

Melange Quick-start Guide

Apr 24, 2013

What is Melange?

It is MAXON Computer's file exchange and render connection library. Melange allows to write, read and render CINEMA 4D project files (.c4d) from outside the program.

What do you need?

Please download the latest version of the Melange library from www.plugincafe.com. You can also find the complete documentation at this page. Minimum OS Requirements:

- Windows XP
- OSX 10.5.8

How to setup your project?

Extract the archive and add the the include path and regarding library version to your project.

Available library versions for Windows:

- Visual Studio 2008, 2010 and 2012 (32-bit & 64-bit / MD & MT / debug & release)

and OSX:

- universal binaries for XCode >= 4.0 (32-bit & 64-bit / debug & release / Intel only)

There are a few functions in Melange which have to be overloaded. For the start you can just include "default_alien_overloads.h" to define them. But if you want to use your own memory functions you can have a look inside this header, copy the functions to your project and change them to your needs.

One important function has to be defined in any case : `GetWriterInfo()`. Because in this one the name of the writer and the unique (!) ID has to be set. This identifier will be saved to the project as well and will be displayed in CINEMA 4D at the "Info" section of the project settings.

```
// overload this function and fill in your own unique data
void GetWriterInfo(LONG &id, String &appname)
{
    id = 1234567; // register your own plugin ID once for your exporter and enter it here (www.plugincafe.com)
    appname = "My Application Name";
}
```

A namespace is defined to avoid trouble when linking Melange to other APIs which may contain the same definitions. Therefore you should add "using namespace _melange;" to relevant part of your project or use the prefix "_melange_:" for the Melange types explicitly.

All about writing a CINEMA 4D project file

(The examples does not contain any error checking to keep the code as simple as possible.)

How to create and save a simple c4d project containing only one Cube object?

```
// create a document (includes all the objects, materials and tags)
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create a cube primitive object
BaseObject* myCube = BaseObject::Alloc(OCube); // other types are Olight, Ocamera, Ospline, ...

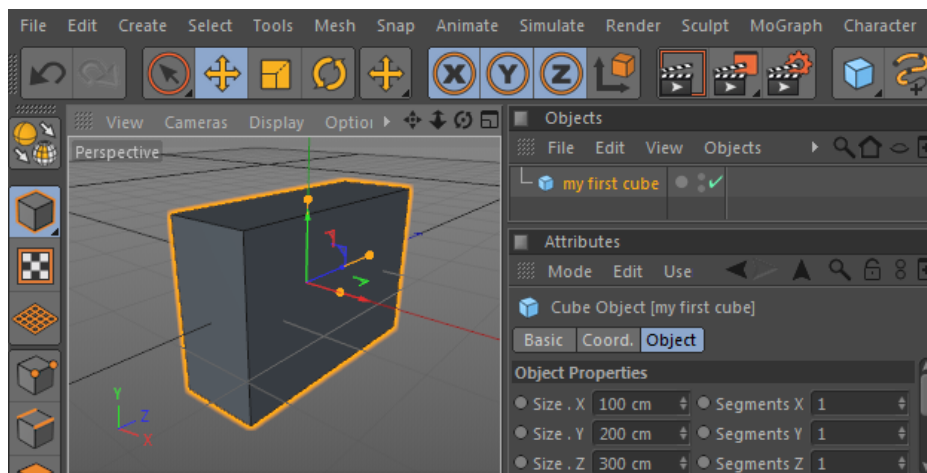
// set the name
myCube->SetName("my first cube");

// set length parameter (the IDs can be found in "c4d_parameter_ids.h" or inside the folder "parameter_ids")
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0, 200.0, 300.0));

// insert into the project
c4dDoc->InsertObject(myCube, NULL);

// write the document (the third parameter is private and must be set to SAVEDOCUMENTFLAGS_0)
SaveDocument(c4dDoc, "simple_project.c4d", SAVEDOCUMENTFLAGS_0);
```

The resulting scene loaded into CINEMA 4D:



How to create and save a more complex c4d project containing also a null and a polygon object?

```
// create a document
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create a null object
BaseObject* myNull = BaseObject::Alloc(Onull);
myNull->SetName("my first null");
c4dDoc->InsertObject(myNull, NULL);

// create a cube object
BaseObject* myCube = BaseObject::Alloc(OCube);
myCube->SetName("my first cube");
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0, 200.0, 300.0));
c4dDoc->InsertObject(myCube, myNull); // insert into the project as children of the null object

// create a polygon object
BaseObject* myMesh = BaseObject::Alloc(OPolygon);
myMesh->SetName("my first mesh");
// set position/rotation/scale
myMesh->SetAbsPos(Vector(0.0, 200.0, 0.0));
myMesh->SetAbsRot(Vector(Rad(45.0), 0.0, 0.0));
myMesh->SetAbsScale(Vector(100.0, 100.0, 100.0));
// insert into the project as children of the null object and after the cube object
c4dDoc->InsertObject(myMesh, myNull, myCube);

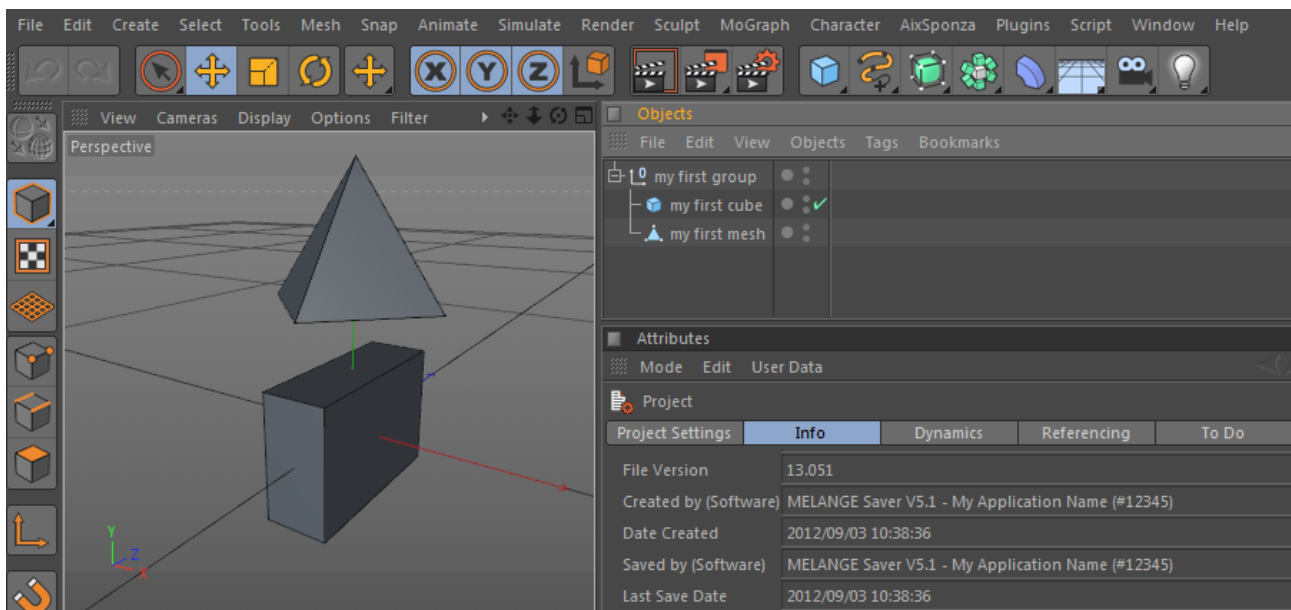
// resize the object, setup points and polygons
((PolygonObject*)myMesh)->ResizeObject(5, 5, 0); // ResizeObject(Point Count, Polygon Count, Ngon Count)
// get the pointer to the vertices for writing
Vector *vAdr = ((PolygonObject*)myMesh)->GetPointW();
// get the pointer of the polygons for writing
CPolygon *pAdr = ((PolygonObject*)myMesh)->GetPolygonW();

// set point data Vector(x, y, z)
vAdr[0] = Vector( 1.0, 0.0, 1.0);
vAdr[1] = Vector(-1.0, 0.0, 1.0);
vAdr[2] = Vector(-1.0, 0.0,-1.0);
vAdr[3] = Vector( 1.0, 0.0,-1.0);
vAdr[4] = Vector( 0.0, 2.0, 0.0);

// set polygon data (point indices CPolygon(a, b, c, d) for quads, CPolygon(a,b,c) for triangles)
pAdr[0] = CPolygon(0, 1, 2, 3);
pAdr[1] = CPolygon(4, 1, 0);
pAdr[2] = CPolygon(4, 2, 1);
pAdr[3] = CPolygon(4, 3, 2);
pAdr[4] = CPolygon(4, 0, 3);

// write the document
SaveDocument(c4dDoc, "complex_project.c4d", SAVEDOCUMENTFLAGS_0);
```

The resulting project loaded into CINEMA 4D:



How to apply materials and textures to the objects?

```
// create a document
BaseDocument* c4dDoc = BaseDocument::Alloc();

// create and add a cube
BaseObject* myCube = BaseObject::Alloc(Ocube);
myCube->SetName("my first textured cube");
myCube->SetParameter(PRIM_CUBE_LEN, Vector(100.0,200.0,300.0));
c4dDoc->InsertObject(myCube, NULL);

// create a material with a color texture
Material* myMat = (Material*)BaseMaterial::Alloc(Mmaterial);
myMat->SetName("my first material");
myMat->MakeBitmapShader(MATERIAL_COLOR_SHADER, "myTexture.jpg");
// insert the material into the project
c4dDoc->InsertMaterial(myMat);

// create a tag which maps the material to the object (the tag is inserted into the object/document)
BaseTag* myTextureTag = myCube->MakeTag(Ttexture);

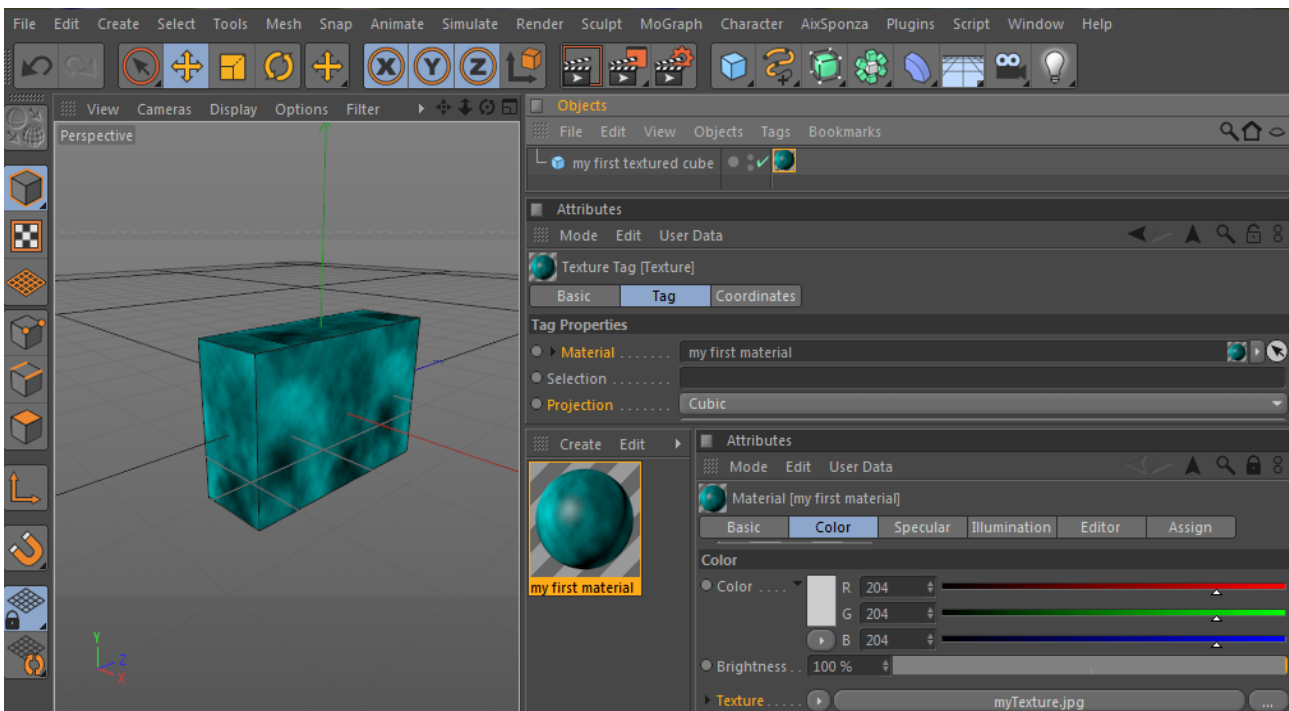
// create a link for the new material
GeAliasLink* myMatLink = GeAliasLink::Alloc();
myMatLink->SetLink(myMat);

// set material link of the texture tag
myTextureTag->SetParameter(TEXTURETAG_MATERIAL, *myMatLink);
GeAliasLink::Free(myMatLink);

// set texture projection to cubic
myTextureTag->SetParameter(TEXTURETAG_PROJECTION, TEXTURETAG_PROJECTION_CUBIC);

SaveDocument(c4dDoc, "simple_material_project.c4d", SAVEDOCUMENTFLAGS_0);
```

The resulting project loaded into CINEMA 4D:



Using the render connection

(The examples does not contain any error checking to keep the code as simple as possible.)

In your Application starting routine call the following initialization functions:

```
// initialize the main thread pointer and timer
GeInitThreads();

// initialize Melange network code
GeIpInitNetwork();

// start the renderer with the path to the executable and the custom port (optional)
// (the current PID is passed to the renderer as well and will shutdown the server if the process ends)
StartRenderServer(PathToCINEMA4D, 1234);
```

For rendering, define your own class derived from IpCommunicationThread:

```
// one thread used to manage the rendering
class RenderThread : public _melange_::IpCommunicationThread
{
public:
    virtual void Main ()
    {
        // open communication to the server
        // (local host + port used when starting, UID of your application, initial timeout, connection timeout)
        Open("127.0.0.1:1234", UNIQUE_APPLICATION_ID, 10, 60);

        // send the scene to the renderer
        // (pass path of CINEMA 4D file or an optional BaseDocument reference as second parameter)
        SendScenefile(SceneFilePath);

        // start rendering
        // (X resolution, Y resolution, bitmap color mode, allow saving the render image, high priority)
        StartRender(640, 480, COLORRESOLUTION_FLOAT, FALSE, FALSE);

        // get the render image and progress while rendering
        Bool running = TRUE;
        while (!TestBreak() && running)
        {
            // calls your callback "UpdateImage" to get the image update
            GetImageUpdate(MyUpdateImage, NULL, &running);

            // check for error and how much rendering is done
            GetRenderStatus(NULL, &rendererror, &percent_done, &renderphase);
        }

        // frees the session on the server (old scenes will be freed automatically if new ones are uploaded)
        FreeSession();

        // ends the communication with the server and shuts it down
        // (you can pass TRUE to shutdown the server at this point if no further rendering is needed)
        Close(FALSE);
    }
};

// only needed if you did not shutdown the server in the render thread (Close(FALSE) used)
class ShutDownThread : public _melange_::IpCommunicationThread
{
public:
    virtual void Main ()
    {
        Open("127.0.0.1:1234", 1, 10, 60);
        Close(TRUE);
    }
};
```

To receive the rendered image define an *UpdateImage* callback:

```
// here the rendered parts of the image are received
void MyUpdateImage(void *userdata, LONG xpos, LONG ypos, LONG xcnt, ColorResolution bpp, UCHAR* rgba_data)
{
    // xpos & ypos defines the starting position and xcnt the amount of pixel of the RGBA data
    // bpp informs about the color depth (8, 16 or 32 bit)
    // rgba_data contains the image data (4 bytes per pixel)
    // fill your bitmap structure with the given data
    // ...
}
```

To start rendering, create a new render thread from your class and start it:

```
RenderThread rThread = NULL; // (global defined)
ShutdownThread sdThread = NULL; // (global defined)

// if no render thread is active start a new one
if (!rThread) rThread = gNew RenderThread;
if (!rThread || rThread->IsRunning()) return;
rThread->Start(TRUE);
```

In your application ending routine shutdown the server, delete the threads and call the following functions:

```
// shutdown the renderer
if (!sdThread) sdThread = gNew ShutdownThread;
if (!sdThread || sdThread->IsRunning()) return;
sdThread->Start(TRUE);
sdThread->Wait(); // make sure that the command will be send to the server before execution continue
// delete the threads
gDelete(rThread);
gDelete(sdThread);

// free Melange network
GeIpCloseNetwork();

// free Melange thread code
GeEndThreads();
```

All about reading a CINEMA 4D project file

(The examples does not contain any error checking to keep the code as simple as possible.)

How to load and browse through a c4d project?

```
// load the project with objects
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS);

// start browsing at the first top level object
BrowseHierarchy(c4dDoc->GetFirstObject());

// delete the project after usage
BaseDocument::Free(c4dDoc);

// define this function to browse through the hierarchy
void BrowseHierarchy(BaseObject* object)
{
    while (object)
    {
        // get basic data
        String name = object->GetName();
        LONG type = object->GetType(); // Ocube, Ospline, Opolygon, Olight, Ocamera, ...
        Matrix ml = object->GetMl(); // local matrix
        Matrix mg = object->GetMg(); // global matrix

        // depending on the type, get specific data for the current object
        GeData data; // universal data type, can handle all data formats (Real, LONG, Bool, Vector, ...)
        if (type == Ocamera)
        {
            object->GetParameter(CAMERAOBJECT_FOV, data);
            Real fov = data.GetReal();
        }
        // some objects have their own type definition
        else if (type == Opolygon)
        {
            PolygonObject* polyObject = (PolygonObject*)object;
            LONG pointCount = polyObject->GetPointCount();
            LONG polyCount = polyObject->GetPolygonCount();
            const Vector* points = polyObject->GetPointR();
            const CPolygon* polys = polyObject->GetPolygonR();

            // access the polygons and points
            for (LONG i = 0; i < polyCount; i++)
            {
                // get the 3 / 4 points of the triangle / quadrangle
                Vector pointA = points[polys[i].a];
                Vector pointB = points[polys[i].b];
                Vector pointC = points[polys[i].c];

                // only if the index of 3rd and 4th point is different we have a quad, if not we have a triangle
                if (polys[i].c != polys[i].d)
                {
                    Vector pointD = points[polys[i].d];
                }
            }
        }
    }
}
```

```

    // do the children
    BrowseHierarchy(object->GetDown());

    // get the next object
    object = object->GetNext();
}
}

```

How to access the materials of a c4d project?

```

// load the project with materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_MATERIALS);

// get the first material from the document and go through the whole list
BaseMaterial* mat = c4dDoc->GetFirstMaterial();
while (mat)
{
    // basic data
    String name = mat->GetName();

    // check if the material is a standard material
    if (mat->GetType() == Mmaterial)
    {
        // check if the given channel is used in the material
        if (((Material*)mat)->GetChannelState(CHANNEL_COLOR) // CHANNEL_LUMINANCE, CHANNEL_BUMP, ...
        {
            // channel color is turned on

            // get the color and brightness
            GeData data;
            mat->GetParameter(MATERIAL_COLOR_COLOR, data);
            Vector color = data.GetVector();
            mat->GetParameter(MATERIAL_COLOR_BRIGHTNESS, data);
            Real brightness = data.GetReal();
        }
    }
    else
    {
        // plugin material
        // parameter depend on the type, get with GetParameter()
        // ...
    }

    mat = mat->GetNext();
}

// delete the project after usage
BaseDocument::Free(c4dDoc);

```

How to get the assigned material of an object?

```

// load the project with objects and materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS | SCENEFILTER_MATERIALS);

// get the first object
BaseObject* object = c4d->GetFirstObject();

// get the texture tag holding the link to the assigned material
BaseTag* tag = object->GetTag(Ttexture);

// get the assigned material from the tag
BaseMaterial* assignedMat = tag->GetMaterial();

// delete the project after usage
BaseDocument::Free(c4dDoc);

```

How to use the extended loading functionality to get a geometric description for all objects?

Important note: This is only possible for projects saved with the “Save Polygons for Melange” option turned on in the CINEMA 4D preferences!

```
// load the project with objects and materials
BaseDocument* c4dDoc = LoadDocument(filepath, SCENEFILTER_OBJECTS | SCENEFILTER_MATERIALS);

// call all the Execute() of the document
c4dDoc->CreateSceneFromC4D();

// delete the project after usage
BaseDocument::Free(c4dDoc);
```

The following “alien” definitions are important. Add these classes for the objects you can transfer to your own internal data structures. Of course `AlienPolygonObjectData` has to be implemented in any case.

```
// self defined PolygonObjectData with own members and definition of the Execute() method
class AlienPolygonObjectData : public PolygonObjectData
{
    INSTANCEOF(AlienPolygonObjectData, PolygonObjectData) // important to get Execute() called!
public:
    // useful to store your own material and layer ID
    LONG layID;
    LONG matID;

    virtual Bool Execute()
    {
        // get object pointer
        PolygonObject* op = (PolygonObject*)GetNode();

        // access the geometry data as described above
        // ...

        return TRUE;
    }
}

// this self defined alien primitive object data will “understand” a Cube as parametric Cube
// all other objects are unknown and we will get a polygonal description for them
class AlienPrimitiveObjectData : public NodeData
{
    INSTANCEOF(AlienPrimitiveObjectData, NodeData)

    LONG type_id;
public:
    AlienPrimitiveObjectData(LONG id) : type_id(id) {}

    virtual Bool Execute()
    {
        // decide the object type
        if (type_id == Ocube)
        {
            // get the size of the cube
            GeData cubeData;
            GetNode()->GetParameter(PRIM_CUBE_LEN, cubeData);
            Vector xyzLength = cubeData.GetVector();

            // create your own parametric cube with length X, Y and Z
            // ...

            // returning TRUE we will NOT get another call of PolygonObjectData Execute()
            return TRUE;
        }

        // return FALSE to get a simpler geometric description of objects like Osphere, Ocone, ...
        // the Execute() of AlienPolygonObjectData will be called
        return FALSE;
    }
};
```

Now change the function `AllocAlienObjectData` to create the custom alien object data at allocation:

```
// this function has to be overloaded in any case (predefined in default_alien_overloads.h)
// it allocates the scene objects (primitives, camera, light, etc.)
NodeData *AllocAlienObjectData(LONG id, Bool &known)
{
    NodeData *m_data = NULL;
    known = TRUE;

    switch (id)
    {
        // supported element types
        case Opolygon: m_data = gNew AlienPolygonObjectData; break;
        case Osphere: m_data = gNew AlienPrimitiveObjectData(id); break;
        case Ocube: m_data = gNew AlienPrimitiveObjectData(id); break;
        case Oplane: m_data = gNew AlienPrimitiveObjectData(id); break;
        case Ocone: m_data = gNew AlienPrimitiveObjectData(id); break;
        case Otorus: m_data = gNew AlienPrimitiveObjectData(id); break;
        case Ocyylinder: m_data = gNew AlienPrimitiveObjectData(id); break;
        ...

        default: known = FALSE; break;
    }

    return m_data;
};
```

Note: The example “commandline” which is included in the SDK archive uses this method. Please have a look at it for further information.