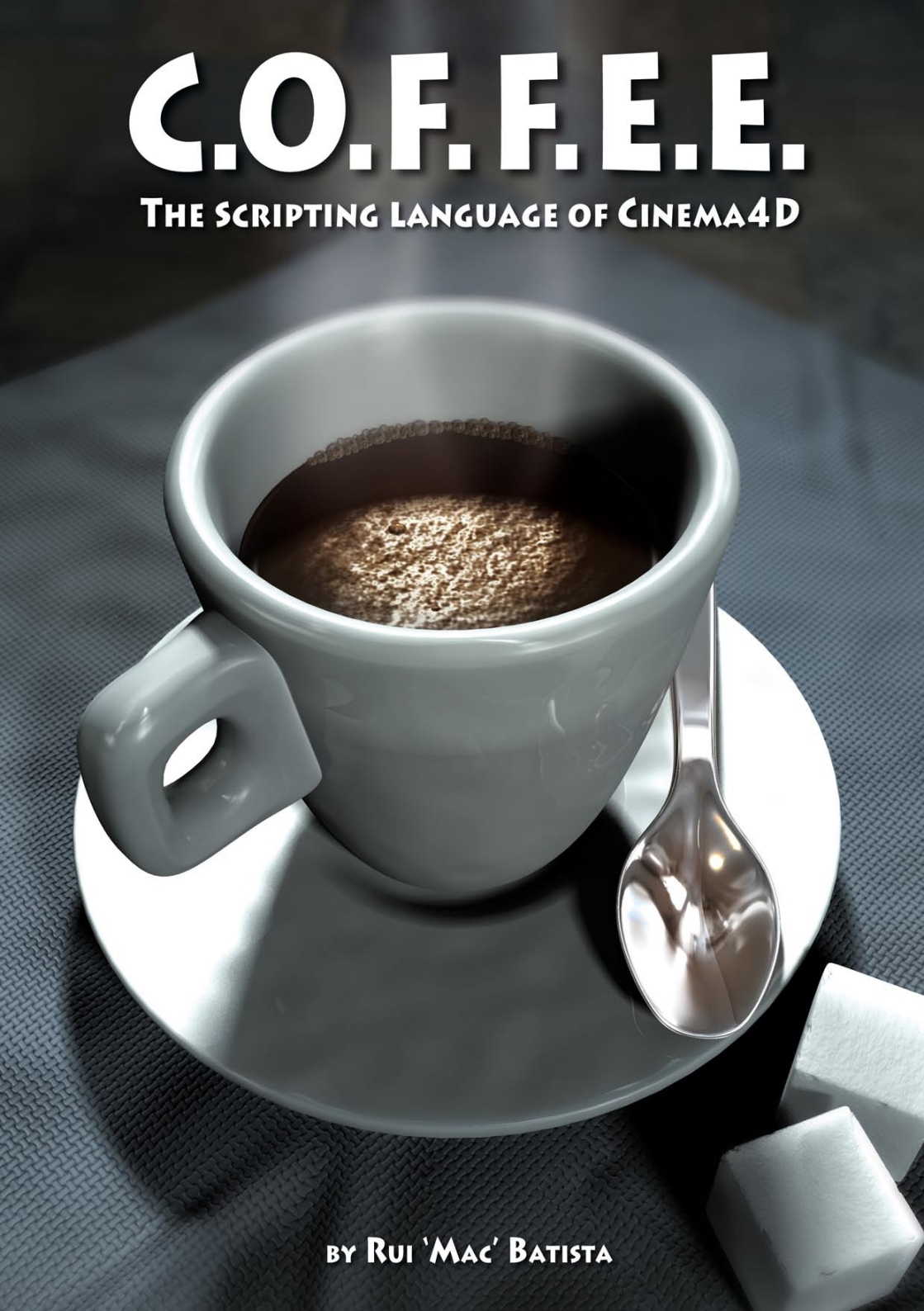
C.O.F.F.E.E.

The Scripting Language of Cinema4D



by Rui 'Mac' Batista

Yup, left blank on purpose.
You can use it to draw whatever you want :-)

Chapter 1

The task I have assigned myself is not an easy one; teach C.O.F.F.E.E.

Not the beverage of course, but the scripting language included in CINEMA 4D. And why is it a daunting task? Because, usually scripting, or anything related to programming, is considered a subject for geeks, chess club stars, rocket scientists and non-artistic people in general. This couldn't be farther away from the truth. I'm not a chess club star, although I do know how to play chess. I'm not a rocket scientist, but I did graduate as a graphic designer and I do know how to code in C.O.F.F.E.E.! OK, I never said I wasn't a geek, but even if many people consider me as such, I don't think I'm one of those :)

What exactly is C.O.F.F.E.E.? It's a scripting language. OK, this is not very meaningful to all you mouse-drag-mouse-click-doodle-stuff-on-screen guys, is it? Instead of "scripting language", let's say it is a programming language. Wow! Now instead of something that you don't understand, it became something you fear. Let me assure you, there is no reason to be afraid. I intend to make this a fun experience from beginning to end. Now that you understand C.O.F.F.E.E. is a programming language, you can, and will, use it to make CINEMA 4D do stuff that would be, otherwise, very hard or completely impossible to do by hand, mouse, pen or whatever.

I have no doubt one of the first questions you "artists at heart" asked yourselves when you first discovered the C.O.F.F.E.E. language;

"Why the hell is it called C.O.F.F.E.E.? Do you need industrial amounts of C.O.F.F.E.E. to learn and tame it? Like the beverage, is it a "black" art?"

I don't have official confirmation from MAXON, but I assume, and please DON'T quote me on this... I will vehemently deny it, that it is because it is very, very, very similar to JavaScript, and java is a type of C.O.F.F.E.E..

Oops, I said another "foreign word", JavaScript. JavaScript is another programming language often used to code stuff, mainly for Internet pages. It is a very structured programming language based on C and C++. Yes, you got it... C and C++ (pronounced "see" and "see plus plus") are also programming languages. As you may have already guessed, there are many, many (a few hundreds, if not thousands of) programming languages. Until now, to you, all of them sounded like the forbidden Black Language of Mordor, (sorry for the Tolkieneske pun) but I promise I will try to make, at least, C.O.F.F.E.E. almost as easy to understand as plain English. If you already know a bit of JavaScript, Java, C or C++ you already know a bit of C.O.F.F.E.E. Believe me, they are almost the same. If you are still a "virgin" at this matter, do carry on reading.

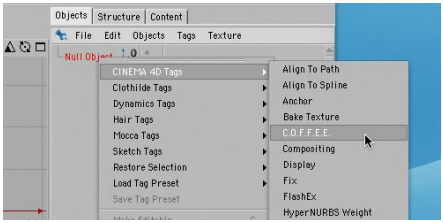
OK, what exactly is a programming language? It's a set of instructions (commands, functions, operands, operators, etc) that, assembled according to a specific syntax, instruct the computer (or, in our case, the CINEMA 4D application) to do stuff. So, what are "commands", "functions", "operands", "operators" and that "syntax" I talked about? Commands are exactly what their designation mean: they are "words" that make the application actually perform an action. Functions are like magic boxes that you place stuff inside and something different comes out. This means that you feed them with data and, once some operations are performed on that data, some sort of result will come out. Operands are the data you work with. It can be a number, a word, a 3D primitive, a texture, etc. Operators are operations you can perform on operands (the data, remember?) The "syntax" is the set of rules you must follow to write correct code, just like the rules of the syntax of your mother language if you want to write correct sentences. I'm telling you all this because, inevitably, I will have to introduce you to some programming-specific terms (sorry, there is no escape from that my friends). Even if they sound complex, cryptic or just plain weird, they will become obvious and even logic after a while.

Let's start with our first program in C.O.F.F.E.E. and, with it, I will introduce you to all the concepts I presented you with before.

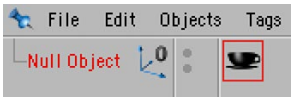
This first program will simply print out the sentence "Hello World". Why?

Well, I can't really explain why but all tutorials about every programming language, whatever they are, start with a simple program that prints out "Hello World". I guess it is because all programming languages have, at least, a command that prints something and, this way, people can get a first "test-drive" of a programming language in a very soft and easy way. Otherwise, they would give up at their first try.

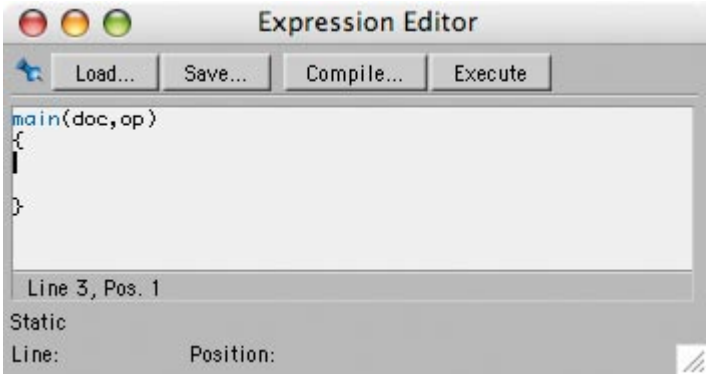
To create your first C.O.F.F.E.E. script you need to add a new C.O.F.F.E.E. tag to an object. It can be any object you want but, for this purpose, let's create the simplest object there is: a Null. Now that you have a Null, add to it a C.O.F.F.E.E. tag. You do that exactly the same way as you add any other tag.



Once you add the C.O.F.F.E.E. tag, you end up with a nice cup of C.O.F.F.E.E. in the Object Manager and, automatically, CINEMA 4D opens a C.O.F.F.E.E. Expression Editor. Even if you close this window, you can always get it back by double clicking the C.O.F.F.E.E. tag attached to your object(s).



As you can see, the C.O.F.F.E.E. Expression Editor has already "typed" some code for you:



In this case, it defines your main function. This is a very special kind of function because it is the first function that is executed. All C.O.F.F.E.E. expressions must have, at least, a main function. Otherwise, you would get an error and nothing would be executed. This function doesn't return any value, actually, (remember my previous definition of function?) but it does include two operands: doc and op.

What are those? Well, they are values that you can use inside the function. Let me give you an example of another function that you may be more familiar with: The power of two function. If you recall your math in school, you can raise any number to the power of two. It's as simple as multiplying the number by itself. So, for instance, 3 raised to the power of two is 9... or 3 x 3. The same way, 5 raised to the power of two is 25... or 5 x 5. Easy, isn't it?

So, let's give this function the name **raise_to_power_of_two**. For it to make its mumbo-jumbo (multiply a number by itself) we must feed it with a number, right? It will then perform whatever operations are needed and spits out a result. The number we feed it with it called an operand. So, we could code the **raise_to_power_of_two** function like this:

```
raise_to_power_of_two(x)
{
return x*x;
}
```

The value that we placed between parentheses, after the name of the function, is the operand that the function will use. Why can't we place a number there? Well, we are just defining what the function does, not really performing any calculation. If we placed, for example, the number 3 there, this function would always calculate 3 raised to the power of two. Not very useful, you must agree.

In this case we used a letter **x**. It's a variable. Why is it called a variable? Well, because it can contain any value.

So, inside the function – the commands between the `{` and the `}` – we calculate **x** times **x** and we return that value. That is exactly what the command `return x*x;` does. I believe that `return x*x` is obvious enough but... what about the `;` at the end? All commands in C.O.F.F.E.E. must end with a semicolon. This is a way for C.O.F.F.E.E. to know when a command as actually finished being defined. This is one of the rules of the syntax of C.O.F.F.E.E. See? You have already learned what a function and an operand are. You also had a little glimpse of what a variable is, and you had a little taste of what "syntax" really means.

We have now seen the `{` and `}` symbols twice and I haven't explained exactly what they are or mean. They refine a block of code. Everything between them defines a set of code instructions that relate to each other. The logic of using them in a function is that, after defining the name and operands of the function, everything between the `{` and the `}` symbols belongs to that specific function. This means that, right after the name of the function (and operands, if any), C.O.F.F.E.E. encounters a `{` symbol. This defines the start of the code of the function. When it finds the correspondent `}` symbol it "knows" that the function ends there. Logic, isn't it?

Now that we know how in interpret the:

```
main(doc,op)
{

}
```

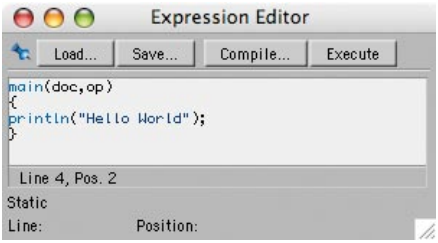
...Let's understand what the doc and op are. From what you learned already, you know they are the operands of the main function, right? And, also, that they are variables. Before explaining exactly what each one is, I must clarify what a variable is. Imagine a variable as a storing box. Inside it you can place values. And what type of values? Well, pretty much everything. You can store numbers, letters, colors, objects, tags, materials, etc. When I say objects, tags, materials, etc. I mean, a value that points to an actual object, tag, material, whatever, inside your document. So, variables are invaluably useful. Without them you would be able to do very little with C.O.F.F.E.E., or any other programming language.

So, if variables can store so many things, what do the doc and op variables store? CINEMA 4D politely provides you automatically with the current document (the one you are working with) in the doc variable and the object that contains the C.O.F.F.E.E. tag in the op variable. This means that as soon as you enter the main function you can access info about your document and also info about the object that contains the C.O.F.F.E.E. tag whose main function is being executed. This is very valuable information but in this first chapter we will not need it because we only want to print a simple sentence. No fiddling with documents or objects is required for now.

If you paid lots of attention to all that you have read here, you may be wondering why is it that the `main(doc, op)` doesn't have a `;` at the end. If you asked that question, you are my best student so far!!

Well, because `main(doc, op)` is not a command. It doesn't instruct the computer to do anything. It just defines a function. It's an instruction, not a command. Remember, I said that only commands require the `;` at the end. Syntax rules, you know? ;-)

Place your cursor between the `{` and the `}` symbols and type:



Notice that the **ln** after the **print** are lowercase **L** and **N**, not an uppercase **i** and lowercase **N**.

What have we just typed? The `println` is a command that instructs CINEMA 4D to print out something. Between the parentheses are its parameters, as in what you want to print. And why is it between quotes? Well, because what we want to print is a literal expression, not a variable. Ok, ok... I'm talking gibberish again. What is a literal expression? Well, it's something that should be interpreted as is! Want an example? Ok, no problem...

Imagine you had written

```
println(hello);
```

instead of

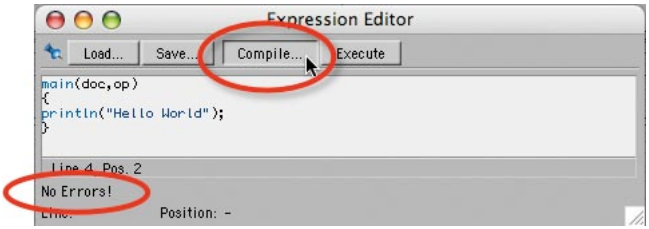
```
println("hello");
```

And you had a variable named **hello** that is storing the word **goodbye**.

Should CINEMA 4D print **hello** or the content of the variable **hello** that is, actually, **goodbye**?

To not confuse CINEMA 4D, we must enclose all literal expression (as in, they should evaluate exactly as they are presented) in quotes. So, `println(hello);` would print **goodbye** because, since **hello** is not enclosed in quotes, it is evaluated as a variable. The command `println("hello");` will print **hello** because, since it is between quotes, it should be taken as it is written. This is pure syntax at work. See how important the syntax is?

Ok, you just wrote your first C.O.F.F.E.E. script. You must now check if it has any errors. To do so, click the **Compile** button on the top of the Expression Editor window. If it all goes fine, you should get a report of **No Errors!** at the bottom of the Expression Editor window.

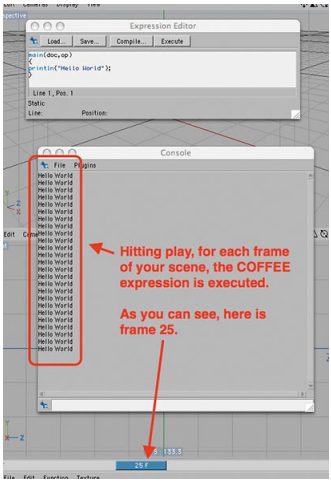


Now, hit the **Execute** button at the top of the Expression Editor window.

Wow! Amazing! Nothing happened! This Rui guy is a charlatan!! Wait... the printout is in there... somewhere. You just need to know where to look for it. If you are using an 8.x version of CINEMA 4D, press Shift+F10. If you are using a 9.x version of CINEMA 4D or higher, press Alt+F9 or simply choose **Console** from the Window menu, in any version of CINEMA 4D.

Now you see your **Hello World** at the bottom of the window? Ah, this Rui guy is not a charlatan, after all. Do you see more stuff printed in the Console window? If you do, that is all the stuff that plug-ins print when they load. Actually, the Console is a very useful place to go to check to see if something is wrong with any plug-in that is not loading or simply misbehaving.

Now, from the File menu of the Console window, choose Clear. You are now staring at a clean Console window. With the Console window still open, press Play, like you would do to check out an animated scene in CINEMA 4D. Wow, lots and lots of Hello World sentences! You now know that your script is executed for each frame of your animation. Actually, it is executed each time something changes in your scene. Choose Clear again from the Console File menu and try moving the Null around. Again, lots of Hello World sentences. This is not particularly useful right now, but it's good to know, for future reference, how often the C.O.F.F.E.E. scripts are executed.



One final thing before we wrap-up this first chapter about C.O.F.F.E.E. Why the hell is the print command named `println` and not just print? In this case, **ln** (lowercase **L** and **N**) stands for **line**. You are instructing CINEMA 4D to print out a line of text. A line of text means that a return is added at the end, automatically, just like if you had pressed Enter, otherwise it would print out all the Hello World sentences "glued" together, as in:

Hello WorldHello WorldHello WorldHello WorldHello WorldHello WorldHello World...

If you want to try it out, replace the `println` with `print`. As soon as CINEMA 4D fills out an internal container of the Console, it will print out a whole line of neatly glued together **Hello World**. You can force the print command to add a return, but why bother if you already have the command `println`, right?

This is it for this first chapter. I hope I didn't frighten you too much with all this code stuff. I also hope I was able to interest you enough to keep you reading on through the rest of the chapters. Let's move on to Chapter 2.

Chapters Index

Chapter 1 ..... **3**
Introduction to programing languages, focusing specially on C.O.F.F.E.E..
Presentation of some basic concepts related to programming languages.

Chapter 2 ..... **10**
Practical examples of some basic programming in C.O.F.F.E.E..
Introduction to variables, conditional and logical expressions.

Chapter 3 ..... **21**
Introduction to one of the more important concepts of C.O.F.F.E.E. programming: objects

Chapter 4 ..... **31**
Lots more about objects.

Chapter 5 ..... **43**
Learn how to use the SDK. Introduction to the concept of loops and cycles.

Chapter 6 ..... **53**
Fully working example of a script, using all the concepts learned so far
while introducing a few more.

Chapter 7 ..... **67**
Fully working example of a more complex script, using all the concepts learned so far.

Chapter 8 ..... **79**
Learn how to use Resedit, the official tool to create GUI dialogs for plug-ins

Chapter 9 ..... **91**
Fully working C.O.F.F.E.E. tag plug-in.

Chapter 10 ..... **103**
Complete and exhaustive explanation of the plug-in presented in the previous chapter.

Chapter 11 ..... **119**
Fully working C.O.F.F.E.E. menu plug-in with explanation of the whole code.

Chapter 12 ..... **129**
Wrapping up... Learn how to compile your plug-ins and hunt for bugs.