

# Missing Plugin Remover

## A plugin for Cinema 4D

### Overview

If you ever load up an old scene, perhaps one created years previously, you sometimes get a message about missing plugins, with a list of IDs of plugins used in that scene which are no longer available and have not been loaded. Sometimes, this has no effect on the scene and you can ignore the message - though it would be nice to get rid of it because you see it every time you open the file. If the plugin was important to the scene, you will have to either get hold of the plugin for your version of C4D (if it is available) or substitute it for another. The problem is, even with a substitute you might still be stuck with the annoying message.

If you intend to give the scene to someone else, it looks very poor to distribute it still with that message showing every time they load it, but it isn't always easy to fix. In order to do so, you first have to find the object/tag/shader/whatever in the scene. However, since the plugin hasn't been loaded, there is no interface and all you are left with is the name. And of course, an object might have been renamed so you can't even search for the name. Worse, you might not even know what the plugin was, if it was a long time ago or this is someone else's file and you never used the plugin.

### How does Cinema handle missing plugins?

It handles them as well as could be expected. If the plugin was something appearing in the object manager, the object name is kept and the icon is replaced with a question mark, but of course it doesn't do anything - it behaves like a null object. If it's a shader, the shader name stays in the shader link in the same channel of the material, but you can't click the shader to show its interface (it doesn't have one, since it hasn't been loaded) and it does nothing. Missing tags appear as question mark icons in the object manager. Missing videoposts appear as normal in the list of post effects, but if clicked, they have no interface and do nothing.

The important things to note here are:

- the plugin has not been loaded into Cinema because it doesn't exist
- however, a placeholder for the missing plugin remains and will return the original ID value

This is good, because if the missing plugin could be restored, it would simply slot into the placeholder(s) left for it in the scene. Alternatively, since the placeholder still returns the original plugin's ID value, it should be possible to find all instances of it using that value.

### Can this be readily solved?

The first issue is, what was this plugin? The four most likely possibilities are that it was:

- an object of some kind (i.e. something visible in the object manager)
- a tag
- a shader
- a videopost (post-effect plugin)

If you don't know what it is, or can't deduce it from the name, you can do two things, and you will need the plugin name and ID value, which Cinema helpfully gives you in the missing plugins message. Make a note of that ID number and the missing plugin name.

If you have compiled the Cinema 4D SDK example plugins, there is an extremely useful plugin among them. This is the Active Objects Dialog. What it does is display details of every object, material, tag, shader, videopost, etc. in the scene, including their ID values. You can search through the list (which can be very long) to find the relevant ID value and then at least you will know what kind of scene element you are looking for. You would see something like this:

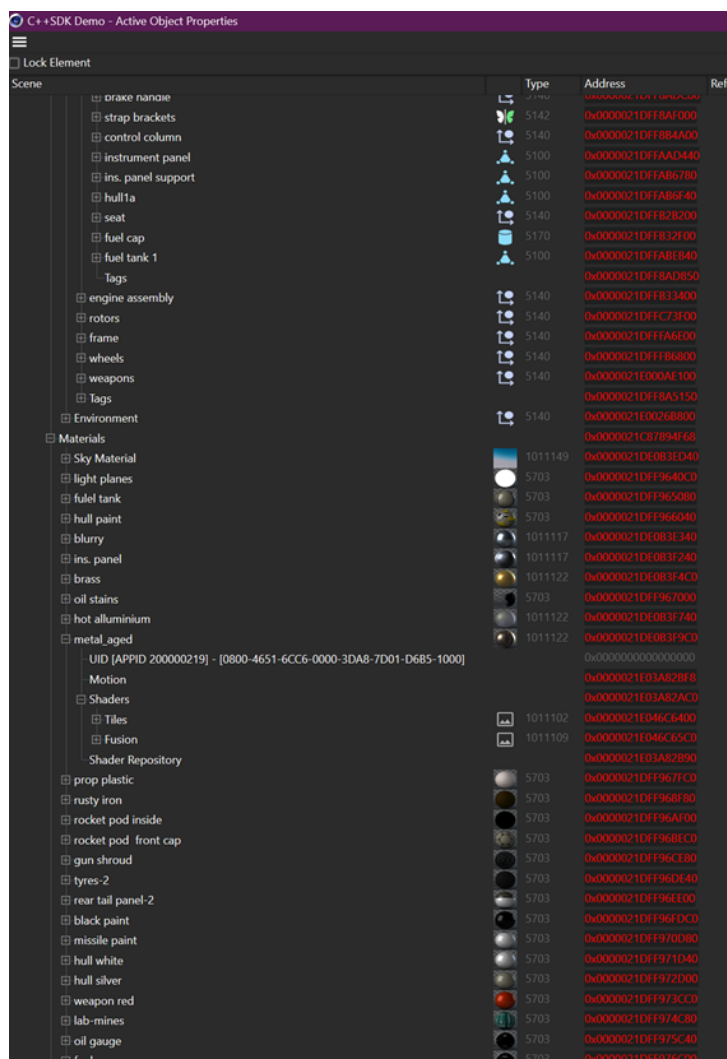


Figure 1. Active Object Dialog

As you can see, this can be a large and very long list and only a small part is shown in this image! Looking through it manually for the relevant ID value - there is no search facility - could well take a long time and some instances could be missed. Unfortunately too, the Active Object Dialog does not let you delete the missing object(s) once they are found. If you don't have this plugin, try searching on Google. You can enter 'cinema 4d plugin ' plus the name (failing that, the ID value) and see what happens. This may, or may not, yield useful results. If it doesn't you're stuck - but read on.

### Example 1

A real-world example may be helpful. Recently, I opened a very old scene and got the message about two missing plugins. This is what I got:

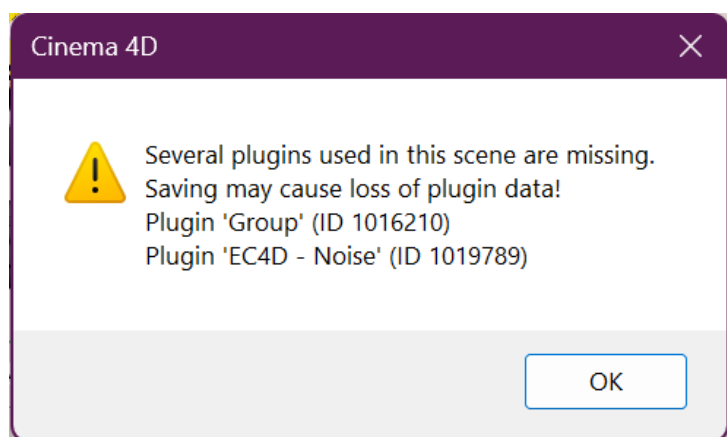


Figure 2. Missing plugins message

As it happened, I knew what one of these was. 'EC4D' was a collection of very good shaders for Cinema, sadly no longer available. So at least I could look for that one in the materials, except that this scene had 38 materials, each with 10 channels which had a link for a shader. The prospect of searching for that shader manually in each material wasn't a happy one, and it would be necessary to look in each channel, even the inactive ones. So to find it I looked through the shader list for each material in the Active Object Dialog then looked in the material itself to find the shader and remove it, which took time but was certainly quicker than a brute-force search through all the materials. Fortunately, the AO Dialog showed the one(!) material out of the 38 which contained the shader, though not which channel it was in.

The other plugin was unknown to me. The name 'Group' isn't helpful, but searching Google with the ID number showed that it was in fact the 'Group' tag from the Riptide Pro plugin, which was a much better importer for .OBJ files than Cinema itself, but which doesn't seem to have been available since C4D R14. So that's fine...except that this scene had over 500 objects, many of them nested deeply in object hierarchies, and trying to find that missing tag, possibly attached to many objects, was even less appealing than trying to find the shader.

A Python script to iterate through all objects or materials, look for the tag/shader and delete it if found, was the simplest solution, but not very flexible. What if in another scene it was an object or videopost plugin which had to be removed? The script would have to be rewritten. A better solution was a full plugin with options to make it as general and flexible as possible.

Using Missing Plugin Remover (henceforth 'MPR'), you can try to identify what a plugin is, so you can then list the instances of it and/or delete them. MPR can search for the four commonest plugin types, namely object, tag, shader and post-effect plugins. If it can't find the specified plugin in the scene, it will ask the Cinema 4D API to identify the plugin type. This is slightly quirky but sometimes returns a helpful answer.

Once the plugin is identified, you can use an action called 'Generate List' to list all the instances of the plugin in the scene, so if you have 20 materials and three use the shader plugin, those three are listed in the interface. The list generation is entirely optional - you can go straight to the removal phase if you wish - but it is perhaps sensible to look at what you're about to delete before doing so!

Finally, if you are happy to delete all the plugin instances, you can use another action to remove them and all the instances of the missing plugin will be stripped from the scene file.

## How Missing Plugin Remover works

MPR is invoked from the 'Extensions' menu. This will open a dialog box with this interface:

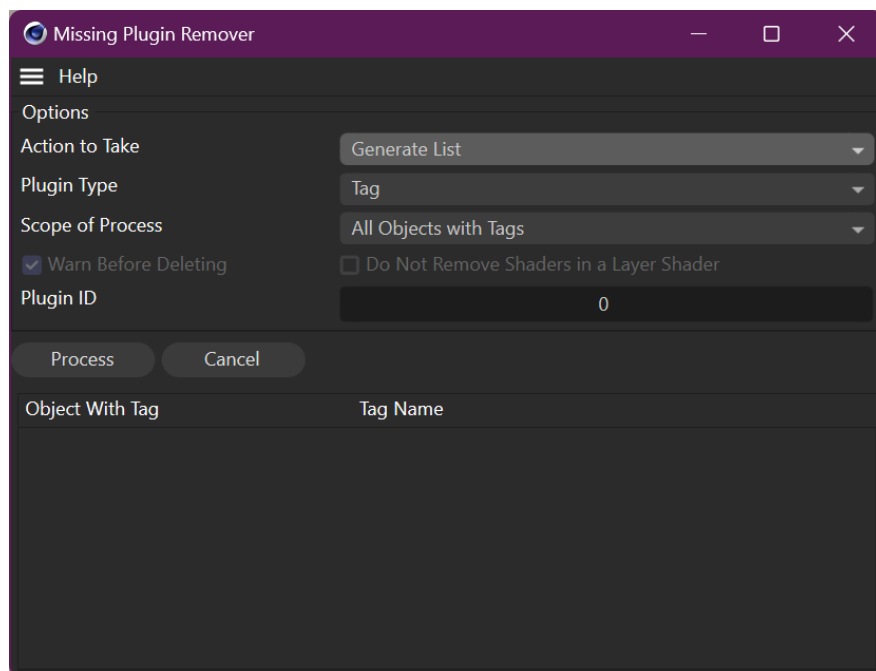


Figure 3. Missing Plugin Remover interface

What it does is determined by the drop-down menu 'Action to Take'. This has three entries:

- Identify Plugin
- Generate List
- Remove Plugin

For all these options, you will need one essential piece of information: the ID value of the plugin you are interested in. If we stick with the default 'Generate List' option for the moment, the ID should be entered into the 'Plugin ID' field.

### Example 2

For this example, starting with an empty scene, change the 'Plugin Type' menu to 'Shader'. In the 'Plugin ID' field, enter the number 1011116. This is the ID value for the Cinema 4D Noise shader, which will certainly be loaded on your system (which is not the same as being used in a scene - loaded just means that it is present and ready to use). Leave the other menus at their defaults of 'Generate List' and 'All Materials', then click the 'Process' button. This is what you see:

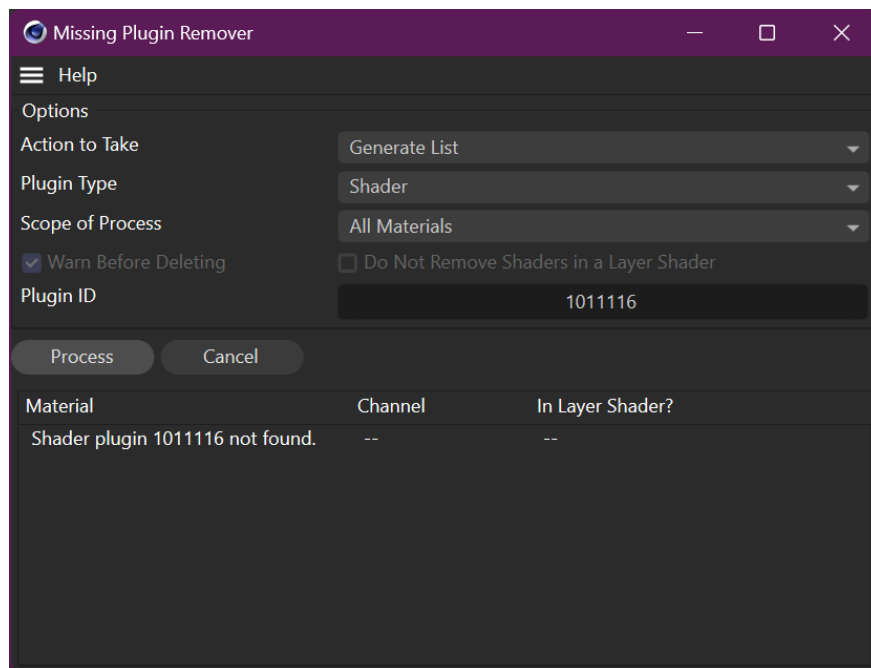


Figure 4. Finding the Noise shader

This is telling you that there were no shader plugins with that ID found. This is as expected: the plugin searches through every material in the scene file for a shader with that ID value, but since there are no materials in the empty scene, there can't be any shaders.

To fix this, create a couple of materials and rename them to something else to distinguish between them. In one, add a Noise shader to a shader link in some channel - Color will be fine. In the other shader, add a Layer shader to a shader link in a different channel and in the Layer shader add a Noise shader. Then click 'Process' again. You see this:

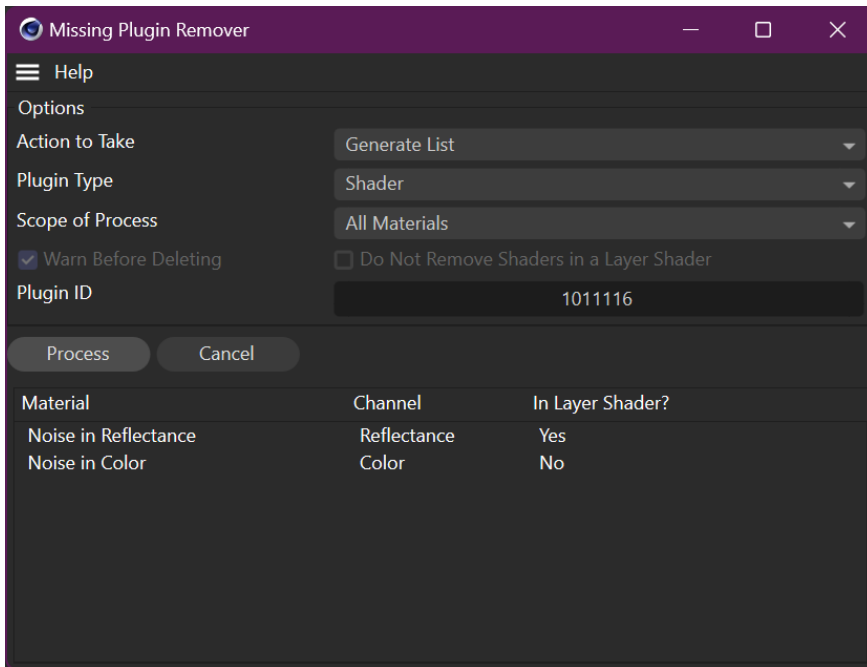


Figure 4. Finding the Noise shader in different materials

You now see that both materials are listed because both contain a Noise shader. You also see the channel the shader is in (if the material has a Noise shader in two channels, you would see two entries for that material, one for each channel with the shader). There is an additional piece of information: is the Noise shader located within a Layer shader or not? You see this in the list column 'In Layer Shader?'. This is important, as you will see below. Note that, if the Layer shader contains more than one Noise shader, you see an entry in the list for each one.

Now change the 'Action to Take' menu to 'Remove Plugin' and click 'Process' again. What you see now is that the Noise shaders are all removed from the two materials.

One refinement is that if you double-click the material name in the list, that material will be automatically selected for you in the material manager - which is very useful if you have a dozen materials all named 'Mat'!

In the above example, we already knew that the plugin was a shader. But what if all we had was the information in the message displayed when the file was loaded? That gives no indication of what type of plugin the missing one is, or where to find it in the scene, and we need to know that before we can remove it.

There are several ways to solve this. We could, for example, try to generate a list of the plugin instances, repeating for the four main plugin types given in the 'Plugin Type' menu. This would work but is rather long-winded and only works if the missing plugin is one of those four types. A better way is to use the 'Identify Plugin' option which you find in the 'Action to Take' menu.

### Example 3

Starting with an empty scene and with 'Identify Plugin' selected, enter the ID of the Noise shader again (1011116). Then click the 'Process' button. What the plugin will do is search for that ID value in all four of the plugin types it recognises, and of course it will fail to find it, because the scene is empty. Then, it will use a Cinema 4D SDK function which returns the type of any plugin with a valid ID, provided that - and this is important - the plugin has been loaded into Cinema, even if it is not in use. In this case, this is what the plugin will report:

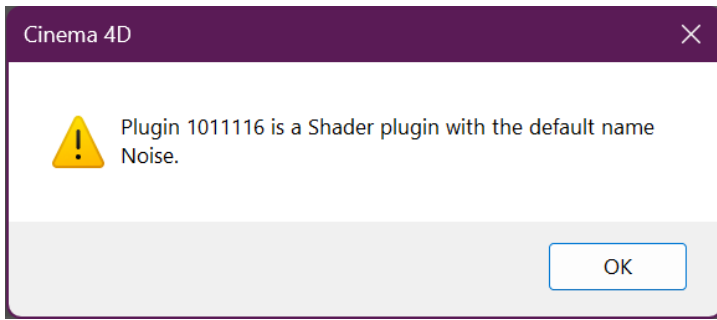


Figure 5. Identifying the Noise shader

This happens because the shader is not used in the empty scene, but it has been loaded and registered with Cinema when the app was run. If the plugin has not been loaded, which a missing plugin will not, then there is nothing for Cinema to report. If we enter the fake plugin ID 1234567, we get this:

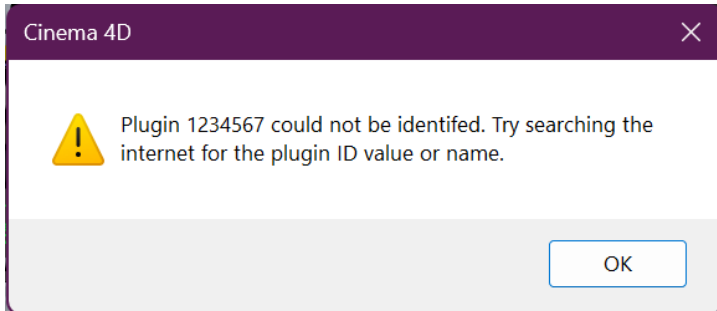


Figure 6. Failing to identify a plugin

This just means that the plugin has not been loaded and that there is no scene element containing a placeholder for the missing plugin. Under those circumstances all you can do is try to find out what it was by searching online or by trawling through the Active Object Dialog if you have that.

There's one final point: this plugin only recognises the four main plugin types. There are other types of plugin which you may see but which will not appear in any of the generated lists, such as scene hook or message plugins. Fortunately, these are rarely, if ever, standalone plugins like a shader or object, but are usually found as part of a larger plugin suite (and part of the same plugin binary file). If they are present in a scene file, the SDK function referred to above will report their type, but note that the results from that function can be a little quirky as shown in the next example.

#### Example 4

Again, starting with an empty scene enter the ID value 5159 and switch to 'Identify Plugin'. This is what you will see:

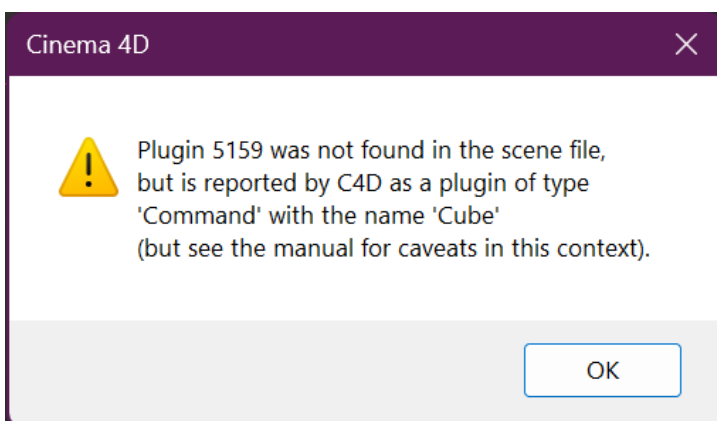


Figure 7. Identifying the Cube primitive in an empty scene

The value 5159 is for a primitive Cube. Add a Cube to the scene and identify the plugin again. Now you see this:

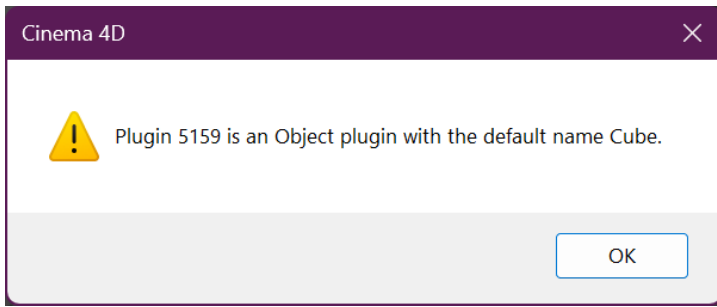


Figure 8. Identifying a Cube primitive when one is in the scene

Why the different reports? If there is a Cube in the scene, it will be found when MPR trawls through the objects in the object manager, and will report (as expected) that yes, 5159 is a Cube. But if there is no Cube in the scene, the SDK function is called and says that this is a Command plugin with the name 'Cube'. Technically, that is correct - when you insert a Cube into a scene, you are calling a command to insert a Cube primitive. Note that this happens only with the inbuilt objects. With plugins (and a lot of what we think of as simply part of Cinema, such as MoGraph, are in fact plugins like any other) it works as expected.

There is another quirk here, too. Add a Cube to an empty scene and it will automatically have a phong tag. Now in the attribute manager for the tag, switch to the Basic tab and rename the phong tag to something else - doesn't matter what. I used 'My\_Phong' instead of the default 'Phong'. When using 'Identify Plugin' with the ID value 5612, which is the Phong tag, it returns this:

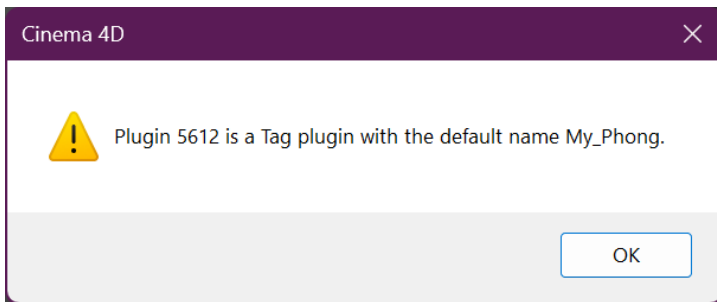


Figure 9. Incorrect default name returned for some tags

The default name is shown as 'MY\_Phong' which is wrong - it should be 'Phong'. What appears to happen is that, for inbuilt tags only, the SDK function reports the actual name of the tag, not the default name. With third-party tags it works correctly, as it does for other inbuilt elements including shaders, objects and post-effects.

Why the difference between inbuilt objects and third-party ones? And what is the deal with inbuilt tags? I don't know but it probably goes back to the very oldest code in Cinema and there's never been a reason to change it. And it only seems to apply to those objects which form the very 'core' of Cinema. If you try this with a Mograph Cloner, for example, you get exactly the same results as with a third-party object. The message is this: be aware that the SDK doesn't always report what you might expect it to.

## Removing the missing plugin

Once you know what the plugin type is, you can proceed to remove them, but it is strongly recommended that you generate a list of plugins to remove first. There are two reasons for this. Most importantly, it's sensible to know what you are about to delete before doing so, and this is especially important for object plugins. If you delete an object, and it has child objects, you will delete all those as well. So it would be safer to check each object in the list to make sure you know what will be deleted. If there are child objects to preserve, you will have to do that manually, by (for example) creating a null object then moving the child objects of the object to be deleted to be child objects of the null instead.

The second reason is that objects, materials and even videotests can be renamed, which can lead to scene elements being deleted unexpectedly, as shown in the next example.

### Example 5

Consider these render settings:

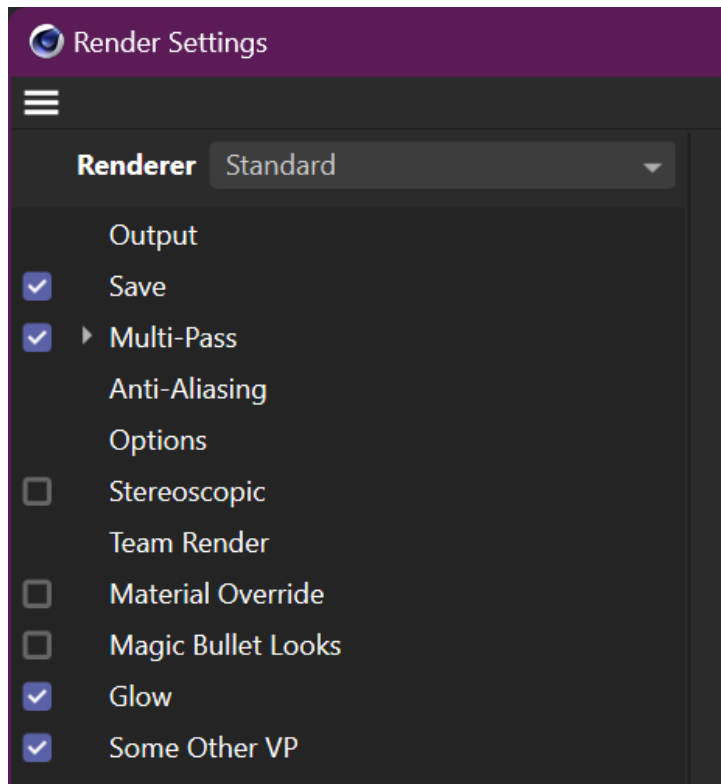


Figure 10. Renamed post-effect plugin

You can see that there are two post-effects which have been added - 'Glow' and 'Some Other VP'. The ID value for 'Glow' is 1001401, and if you delete plugins with that ID, you might be surprised to see both the post-effects being deleted. This is because 'Some Other VP' is also an instance of the Glow post-effect, which has been renamed. If perhaps you only want to delete 'Some Other VP' and leave 'Glow' alone, you can do it manually in the render settings dialog.

## How to use Missing Plugin Remover

### Step 1. Load the scene file with the missing plugin

Once loaded, make a note of the name and ID of the missing plugin(s) as you will need these.

### Step 2: Identify the missing plugin

If you already know what sort of plugin this is, you can skip this step and go to step 3. Otherwise, choose 'Identify Plugin' from the 'Action to Take' menu, enter the ID value in the 'Plugin ID' field and click 'Process'. With luck, you will see a message indicating that the offending plugin is one of the four types MPR can handle. You might see a message that it is some other kind of plugin, such as a Scene Hook or Message plugin. If it is, MPR cannot progress any further and you would need to search online to try to find out what it is.

### Step 3: List the instances of the plugin

This step is optional but strongly recommended before deleting anything.

First, change 'Action to Take' to 'Generate List', then change 'Plugin Type' to the correct type - tag, shader, etc. This is essential: if you specify the wrong type, MPR will simply report that the plugin could not be found. Then you can change the 'scope' of the process, if desired. This is explained more fully below.

Now click 'Process'. This will generate a list of all the plugin instances according to the criteria you just entered.



#### **Step 4: Remove the instances of the plugin**

Change 'Action to Take' to 'Remove Plugin', leaving the other parameters as they were, and click 'Process'. All the instances of the plugin will be deleted, unless you narrowed the scope of the process, in which case only those falling within that scope will be removed.

## **Reference**

### **1. Menus**

There is one menu, 'Help', with the following entries:

#### *i) About....*

Displays a message box with the version number and copyright.

#### *ii) Show Help File (.PDF)...*

Opens this manual, provided that the manual is in the same folder as the plugin binary file.

### **2. Action to Take**

This drop-down menu indicates what action will be carried out when the 'Process' button is clicked. The options are:

#### *i) Identify Plugin*

The other two drop-down menus will be grayed-out when this is selected. MPR will try to identify the plugin whose ID value is in the 'Plugin ID' field, as described above. If that plugin is not present in the scene file, or is not one of the four plugin types MPR recognises, the SDK function will be called to try to identify it.

#### *ii) Generate List*

MPR will generate a list of all the instances of the object. Before clicking 'Process' you MUST first change the 'Plugin Type' drop-down menu to reflect the correct plugin type. In other words, if you want to list instances of a tag plugin, 'Plugin Type' must be set to 'Tag'. If it is set to anything else, the list will not be generated and the list field will display a message indicating that the plugin could not be found.

The information you see will vary depending on the plugin type. For objects and post-effects you see only the name. For tags, you get the name of the object holding the tag and the name of the tag itself (the actual name, because tags can be renamed). For shaders, you are shown the name of the material the shader is in, the channel it is in (Color, Bump, etc.) and - very importantly - whether the shader is contained in a Layer shader.

Once the list has been generated, for all objects except post-effects, double-clicking an entry in the list will automatically select the relevant object (for objects and tags) in the object manager, or material in the material manager. This makes it easy to find that object and inspect it if required before deleting the object/tag/shader. For post-effects this isn't possible and you will have to open the render settings and find the plugin there.

Note that in the case of shaders and post-effects, the plugin will still be found even if the renderer with the post-effect is not the active one. For example, if you add the post-effect 'Glow', which is available in the standard renderer, then switch the render engine to Redshift, where Glow is not available, MPR will still find the Glow effect and you would need to switch render engines to see it. Note that MPR can identify, list and remove Redshift post effects.

The same would be true for standard C4D materials in a scene using Redshift - the plugin will still be found and can be removed. What MPR cannot do, however, is find and remove C4D shaders present in a Redshift material. This is currently a limitation.

### *iii) Remove Plugin*

The plugin will attempt to delete all instances of the specified plugin that it can find. As with 'Generate List' it is essential that the 'Plugin Type' menu is set to the correct plugin type first; trying to delete a tag when the plugin ID value is for a shader won't work.

### **3. Plugin Type**

This menu indicates which type of plugin will be listed or removed. This must be set correctly so that the plugin ID value is of the correct type. That is, if the plugin ID is for a post-effect plugin, this menu must be set to 'Post Effect'; if it is not, neither listing nor removing the plugin will work.

The four entries in this menu are the four plugin types which this plugin can list and/or remove.

### **4. Scope of Process**

In most cases you would probably want to remove all instances of a plugin but this menu gives the option of restricting the action to only some instances. It is applicable both when listing a plugin or removing it. The menu varies depending on the plugin type selected, and the options, starting with tag plugins, are:

#### *i) All Objects With Tags*

This will list all the objects with the specified tag plugin, or will remove the tags from all such objects.

The alternatives are (depending on the plugin type):

- All Materials: for shader plugins
- All Objects: for object plugins
- All Videoposts: for post-effect plugins

#### *ii) Tags on Selected Objects and Children*

Tags on any objects selected in the object manager and on any child objects of those selected objects will be listed or removed.

The alternatives are:

- Selected Materials Only: for shader plugins
- Selected Objects and Children: for object plugins

#### *iii) Tags on Selected Objects Only*

Tags on any objects selected in the object manager (but not on any child objects of those selected objects) will be listed or removed.

The alternatives are:

- Selected Objects Only: for object plugins (but see the note below)

In the case of object plugins, choosing one of the selected objects options should be used with caution. Suppose you have a simple hierarchy like this:

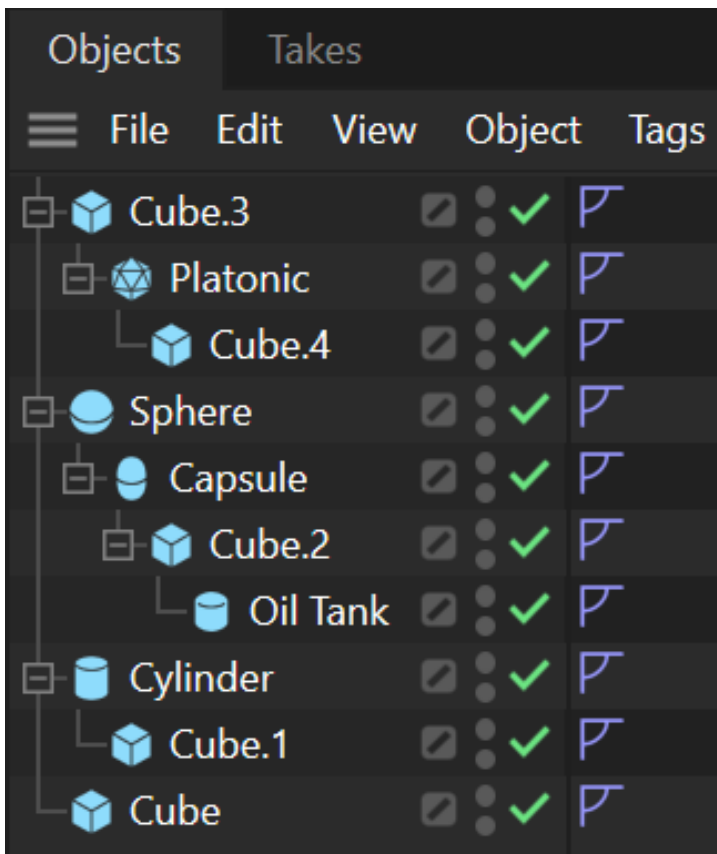


Figure 11. Object hierarchy with several Cube objects

If you enter the plugin ID for a Cube (5159), and choose 'All Objects' in the scope, all the Cube objects will be listed and if you opt to remove them, all will be deleted. However, you will also lose any child objects of these Cube objects, leaving you with this:

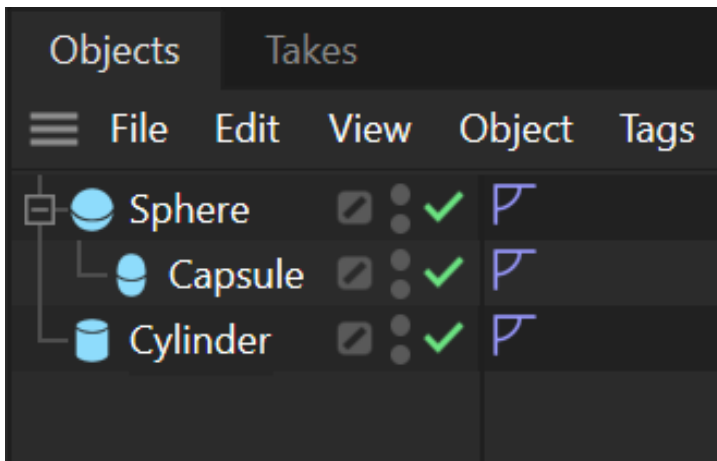


Figure 12. Object hierarchy after all Cube objects removed

That's straightforward and as expected. Now suppose Cube.3 is selected in the object manager and you choose 'Selected Objects and Children' as the scope. This will list Cube.3 and Cube.4, and those will be deleted if you opt to remove the plugins. But now, suppose Cube.3 is selected and 'Selected Objects Only' is the scope. The list will only contain 'Cube.3' because child objects are ignored. However - and this is the catch - if you remove Cube.3 you will also lose Cube.4, because deleting an object will automatically delete its child objects as well. Note that removing only Cube.4 is not a problem, because you can select that in the object manager and only it will be removed, as it doesn't have any child objects.

The message here is that, with object plugins where there is a hierarchy of objects in the object manager, you may remove more objects than intended. If, in this case, you really did want to remove only Cube.3 and keep the child objects, you would have to do it manually - say, by creating a Null object, moving the child objects of Cube.3 to be child objects of the Null, then deleting Cube.3.

## 5. Options

There are two additional options. These are:

### *i) Warn Before Deleting*

This option is turned on by default. When you opt to remove a plugin, you will be asked to confirm that you really want to do so. Click 'Yes' to carry out the deletion or 'No' to abort the process.

### *ii) Do Not Remove Shaders in a Layer Shader*

This is only applicable to shader plugins. When turned on, any shader plugin within a Layer Shader will not be removed and they will remain listed in the list box. You can then navigate to the material by double-clicking the material name in the list and remove the plugin entry in the Layer shader. This is to avoid the replacement of a shader in a Layer shader with the 'empty' placeholder (see 'Limitations' below).

## **Plugin ID**

In the field you should enter the unique ID value of the plugin to be removed. If you enter an invalid value you will be informed that a plugin with that ID could not be found. MPR has no way to know whether any ID value is valid or not, since new plugins and components could be added at any time with their own unique ID.

## **List box**

This is the box in which the plugins of the specified type will be listed, if there are any in the scene. For tags and objects, double-clicking the object name will select the object in the object manager. For shaders, double-clicking the material name will select the material in the material manager.

## **Known limitations**

You should be aware that for shader plugins, the plugin might be contained in a Layer shader. MPR will still delete them, but it does not remove the entry in the Layer shader. Instead, Cinema leaves behind a placeholder named 'empty' which returns a black colour. This is likely to cause a significant change to the output of the material. You would need to look through the Layer shaders and remove this placeholder manually because the placeholder will not return the ID of the deleted plugin. For this reason, there is an option not to remove shader plugins in a Layer shader. If this is turned on, no shader plugin within a Layer shader will be removed. The shader plugin will still be listed in the list box and you can go to each one in turn and remove the plugin manually, avoiding the 'empty' placeholder Cinema would otherwise insert.

For objects and tags, the render engine used in the scene is not relevant and MPR will work in the same way regardless of the renderer. It should also work correctly for videopost plugins but this has only been tested with Redshift. However, for materials MPR only works for the standard C4D materials. It cannot detect or remove shaders in Redshift materials or those from other renderers.

MPR does not currently handle bitmaps as opposed to procedural shaders. There is the Project Asset Inspector in Cinema which should find missing bitmaps, so this functionality has not been reproduced in MPR.

## **Conclusion**

That's all there is to it. MPR grew out of the need to remove missing plugins from older scenes but which is flexible enough to remove any of the four main plugin types from a scene. I hope you find it useful. If you have any comments, feature requests or bugs to report, please contact me through my website at <https://www.microbion.co.uk/html/contact.htm>

Steve Pedler  
December 2025