

and

`Open(TRUE,…)` becomes `Open(DLG_TYPE_ASYNC,…)`

*BaseThread*

BaseThread::Start now needs to be passed `THREADMODE` instead of type `Bool`.

Start(FALSE) becomes Start(THREADMODE_SYNCHRONOUS)

And

Start(TRUE) becomes Start(THREADMODE_ASYNC)

*UVWTag*

`Cpy` was renamed to `Copy` and `Get`, `Set` and `Copy` all need the datatype `UVWHandle` instead of `void*`

*BaseBitmap*

The former members `GetLine` and `SetLine` have been discontinued. Please use `GetPixelCnt` and `SetPixelCnt` instead.

### 1.8 Nearly there…

Congratulations! Your plugin code should now soon compile without any errors or warnings.

For any more errors or warnings that occur please always look up the function or member definition and adapt the remaining places.

Please **do not ignore** any warnings – clean code does not produce any warnings and ignoring warnings sooner or later will lead to instability and unexpected behaviour!

Btw. code completion tools e.g. like VisualAssist X are an awesome help in this process as they instantly highlight and show the correct definitions and all possible members of a class or enumeration.

# Chapter 2 - Getting your code to work

### 2.1 Double Precision

With R12 C4D made the transition to double precision floating-point numbers, reducing the amount of numerical problems. While C4D's render engine always used double precision since its beginning the rest of the application now completely runs on double precision too. Due to this change the file format is no longer downward compatible – so R11.5 or lower can't read R12 files anymore.

For the highest possible portability use the datatypes `Real`, `Vector` and `Matrix`. Those datatypes internally can either be single or double precision. Starting with R12 they default to double precision, but in case there may be a new architecture or platform they still *could* be defined as single precision. As long as you rely on C4D's datatypes you don't have to deal with the precision issue though, just keep in mind that `Real`/`Vector`/`Matrix` are not guaranteed to be a certain size.

If you explicitely need a certain type of precision use the datatypes `SReal`, `SVector` and `SMatrix` for single precision or `LReal`, `LVector` and `LMatrix` for double precision.

Here the current definitions:

Datatype `Real` is 8 bytes wide (defined as ‚double')
Datatype `Vector` is 24 bytes wide
Datatype `Matrix` is 96 bytes wide

Each datatype also exists as explicit Short or Long version:

`SReal` (4 bytes), `Real` (8 bytes), `LReal` (8 bytes)
`SVector` (12 bytes), `Vector` (24 bytes), `LVector` (24 bytes)
`SMatrix` (48 bytes), `Matrix` (96 bytes), `LMatrix` (96 bytes)

Certain constants that are dependent on the precision are also available in different versions:

`MINREALs / MAXREALs` for Single Precision
`MINREALl / MAXREALl` for Double Precision
`MINREALr / MAXREALr` for datatype `Real` (single or double depending on C4D)

Please note that those values are C4D's own definitions and not the IEEE min/max values.

## 2.2 Single Precision Exceptions

There is no rule without exception – a couple places in C4D still use single precision floating-point numbers for memory consideration.

Here's the overview

| *Area* | *Occurence* |
|---|---|
| VertexMap data | PointObject::CalcVertexMap<br>VertexMapTag::GetDataAddressR<br>VertexMapTag::GetDataAddressW |
| Floating-point bitmaps | BaseBitmap::SetPixelCnt<br>BaseBitmap::GetPixelCnt |
| Weights for SDS | PolyWeight structure<br>HNData structure<br>SDSObject::GetPointWeights<br>SDSObject::GetEdgeWeights |
| OpenGL | large parts of OpenGL require single precision |
| Render Fragments | Parts of the PixelFragment structure use single precision |
| VideoPost effect | The PixelPost structure points to single-precision pixels (similar to bitmaps, see above) |
| Character animation Weight maps | CAWeightTag::GetWeightMap<br>CAWeightTag::SetWeightMap |

Try to check all those places (especially `GetDataAddressR/W` and `GetPixelCnt/SetPixelCnt`) to see if you're using the correct datatype. We also recommend that you quickly do a search for `(Real*)` or `(Vector*)` in your code as those places can lead to crashes if the wrong datatype was used.

### 2.3 Draw overrides

In R12 C4D's OpenGL has been rewritten from the ground up.

In any overrides to the Draw function (e.g. Objects, Tags, Scene Hooks etc.) you need to set a matrix before you start drawing that defines a coordinate system all operations are based upon. It is important that you don't forget this –you'll get random drawing results otherwise.

***Old***

```
Bool DoubleCircleData::Draw(PluginObject *op, LONG type, BaseDraw *bd,
BaseDrawHelp *bh)
{
	if (type!=DRAWPASS_HANDLES) return TRUE;

	LONG hitid = op->GetHighlightHandle(bd);
	const Matrix &m = bh->GetMg();
	if (hitid==0)
		bd->SetPen(GetWorldColor(VIEWCOLOR_SELECTION_PREVIEW));
	else
		bd->SetPen(GetWorldColor(VIEWCOLOR_ACTIVEPOINT));

	bd->Handle3D(GetRTHandle(op,0)*m,HANDLE_BIG);
	bd->Line3D(GetRTHandle(op,0)*m,m.off);

	return TRUE;
}
```

***New***

```
DRAWRESULT DoubleCircleData::Draw(BaseObject *op, DRAWPASS type, BaseDraw
*bd, BaseDrawHelp *bh)
{
	if (type!=DRAWPASS_HANDLES) return DRAWRESULT_SKIP;

	LONG hitid = op->GetHighlightHandle(bd);
	const Matrix &m = bh->GetMg();
	if (hitid==0)
		bd->SetPen(GetViewColor(VIEWCOLOR_SELECTION_PREVIEW));
	else
		bd->SetPen(GetViewColor(VIEWCOLOR_ACTIVEPOINT));

	bd->SetMatrix_Matrix(op, m);
	bd->DrawHandle(GetRTHandle(op,0),DRAWHANDLE_BIG,0);
	bd->DrawLine(GetRTHandle(op,0),Vector(0.0),0);

	return DRAWRESULT_OK;
}
```

In this example from C4D's SDK the matrix „m" is set for all subsequent drawing operations. You only need to set the matrix if any drawing operations follow. As a nice side-effect the matrix doesn't need to be multiplied manually anymore, saving valuable calculation time (marked in red).

Please also note that the result now is now expected to be one of three states instead of the former boolean value. `DRAWRESULT_SKIP` should be returned if nothing was drawn (either because currently there is nothing to draw or because elements are outside of the view area).

### 2.4 Printf, Scanf, SScanf

Please note that %f (float) is no longer valid. Use %lf (double) instead.

### 2.5 BaseFile and HyperFile

All C4D fileClasses now support 64 bit also in the 32 bit version. GetLength(), GetPosition(), Seek() all return LLONG. Make sure you use LLONG when accessing those members as otherwise you'll run into problems when the file length exceeds 2 GB.

### 2.6 AddToExecution

Objects that override the method `Execute` need to set `OBJECT_CALL_ADDEXECUTION` during registration, otherwise no call to `Execute` will be made.

***Old***

```
RegisterObjectPlugin(myid,myname,
PLUGINFLAG_HIDEPLUGINMENU,MyObject::Alloc,"Omydescription",NULL,0);
```

***New***

```
RegisterObjectPlugin(myid,myname,OBJECT_CALL_ADDEXECUTION|
PLUGINFLAG_HIDEPLUGINMENU,MyObject::Alloc,"Omydescription",NULL,0);
```

### 2.7 BaseSelect

`BaseSelect::FlushAll` needs to be replaced by the call `BaseSelect::DeselectAll`

### 2.8 GeGetVersionType

Before R12 GeGetVersionType returned a flag set that could combine several values. Now you'll get exactly one enumeration value, specifying what version of C4D is running.
For a precise overview which features are present in all editions please visit the official website www.maxon.net

| | |
|---|---|
| VERSIONTYPE_PRIME | CINEMA 4D Prime |
| VERSIONTYPE_BODYPAINT | BodyPaint 3D |
| VERSIONTYPE_STUDIO | CINEMA 4D Studio |
| VERSIONTYPE_VISUALIZE | CINEMA 4D Visualize |
| VERSIONTYPE_BROADCAST | CINEMA 4D Broadcase |
| VERSIONTYPE_BENCHMARK | CINEBENCH benchmark<br>Your plugin won't be loaded in that case, so you shouldn't encounter this result |
| VERSIONTYPE_UPDATER | C4D Updater<br>Your plugin won't be loaded in that case, so you shouldn't encounter this result |
| VERSIONTYPE_INSTALLER | C4D Installer<br>Your plugin won't be loaded in that case, so you shouldn't encounter this result |
| VERSIONTYPE_NET_CLIENT | CINEMA 4D NET Client |
| VERSIONTYPE_NET_SERVER_3 | CINEMA 4D NET Server – 3 seat license |
| VERSIONTYPE_NET_SERVER_UNLIMITED | CINEMA 4D NET Server – unlimited seat license |

Independent of GeGetVersionType you can retrieve the following information from GeGetSystemInfo:

| | |
|---|---|
| SYSTEMINFO_COMMANDLINE | Application was started in command line mode without graphical user interface.<br>Your plugin might allow operation in that case so the user is able to render on render farms |
| SYSTEMINFO_DEMO | Application is a save-disabled demo<br>As we don't offer this edition currently you shouldn't encounter this result |
| SYSTEMINFO_SAVABLEDEMO | 42-day trial version of CINEMA 4D |
| SYSTEMINFO_SAVABLEDEMO_ACTIVE | 42-day trial version of CINEMA 4D that was successfully activated.<br>If this bit is returned SYSTEMINFO_SAVABLEDEMO is automatically set too. |

### 2.9 Spline length calculations

The `SplineObject` methods `InitLength`, `FreeLength`, `UniformToNatural`, `GetLength` and `GetSegmentLength` were not thread-safe before R12. In order to change this they are no longer available in the spline object, but in a new helper class `SplineLengthData`.

***Old***

```
SplineObject *op;
...
if (!op->InitLength()) goto Error;
newval = op->UniformToNatural(oldval);
op->FreeLength();
```

***New***

```
SplineObject *op;
...
AutoAlloc<SplineLengthData> sld;
if (!sld || !sld->Init(op)) goto Error;
newval = sld->UniformToNatural(oldval);
```

## 2.10 Removed functionality

- R12 no longer supports the PowerPC platform under OS X and requires OS X 10.5 or higher. This allows you to build smaller universal binaries and results in quicker build times.
  As long as your project does not override C4D's API settings you'll automatically benefit from this change.

- Bones are no longer supported. Use Joints instead

- Anything bone-related is no longer supported – e.g. Claude Bonet Tags or the Bone Tool

- Mocca IK is no longer supported. Use the CA IK Tag instead.

- the tags FixExpression, IKTag, KinematicTag and AnchorTag are no longer supported. Use the CA IK Tag instead.

- PoseMixer is no longer supported. Use PoseMorph instead.

- MorphTags are no longer supported. Use PoseMorph instead.

- Shockwave export is no longer supported

- UZR export is no longer supported

- Photoshop filters are no longer supported

- FlashEx export is no longer supported

- FBX 6 im-/export is no longer supported. Use FBX 2010 instead.

## 2.11 Freeze Transformations

With R12 you probably have noticed that `BaseObject` no longer has members like `GetPos`, but instead twice as many: `GetRelPos` and `GetAbsPos`.

To understand what's going on I'll quickly introduce you to Freeze Transformations.

For (character) animation it is important that you can define rest poses. Think e.g. of the typical CG human that has arms and legs stretched. Most objects carry all kinds of complicated position, scale and rotation information. Freeze Transformations allow you to "freeze" this information, but reset position/rotation to 0.0 and scale to 1.0.
This is great as it allows you at any time to go back to the original – just reset all position and angle information to 0.0 - and your object has reached its rest state again.
You could achieve the same by adding for every single object in your hierarchy a parent null object, but Freeze Transformations let you do this without the need to change the hieararchical structure. Technically however you can think of it as a "parent null object" without the actual body of a physical object.

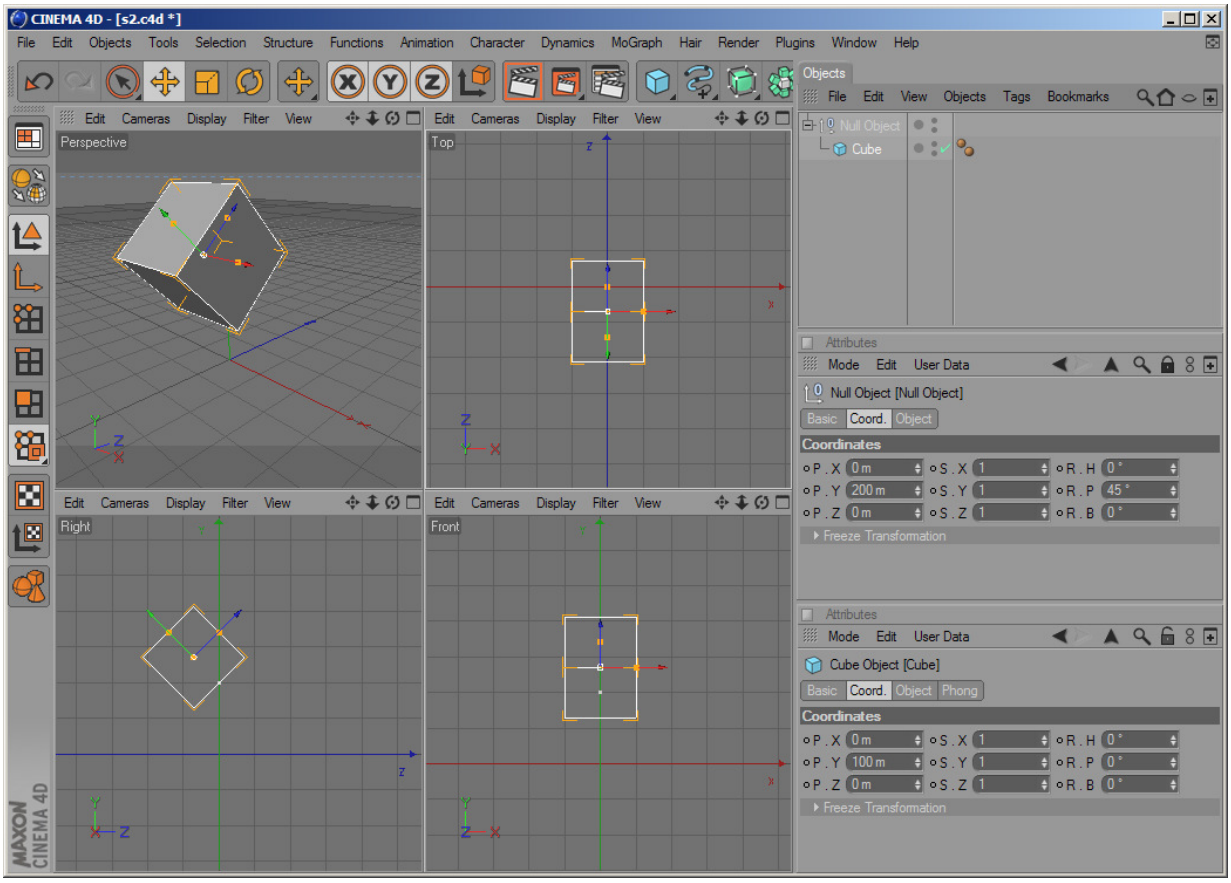
All parts of the application that deal with keyframing only access values without the freeze transformation part. This way you can conveniently animate (the rest pose is always 0.0 for position and rotation) and avoid rotation singularities. The correct routines for animation and keyframing are labeled "Rel".

Other parts of the application however (e.g. like a target expression) need to get the absolute local position of a target – the position that consists of frozen and regular matrix. For those purposes you need to use the routines that are labeled "Abs".

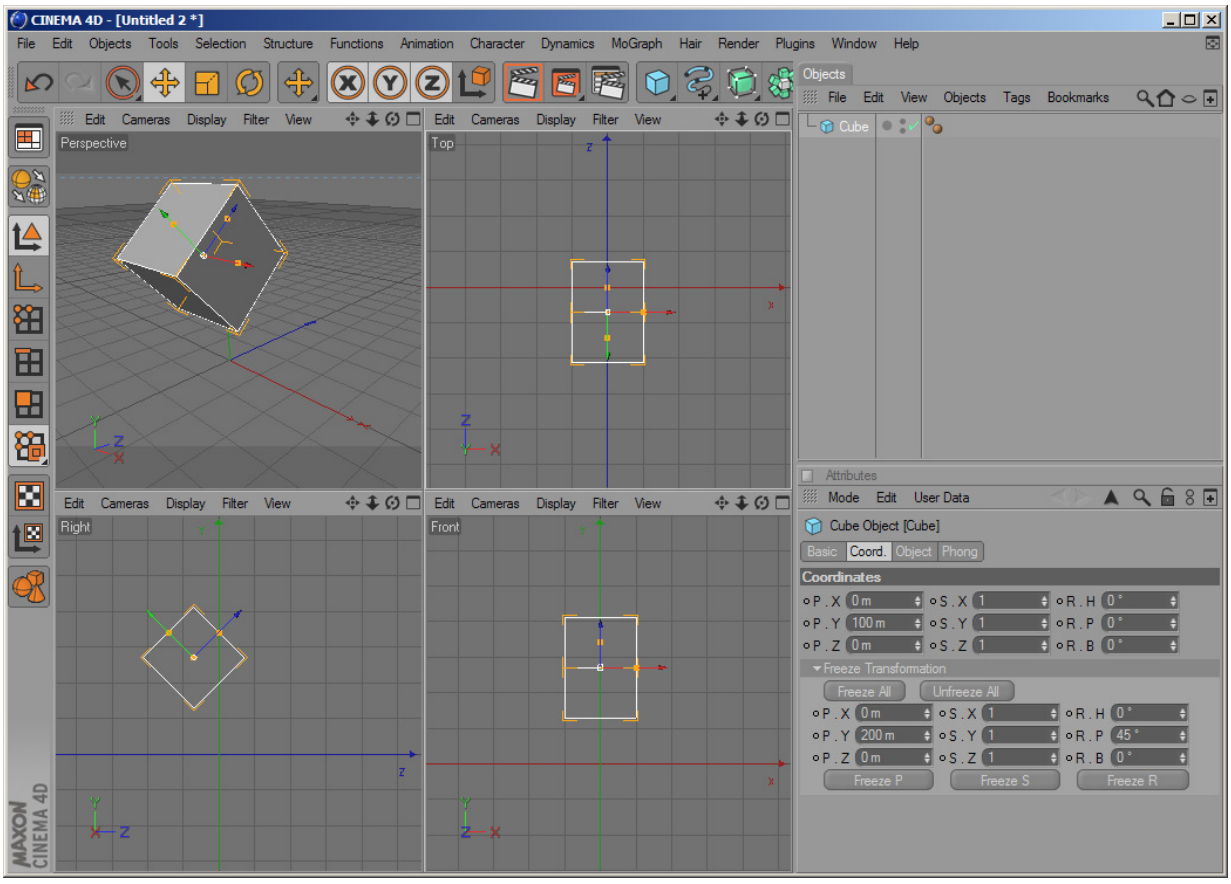
To clarify: as long as an object has no Freeze transformation assigned (which is always the default case when you create a new object) there is no difference between the Abs or Rel. So e.g. after calling `AllocObject()` it doesn't matter which routine you choose. If you however want to read the position of an object (and that object uses freeze transformations) there is a difference.

`GetMl` / `SetMl` and `GetMg` / `SetMg` always include the frozen information, thus are "Abs" versions.

After so much theory a practical example. Take a look at the cube and null object on the next page. The null object has a position of `(0/200/0)` and rotation of `(0°/45°/0°)`. The cube has a position of `(0/100/0)`. You can see to the left its global position in space.




Now let's try to do the same with one single object:



You can see that the Null object's position / rotation have been transferred to the Cube's frozen position and frozen rotation. The result is exactly the same as before, just without the need for an extra null object.

Now if you call `cube->GetAbsPos()` the return value will be `(0/270.711/-70.711)`. When you call `cube->GetRelPos()` however you'll get `(0/100/0)` as a result. The coordinate manager offers exactly those two display modes: Object (Abs) and Object (Rel).

If you use `GetParameter` and `SetParameter` to access values the same distinction exists: instead of `ID_BASEOBJECT_POSITION` there is `now ID_BASEOBJECT_REL_POSITION` and `ID_BASEOBJECT_ABS_POSITION` (along with `ID_BASEOBJECT_FROZEN_POSITION`).

As copying position, rotation and scale data from one object to another is much more complex than before there is a new routine `BaseObject::CopyMatrixTo` that copies all position, rotation and scale data (including the frozen states) over to another object.

Let's take a look at the mathematical background:

In C4D the matrix multiplication order for two matrices A and B

```
[C] = [A] * [B]
```

is defined as the rule "A after B"… so the following two lines are equivalent:

```
transformed_point = original_point * [C]

transformed_point = (original_point * [B]) * [A]
```

(the original point is first transformed by matrix `[B]` and then later by `[A]`)

An object's global matrix is calculated this way:

```
[Global Matrix] = [Global Matrix of Parent] * [Local Matrix]
```

or

```
[Global Matrix] = op->GetUpMg() * op->GetMl()
```

To calculate the local matrix of an object the formula is:

```
[Local Matrix] =  [Frozen Translation] * [Frozen Rotation] *
                  [Relative Translation] * [Relative Rotation] *
                  [Frozen Scale] * [Relative Scale]
```

or

```
[Local Matrix] =
      MatrixMove(op->GetFrozenPos()) * HPBToMatrix(op->GetFrozenRot()) *
      MatrixMove(op->GetRelPos()) * HPBToMatrix(op->GetRelRot()) *
      MatrixScale(op->GetFrozenScale) * MatrixScale(op->GetRelScale());
```

or equally

```
[Local Matrix] =
      MatrixMove(op->GetAbsPos()) *
      HPBToMatrix(op->GetAbsRot()) *
      MatrixScale(op->GetAbsScale);
```

Please note that the both scales are applied at the very beginning, so the local matrix is not the same as [Frozen Matrix] * [Relative Matrix]. This is necessary in order to guarantee that your local matrix always stays a rectangular system; distorted systems would be too complicated to handle.

## 2.12 Units

C4D is now fully "unit aware". The base unit is defined in the "document scale" document settings. Any metric value (e.g. a position) is defined in relation to this base unit.
E.g. if `op->GetMg().off` returns `(0/100/0)` and the document scale is set to `4 yards` this means the value equals `(0/400 yards/0)` "in real life". C4D's in- and output edit fields automatically do all the conversions for you, so normally you never have to deal with that.

There are a couple exceptions:

- when you write im-/ or exporters a translation needs to take place (as external file formats most of the time will not support units).
  In the SDK there is an example STL im-/exporter that shows how to offer scale controls (datatype `UnitScaleData`) for the user. The filter then uses the routine `CalculateTranslationScale` to calculate a simple size multiplier for im- or export.

- when you copy/paste objects (or e.g. materials, videopost effects etc,) from one document to another the sizes need to be adapted. In that case use the routine `CalculateTranslationScale` and then `baselistelement->Scale(scale factor)`

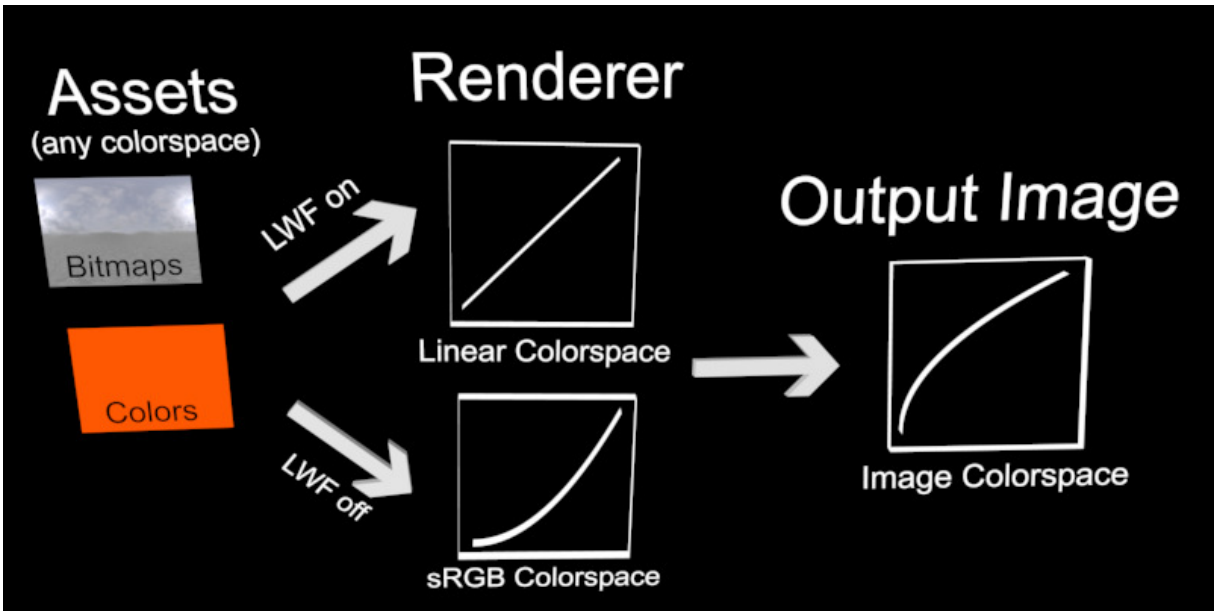
- if you need to convert a metric value into a String make sure to use `FormatNumber` and not `RealToString`. `RealToString` will do a raw conversion of your number without units whereas `FormatNumber` will convert the number into a string evaluating the correct preference/document settings.

**2.13 Color Profiles**

C4D now supports color management and color profiles in all places of the application. The render engine natively supports linear workflow.

You mostly need to be aware of linear workflow and color conversion when you write shaders, materials or videopost effects.

Here's a schematic of C4D's colorspace transformation pipeline:




Before rendering all assets (bitmaps, colors or other external references) are transferred into the render engine's colorspace. This can either be Linear Colorspace – if the user has "Linear Workflow in C4D's document settings enabled – or sRGB Colorspace.

The render engine now does all lighting calculations and creates an image – which is still in the colorspace that the renderer is operating in.

As a final step the render output is transformed into a user chosen output color space. Mostly this will be sRGB if the image has a bit depth of 8 or 16 and Linear for 32. But the use can choose any output colorspace in the render settings.

Please note that multi-pass images produced by the renderer always have render colorspace assigned – this is different from the regular image where you can set any arbitrary output colorspace. The reason for this is that a multi-pass image tries to recreate the render operations (adding multiple passes). If this didn't happen in render colorspace the results would look completely wrong.
On a further note this is the reason for applications like Photoshop that cannot show multi-passes correctly if bit depth is 32 and LWF is off – they have a hard-wired built-in logic that 32 bit images cannot have any color profile assigned and always use linear space.

### 2.13.1 Shaders and Materials

In C4D all assets (textures, colors etc.) need to be transformed to render colorspace before a rendering is initiated (see last page). The default in R12 is that C4D uses linear workflow, so the render colorspace is linear.

- If your shader or material uses any colors those need to be transformed in `InitRender`. A good example is the SDK mandelbrot shader. It transforms all input colors into render colorspace. To do that use the call

  ```
  transformed_color = irs.TransformColor(original_color)
  ```

  where 'irs' is the `InitRenderStruct` that is passed to `InitRender`.

  If you use another (sub-)shader as input you don't need to do anything – as its colors are already corrected!

- In some cases, e.g. when your color is returned from a bitmap image (not bitmap shader) you can't precompute colors. In that case TransformColor must be called during `Output / CalcSurface` for each call. As the `InitRenderStructure` is no longer present you need to copy `InitRenderStructure` in `InitRender` and reuse it later in `Output / CalcSurface`.
  Or you can also save the members 'linear_workflow' and 'document_colorprofile' of `InitRenderStruct` and the later do the color-conversion yourself:

  ```
  if (document_colorprofile==DOCUMENT_COLORPROFILE_SRGB &&
      linear_workflow)
    output_col = TransformColor(input_col,
                 COLORSPACETRANSFORMATION_SRGB_TO_LINEAR);
  else if (document_colorprofile==DOCUMENT_COLORPROFILE_LINEAR &&
           !linear_workflow)
    output_col = TransformColor(input_col,
                 COLORSPACETRANSFORMATION_LINEAR_TO_SRGB);
  else
    output_col = input_col;
  ```

- If you use a gradient C4D takes care of all colorspace conversion as long as you pass on the `InitRenderStruct` structure to its `InitRender` call.

  The only thing you need to modify is the resource of the gradient:

  ```
  GRADIENT SDKGRADIENTSHADER_COLOR { COLOR; ICC_BASEDOCUMENT; }
  ```

  `By adding` the keyword "`ICC_BASEDOCUMENT`" you notify C4D that the gradient uses the document's colormanagement

  In the SDK you'll find the GradientShader example that uses this concept.

### 2.13.2 VideoPost effects

For VideoPost effects the situation depends on which calls are overriden/execute. Depending on the videopost call you will be confronted with different color spaces:

- In `ExecutePixel` you'll get render color space

  Here's an example: you override `ExecutePixel` and assign a specific color to every 2nd pixel. This color that is set needs to be transformed before – from asset color space to render color space. Use `TransformColor` like for Shaders and Materials.

- In `Execute` you'll get image color space when you access any `VPBuffer`.

  Let's say you override Execute and want to embed a logo at the end of the render process. This logo bitmap needs to been transformed – from asset color space to image color space.

  To get from asset space to output space you need to invoke to transforms: from asset to render space and from render space to output space.

  For asset to output space use `TransformColor`.
  For render to output space use `vps->render->IccConvert`. IccConvert can convert in both directions, so either from render to output space or from output to render space.

  Instead of this you can also transform in one batch – from asset to output space. To do this use the following code:

```
Bool transformation_necessary=FALSE;

const ColorProfile *input_profile  = logo_bitmap->GetColorProfile();
const ColorProfile *output_profile = ((MultipassBitmap*)vps->render-
>GetBuffer(VPBUFFER_RGBA,NOTOK))->GetColorProfile();

AutoAlloc<ColorProfileConvert> convert;
if (convert)
{
  transformation_necessary = convert->PrepareTransform
      (COLORMODE_RGBf,input_profile,COLORMODE_RGBf,output_profile,FALSE);
}
```

…and later on…

```
convert->Convert(src_data,dst_data,count,skip_input,skip_output)
```

As a final example please take a look at the VPReconstructImage in the SDK. The videopost effect calculates all fragments for any pixel and reconstructs the image based on those fragments. All fragments contain colors in render colorspace. That's why after weighting those colors the call IccConvert is invoked to transform the color from render colorspace to output colorspace.

# Chapter 3 – Using new safety features

### 3.1 sprintf and vsprintf

Try to remove any occurences of sprintf and vsprintf in your code. Instead use the following definitions:

```
int sprintf_safe(char *_DstBuf, int _MaxCount, const char * _Format, ...);

int vsprintf_safe(char *_DstBuf, int _MaxCount, const char * _Format,
va_list _ArgList);
```

The _safe versions will ensure that there won't be any buffer overruns (and thus crashes). They're identical to snprinf and vsnprinf, but as the Microsoft implementation is not safe (and crashes…) by using C4D's version you are on the safe side.

### 3.2 GeAllocType

We introduced clearer (and safer) statements for allocation, clearing and copying of data types. While the old calls are still available, please try to use the new one as much as possible – you might detect a couple of memory overwrites and crashes that are hard to find otherwise.

***Old***

```
SReal *myptr = (SReal*) GeAlloc(sizeof(SReal)*10);
```

***New***

```
SReal *myptr = GeAllocType(SReal,10);
```

The new code does not contain any typecasts anymore. The big advantage now is that whenever you change a datatype – say this was R11.5 code and now you need Real instead of SReal for better precision – you'll get a compilation error for the following case:

```
Real *myptr = GeAllocType(SReal,10);
```

It'll automatically detect that the two datatypes Real and SReal (which was forgotten to be renamed) do not match.

Like GeAllocType there is also GeAllocTypeNC (allocation that does not clear its memory).

### 3.3 GeCopyMemType and FillMemType

GeCopyMemType copies memory, but again uses type checks:

***Old***

```
SReal *src,*dst;
…
CopyMem(src,dst,10*sizeof(SReal));
```

***New***

```
SReal *src,*dst;
…
CopyMemType(SReal,src,dst,10);
```

FillMemType copies memory, but again utilizes type checks:

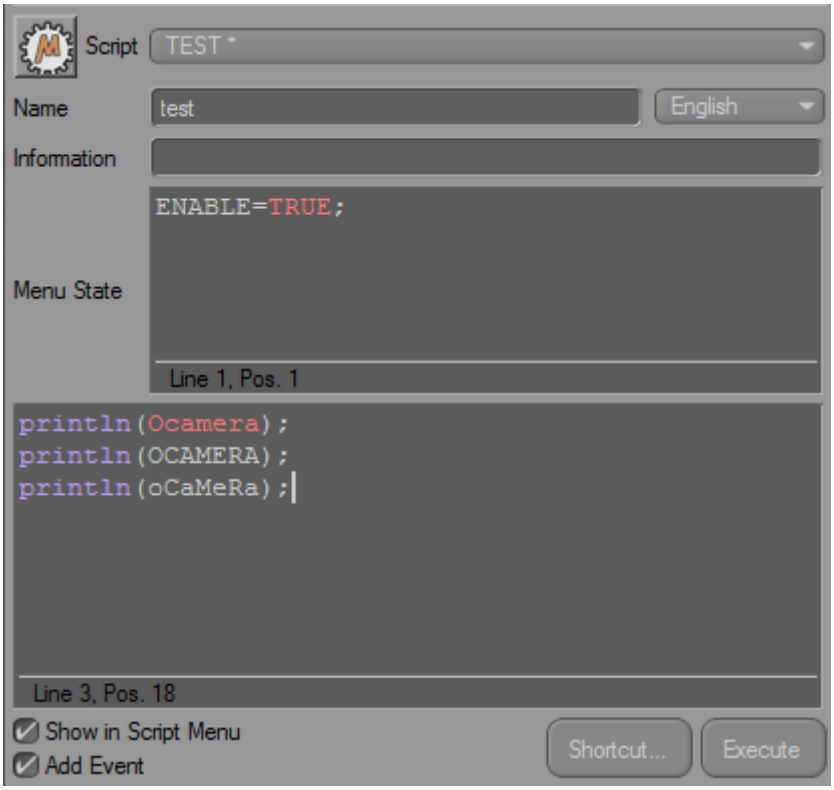
```
SReal *mem;
…
FillMemType(SReal,mem,10,0x00);
```

Please note that the fill parameter (last parameter) needs to be in the range 0..255

# COFFEE Transition to R12

- One of the biggest changes is that COFFEE constant now are fully case-sensitive. Before, due to a bug you could write Ocamera or OCAMERA or oCaMeRa – your code would still work in that case. With R12 this is no longer possible, you need to use the correct constants.

  C4D's syntax highlighting will assist you in this process – if a constant is not highlighted, then it won't be recognized




- COFFEE constants are a 100% in sync with the C++ API. Any constant documented in the C++ API is accessible in COFFEE.

- COFFEE does no longer support deprecated symbols from Release 6, 7, 8 and 9. It still supports lot of deprecated symbols though, e.g. `UNDO_NEW` (the correct name would be `UNDOTYPE_NEW`).

- Please also read chapter 2 of "C++ Transition to R12", as this largely affects COFFEE programming.

- While the routines like `BaseObject::GetPosition` still are available try to change your code to the correct version `BaseObject::GetAbsPos` or `BaseObect::GetRelPos`. See also Chapter 2.7