DeCaf 1.0

Introduction

Welcome to the wonderful world of programming.

Well, maybe I shouldn't be starting the text with this sentence. After all, if you want to learn DeCaf it is probably because you are NOT a programmer. And, for non-programmers, the "world of programming" is not a very attractive place.

I hope DeCaf will allow you to start giving your first steps in that — at least for now — "non-attractive" world of programming.

●

Nowadays the number of available computer programming languages is overwhelming.

And you have computer programming languages that suit all tastes, from the very powerful and complex to the very limited and simple. Unfortunately, they usually suit the tastes of... guess who? Programmers. What I mean by this is that there aren't many computer programming languages that are suited for people whose mere mention of the word "programming" induces spasms or severe rashes.

Generally, people that are not programming-biased tend to consider programming as something that requires huge amounts of mathematical knowledge, a very organized memory and lots of logical thinking. Well, all of these are useful but not required. At least not in large doses. For those non programming-biased people, any computer programming language looks or sounds like an extra-terrestrial gibberish. Because of that, some computer programming languages started emulating regular written English. Such is the case of AppleScript, the scripting language of MacOS© or ActionScript, the scripting language of Adobe Flash©.

Those languages allow a much more flexible construction of a program. They are also much simpler to read and code. If they are so much more accessible to the masses, why aren't all programming languages based on the paradigm of regular written languages? Well, those types of languages have their own drawbacks. They execute much slower and, usually, they are not as efficient or powerful as the other types of languages. That happens because they rely on a syntax that is much more ambiguous than a strict, rule-based syntax common to most programming languages. Also, the regular written language lacks the effectiveness and structural cleanness needed to define and manipulate certain types of events, algorithms or data structures.

Wow! I've already started using those alien gibberishes: Events? Algorithms? Data structures? These are terms common in most computer languages. But I really don't want to scare you, so I will refrain from using such terms. Well, at least for now.

As you may be wondering, DeCaf is a computer programming language that is based on regular written English. Kind of... I mean, it is much easier to read and understand than most other computer programming languages. But if you show it to an English teacher, telling him that it is something you wrote, he will promptly contact a mental health institute.

When I first started creating DeCaf I had several purposes:

- to create a language that would be simple enough to allow people that know nothing about programming start coding.
- instead of using abbreviations or mnemonics for commands and functions, like most other computer programming languages I wanted to use expressions similar to regular written English.

- to avoid strict and/or complex syntax, very common to most computer programming languages.
- to allow for flexible scripting with the possibility of using several variations of expressions for the same purpose.
- to allow the use of abbreviated expressions.
- to build an editor that would help users learn DeCaf while coding.
- to present very clear and informative error messages.

Fortunately, all of those purposes were fulfilled. Nevertheless, while creating DeCaf I stumbled upon several ambiguous and doubtful situations that forced me to make some surveys among non-programmers (potential DeCaf users) and make some decisions.

For example, I decided not to deviate too much from some common programming paradigms. This means that DeCaf is not like proper written English, like I said before, but it is a whole lot more comprehensible than COFFEE or C++ (the other two programming languages used in Cinema 4D).

Keeping DeCaf close enough to other programming languages — while still making it simpler — had two advantages: it was a little easier for me to code it and it served as a stepping stone for DeCaf users to be able to learn enough programming paradigms so that they could, in the future, evolve to more complex programming languages.

Let me show you some examples of what I mean. First, an example of the simplicity of DeCaf.

In COFFEE, if you want to create a new Null Object you need to use the following code:

```
main()
{
var null_obj,document;

document=GetActiveDocument();
null_obj=new(NullObject);
document->InsertObject(null_obj,NULL,NULL);
}
```

In DeCaf, to do the same you would use the following code:

```
create null
```

Simpler, isn't it? Almost everything — if not everything — has this degree of simplicity, specially when compared with its counterpart in COFFEE or C++.

Now an example of where I could not keep DeCaf away from regular programming paradigms.

In COFFEE (and in all other programming languages) we have a way to create cycles (groups of instructions that are executed a specific number of times). Actually there are many ways to create cycles, but one of the simplest is:

```
main()
{
var f;
for(f=1;f<=10;f++)
 {
 println("Iteration:");
 println(f);
 }
}
```

For the non-code-biased people (the majority of the readers of this manual) this snippet of code is as intelligible as the scribbles of a two year old child. I will not even try to explain what each line does (for that you can buy my book about the COFFEE programming language).

What the code does is to print the following in the Console window:

```
Iteration:
1
Iteration:
2
Iteration:
3
Iteration:
4
Iteration:
5
Iteration:
6
Iteration:
7
Iteration:
8
Iteration:
9
Iteration:
10
```

The same would be done in DeCaf with the following code:

```
variable f
repeat 10 times using f
<<
print "Iteration:"
print f
>>
```

The good news is that DeCaf would also do the same if you wrote:

```
var f
repeat 10 f
<<
print "Iteration:"
print f
>>
```

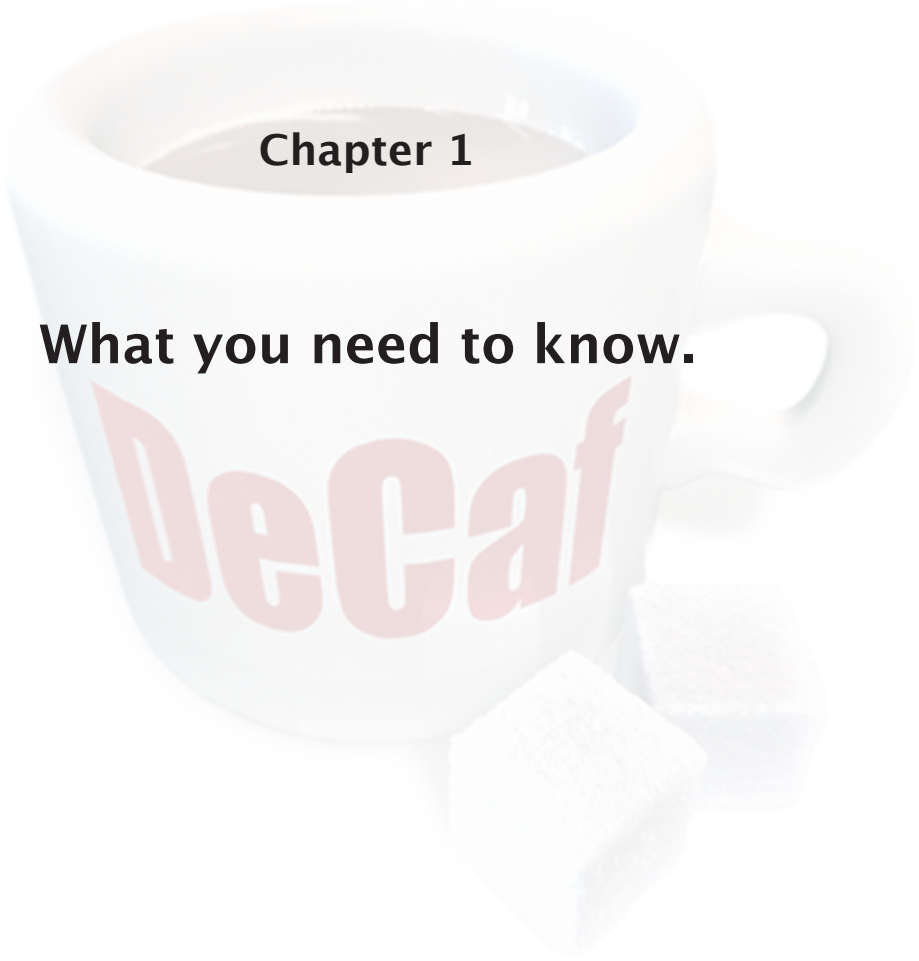
As you can see, DeCaf is versatile enough to allow for different versions of the same commands. You can type them in full, allowing for more clear reading and understanding of your code. But when you are already proficient in DeCaf you may very well use the shorter syntax.

Some commands can even be completely replaced by alternative synonyms. For example, **ask** and **input** will perform the same task.

You may be asking where, in the DeCaf listing, I could not escape from regular programming language paradigms. Well, in the COFFEE listing we define blocks of code by placing commands between **{** and **}**.

In DeCaf we define blocks of code by placing commands between **<<** and **>>**. Later in the manual I will explain why did I decided to use **<<** and **>>**. I will also explain why do I even need to be able to create blocks of code.

This is the type of information that will allow you to give your first steps in the world of programming. After becoming a master in DeCaf you will be a few steps away from learning COFFEE or even C++. But, at least, the learning curve will be much smoother.



Chapter 1

What you need to know.

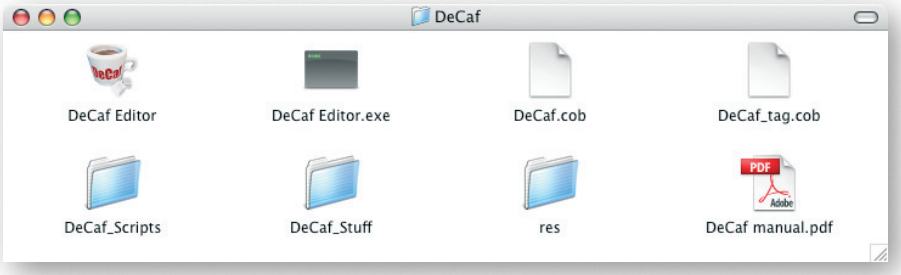
Chapter 1 - What you need to know.

Like all other plug-ins for Cinema 4D, DeCaf must be installed in the plugins folder that resides inside your Cinema 4D folder. When I say “installed” I mean copied because DeCaf, like most other Cinema 4D plug-ins, doesn’t really require any installation.

It is very important that you check the privileges of the DeCaf folder and all its content (folders and files). The user that is running Cinema4D — probably the administrator — should have read and write privileges to the DeCaf folder and all its content. This is because DeCaf is a very complex piece of software and reads and writes a lot to its folder.

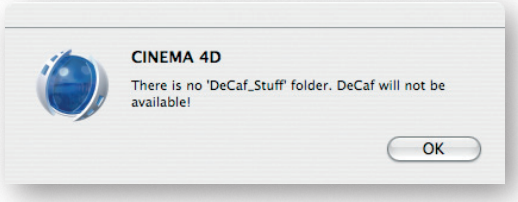
Also, the folder/file structure is very important and should not be changed in any manner. If you change the name or location of any folder or file inside the DeCaf folder, DeCaf will simply stop working. The only folder whose content you can mess with is the **DeCaf_Scripts** folder. Inside it reside all the scripts you create and you can manage them as you feel like it. You can delete them, duplicate them and even rename them. Just don't change or delete the extensions of the script files.

The usual structure of the DeCaf folder is the following:



The first time you run Cinema 4D after *installing* DeCaf, you will be asked to reveal where your Cinema 4D application is. This is just for DeCaf to know the name of your Cinema 4D application as it will need it when switching between Cinema 4D and the external DeCaf editor. If you don't change the location of your Cinema 4D application this will usually only be asked once.

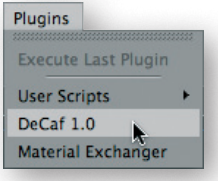
Each time you run Cinema 4D, DeCaf checks for its folder structure to make sure everything it needs is in place. If something doesn't pass the tests, you will get a warning and DeCaf will not be available. For example:



If everything checks out fine you will be asked to input your serial number. Don't worry as this will only happen the first time you run Cinema4D with DeCaf in the plug-ins folder. Just type the serial that you received when you purchased DeCaf and press the **OK** button.

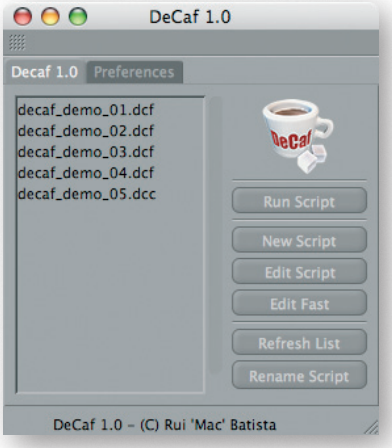
Chapter 1 - What you need to know.

It is **VERY IMPORTANT** that you click in OK and not just press Enter at the end of inputting the whole serial. DeCaf will only check the serial if you **PRESS THE 'OK' BUTTON!** The serial has five groups of five characters each, separated by hyphens. This is how it is provided to you but, when inputting it, you just have to type the characters. The cursor will automatically advance to the next field as soon as you type five characters and you should **NOT** type the hyphens.



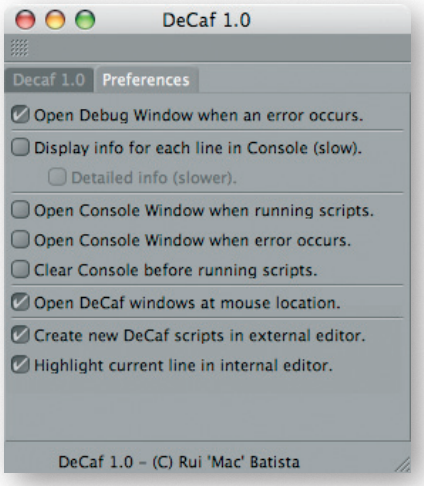
Assuming that everything is ok and that the serial is correct, once Cinema 4D finishes loading, you will get access to DeCaf from the plugins menu.

After choosing DeCaf from the plugins menu you will see its interface.



On the left side of the window there is a list of all available scripts. This list of scripts reflects the content of the **DeCaf_Scripts** folder, showing the regular DeCaf script files — with the extension **.dcf** — and the password protected DeCaf script files — with the extension **.dcc**.

On the right side, there are six buttons that perform operations on the script that is selected in the list. The top button, named **Run Script** will execute the script that is selected in the list. The button named **New Script** will create a new empty script. Depending on the preferences it will automatically open the external DeCaf editor or ask you for the name of the new script and open a new window for editing it. Pressing the **Edit Script** button will open the script selected in the list in the external DeCaf editor. Pressing the **Edit Fast** button will open the script selected in the list in a window,



without leaving Cinema 4D. Pressing the **Refresh List** button will reload all the DeCaf scripts that are inside the **DeCaf_Scripts** folder into the list, sorting then alphabetically. This is specially useful if you organized your DeCaf scripts manually. The last button is named **Rename Script**. As its name implies, you use it to rename a script — it has to be selected, of course — without leaving Cinema 4D.

You always have the choice to rename the scripts using the Mac Finder or Windows Explorer. Just don't forget that you may need to refresh the list — pressing the **Refresh List** button — if you edit something inside the **DeCaf_Scripts** folder while Cinema 4D is still opened.

Deleting and duplicating scripts always has to be done using the Mac Finder or Windows Explorer. Once again, if you do any of these operations while Cinema 4D is still opened you have to refresh the scripts list by pressing the **Refresh List** button.

As you may have noticed, there are two tabs in the DeCaf interface. The one described earlier is named **DeCaf 1.0** and it will be the one you will be using more often. The other one, named **Preferences** contains all the available options that allow you to customize DeCaf behavior.

In the following pages there is a detailed explanation about all the options available in the **Preferences** tab.

Open Debug Window when an error occurs. — **Default: ON**

When turned on, DeCaf will open the Debug Window whenever an error occurs. The Debug Window shows the line where the error occurs, the error message and a list of all variables an their content. It is specially useful when you are still learning DeCaf or when you are debugging your DeCaf code.

Display info for each line in Console (slow). — **Default: OFF**

When turned on, DeCaf will display the content of each line of code in the Console Window, as it is being executed. It states that this will slow down the code but, in fact, it is just a few milliseconds slower. Just turn this off if you want your code to run as fast as possible. This option is specially useful when you are still learning DeCaf or when you are debugging your DeCaf code.

Detailed info (slower). — **Default: OFF**

This option is available only when the previous option is turned on. When turned on, DeCaf will display additional information about the content of each line of code in the Console Window, as it is being executed. It states that this will slow down the code but, in fact, it is just a few milliseconds slower that the previous option. Just turn this off if you want your code to run as fast as possible. This option is specially useful when you are still learning DeCaf or when you are debugging your DeCaf code.

Open Console Window when running scripts. — Default:OFF

When turned on, DeCaf will automatically open the Console Window when you press the **Execute Script** button. This way you don't have to, manually, choose Console from the Window menu or press Shift+F10. It is specially important to keep the Console Window opened when you are starting to learn DeCaf or when you are debugging your code because very important information is displayed there.

Open Console Window when error occurs. — Default:OFF

When turned on, DeCaf will automatically open the Console Window whenever an error occurs. This is very useful because, sometimes, your code may not be doing what you want and you don't know what may be happening. If you have this option turned off and some important event occurs, DeCaf will open the Console and display a message asking you to turn on this option.

Clear Console before running scripts. — Default:OFF

When turned on, DeCaf will automatically clear the Console Window when you press the **Execute Script** button. This is very useful because, sometimes, the Console Window is full of "garbage" and you may want to have a clean display as you run each of your scripts.

Open DeCaf windows at mouse location. — Default:ON

When turned on, DeCaf will open its windows (editor and Debugger) at the mouse location. Otherwise, the windows will open at the top-left corner, always at the same location (128 pixels from the left of the screen and 128 pixels from the top of the screen).

Create new DeCaf scripts in external editor. **Default:ON**

When turned on, new DeCaf scripts will be created using the external editor (opening it if it is not already open). Otherwise, new scripts will be created in Cinema 4D, using the simpler editor. When you are still giving your first steps in DeCaf it is advisable to use the external editor. When you get more confident with DeCaf you can start using this internal, simpler editor.

Highlight current line in internal editor. **Default:ON**

When turned on, the line where the cursor currently is located, in the internal DeCaf editor is highlighted. It may be useful, specially for people with screens set to a very high resolution since the blinking cursor is not particularly visible.

If you ever decide to revert the preferences to their default values in one go, simply delete the **decafpref.txt** file that is inside the **DeCaf_Stuff** folder. DeCaf will create a new **decafpref.txt** file with the default values the next time you run Cinema 4D.

If you plan to use DeCaf very often you may want to dock the DeCaf window somewhere in your layout. Just drag the dotted square that is on the top-left corner of the DeCaf window (above the DeCaf tab) and dock the window wherever you prefer. You may even turn the docked DeCaf window into a tabbed window. This way you can associate it with other windows and save some space since DeCaf is not something that you need to have accessible all the time.

Chapter 2

Using the internal and external DeCaf Editor

Chapter 2 - Using the internal and external DeCaf Editor

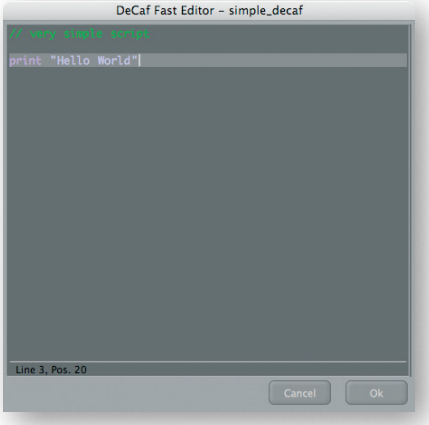
DeCaf comes with a special external editor. It is special because it is specifically tailored for DeCaf and it was made to teach DeCaf to anyone who uses it. You can always use it to create new DeCaf scripts or to edit already made scripts or you can decide to use the simpler internal editor which doesn't require you to leave Cinema 4D. Anyway, I advise you to use the external editor at least when you are still learning in how to use DeCaf. The external editor has lots of advantages:

Features	Internal	External
Syntax coloring	✗*	✔
Custom format display of code	✗	✔
Information about current script	✗	✔
Real-time information about code	✗	✔
Extensive information about DeCaf	✗	✔
Automatic insertion of labels or variables	✗	✔
Encryption of scripts	✗	✔
Find and replace	✗	✔
Verification of matching parenthesis and blocks	✗	✔
Fast edition followed by immediate execution	✔	✗
Highlighting of the current line	✔	✗

* Due to limitations of COFFEE (the language that was used to create DeCaf) the syntax coloring is not fully functional. Only keywords that are shared between COFFEE and DeCaf (for example, **print**, **if**, **end**) are colored. Comments are colored correctly, though. If I didn't included syntax coloring – even if it is not working correctly – it would not be possible to indent lines using the tab key. So, I believe it is a good compromise.

Besides, the external editor has a much nicer interface.

If you just want to quickly edit a script — and you already master the DeCaf set of commands — you can favour the internal editor. It is, without question, the fastest way to create/edit a script and test it out. Everything is done inside Cinema 4D, using a familiar interface. As soon as you end editing the script you press **OK** (no **Save** is required in the internal editor but you have to do it in the external editor) and you are ready to press the **Execute Script** button. This is how the internal editor looks like:



Not very pretty, I agree. But it does the job and does it fast. Its interface couldn't be much simpler. If you want to finish editing your script without saving changes simply press the **Cancel** button.

To accept the changes, press the **OK** button.

When the internal editor is evoked, it opens centered at the mouse location if the **Open DeCaf windows at mouse location** preference is turned on. If this preference is turned off the internal editor window always appears with its top left corner located 128 pixels away from the left margin of the screen and 128 pixels down from the top margin of the screen.

There is not much more to say about the internal editor since it is so simple. Besides, this chapter is dedicated to the external editor and there are lots of things to say about it.

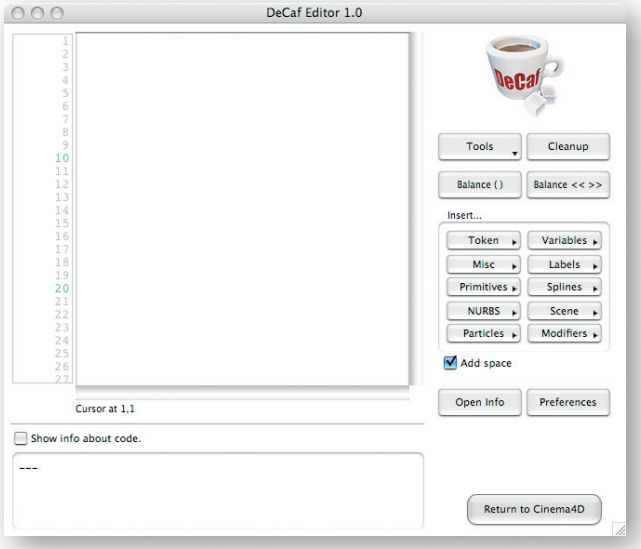
First, it is very important that the **DeCaf Editor** application is located at the correct location. It should be inside the DeCaf folder, floating around, NOT inside any other folder. If DeCaf doesn't find the **DeCaf Editor** application at its expected location, it will refuse to run.

There are two DeCaf Editor applications inside the DeCaf folder. One is for Macintosh computers and is named **DeCaf Editor.app** but the extension .app is set to be hidden. The other one is for Windows computers and is named **DeCaf Editor.exe**.

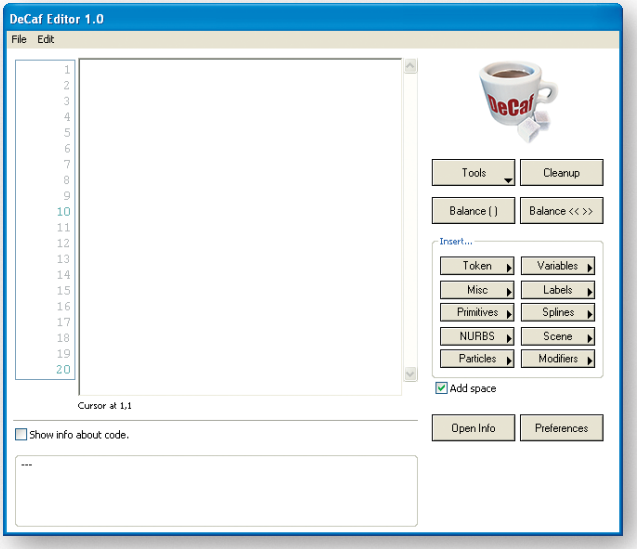
If you use a Macintosh computer there is no problem in keeping the **DeCaf Editor.exe** application inside the DeCaf folder. In the same way, if you use a Windows machine, there is no problem in keeping the **DeCaf Editor.app** application inside the DeCaf folder. But, if you want to delete the application that is not intended for your machine, there will be no problem also. DeCaf only searches for the **DeCaf Editor** application that is suited for the system where it is residing. The Macintosh version of the **DeCaf Editor** will appear as a folder named **DeCaf Editor.app** on Windows machines and the Windows version of the **DeCaf Editor** will appear on Macintosh computers with the icon that is set on that system to represent PC executable files.

You can always execute the **DeCaf Editor** application by double clicking it but it will mainly be evoked from inside Cinema 4D, when you press the **Edit Script** button.

When you open the **DeCaf Editor** application, this is what you see:

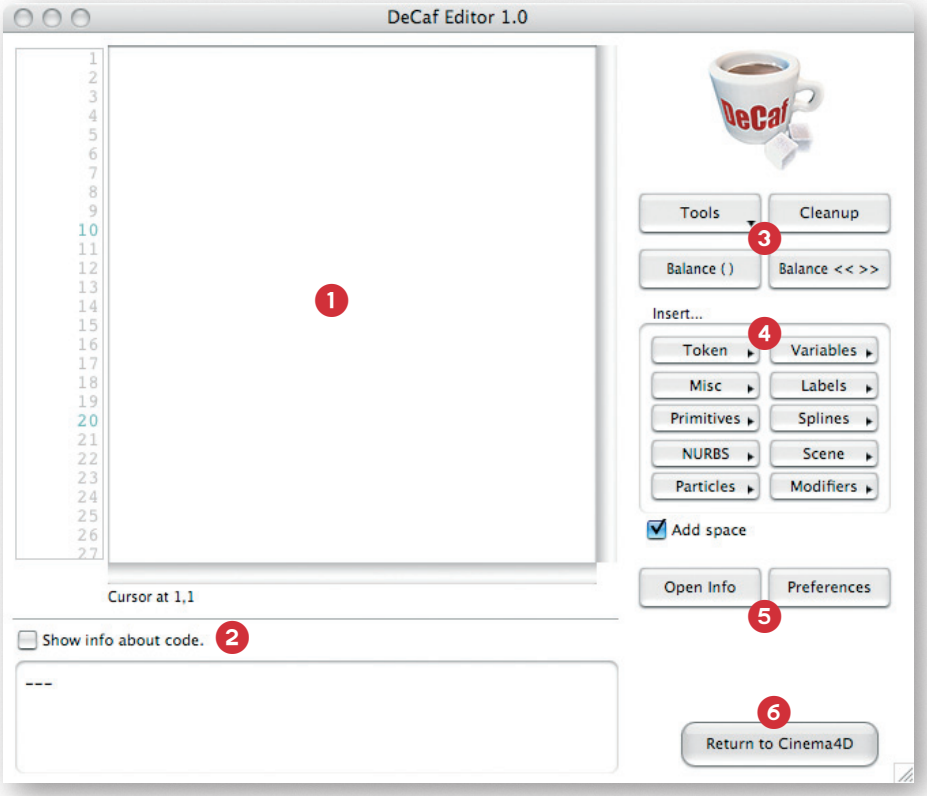


Macintosh version



Windows version

From now on I will always use screenshots from the Macintosh version because they look much nicer ☺



The main area ❶ is where you type your code. At the left of this area there is a list of line numbers. Their only purpose is to serve as a reference, specially when errors are found, as errors report to the line number where they occured.

Below the code input area there is an informative line displaying the location of the cursor. Once again, this is for your convenience only.

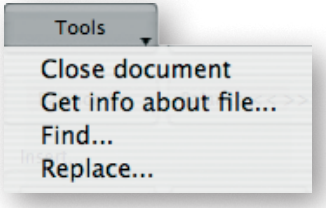
At the bottom left location of the DeCaf Editor window there is an area ❷ where additional information is displayed about the code that you type.

If the **Show info about code** option is turned on, as you place your cursor over any De-

Caf token in the main input area, detailed information about that token is displayed in that area at the bottom left. Tokens are commands, functions or reserved words that are displayed in the color that is set for **Code Keywords** in the **DeCaf Editor** preferences[1] (see the DeCaf Editor preferences on page ???).

The information that is showed is not as detailed as the one you can get from the **Info** window that you get when you press the **Open Info** button but it is usually very useful when you are start learning DeCaf.

At the area labeled with ❸ you have four buttons with useful functions. The first one, named **Tools** is, in fact, a menu. When you press it, you get a list of options:

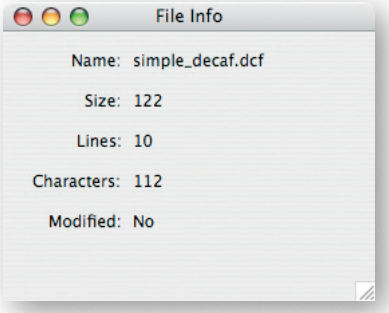


The first one – **Close document** – will close the current document and, if changes were made since the last save or if it is a new document, it will ask you if you want to save the current state of the document.

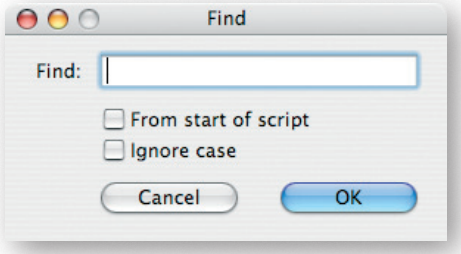
If the DeCaf Editor was evoked from the **Edit Script** button to edit a script all changes must be saved before returning to Cinema4D. Unlike the internal editor, the DeCaf Editor does not save changes automatically when scripts are closed.

1 You must have the **Color code** option turned on for displaying the tokens in a different color.

Selecting the second option – **Get info about file**... – will open a window with information about the current state of the file:



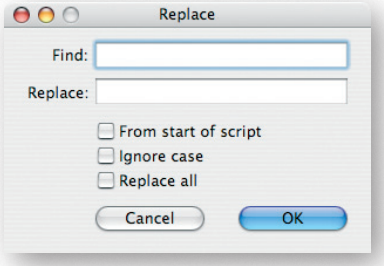
The third item – **Find**... – will open a window with options to perform a search in your script:



If you keep the **From start of script** option turned off, the search will be performed from the location of the cursor. If the same option is turned on, the search will start from the beginning of the script.

If the **Ignore case** option is turned off, only a complete match of the exact same word will obtain results. This means that if you search for the word **sphere**, the words **Sphere** or **SPHERE** will not result in a match. If the option is turned on, **sphere**, **Sphere**, **SPHERE** or even **SpHeRe** are all treated as the same word.

The fourth item – **Replace**... – will open a window with options to perform a search and replace operation in your script:



If you keep the **From start of script** option turned off, the search will be performed from the location of the cursor. If the same option is turned on, the search will start from the beginning of the script.

If the **Ignore case** option is turned off, only a complete match of the exact same word will obtain results. This means that if you search for the word **sphere**, the words **Sphere** or **SPHERE** will not result in a match. If the option is turned on, **sphere**, **Sphere**, **SPHERE** or even **SpHeRe** are all treated as the same word.

If the **Replace all** option is turned off, the operation will stop after each successful replacement and wait for you to decide what to do: if you want to go on searching and replacing (press the **OK** button) or if you want to give up (press the **Cancel** button).

If the **Replace all** option is turned on, after pressing the **OK** button, ALL occurrences that match the searched expression will be replaced by the replacement expression. Of course, choosing Undo will revert the script to its previous state before all the replacements.

After each substitution you have the option of choosing **Undo** from the **Edit** menu (or press ⌘-Z or Ctrl-Z) to revert to the expression that was previously there.

When you press the **Cleanup** button (still in the area labeled ❸) the **DeCaf Editor** attempts to clean up the line where the cursor resides. Cleaning up means that any extra spaces are deleted. Spaces that are inside quotation marks are kept, of course.

Still in the area labeled ❸, there is a button named **Balance ( )**. Its purpose is to check for matching parenthesis in a arithmetic expression since all opening parenthesis must have a correspondent closing parenthesis. If there aren't as many opening parenthesis as there are closing parenthesis in a numeric expression, an error will occur. So, to make sure your numeric expression is correctly balanced place your cursor somewhere in the middle of the expression. Then press the **Balance ( )** button. **DeCaf Editor** will search to the left and right of the location of the cursor and highlight the whole extension of characters between correspondent opening and closing parenthesis.

Since an expression can have many opening and closing parenthesis he best way to explain all this is through some examples.

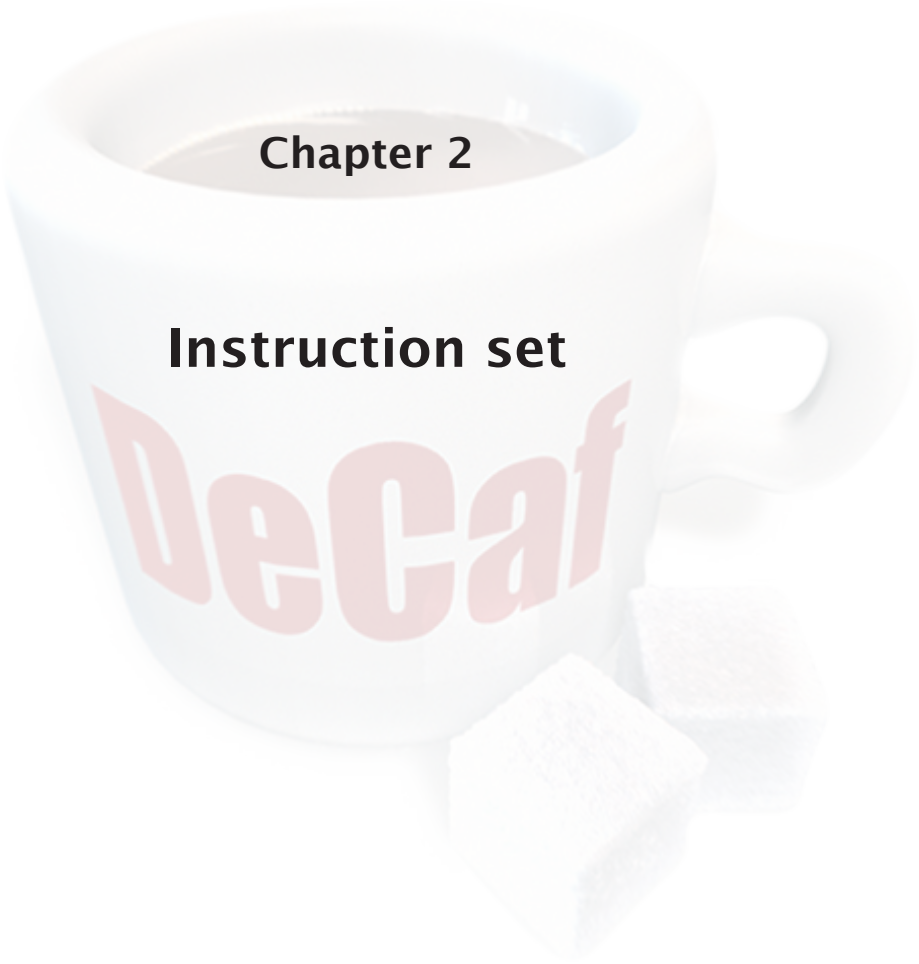
Consider the following expression:

5*(sine(x-3.1415)/(y+z)+2)

Let us see some possible situations:

Placing the cursor here...	... and pressing **Balance ( )** will highlight...
5*(sine(x-3.1415)/(y+z)+2) ↑	5*(sine(**x-3.1415**)/(y+z)+2)
5*(sine(x-3.1415)/(y+z)+2) ↑	5*(**sine(x-3.1415)/(y+z)+2**)
5*(sine(x-3.1415)/(y+z)+2) ↑	5*(sine(x-3.1415)/(**y+z**)+2)

If there are no parentheses to match or if there is a mismatch of opening and closing parentheses, a beep will sound.



Chapter 2

Instruction set

DeCaf – Introduction

wildcard

wildcards

examples: **wildcards** on
wildcard off

Parameters

required	*on/off*

Wildcard naming can be turned on or off to perform searches in your document (see page §§§).

When wildcard naming is turned off, only strict name matching could be performed. This means that a search for "cube" will only find a match when an object named "cube" with all lowercase letters is found. So, objects named "CUBE" or "Cube", for example, will not be considered.

Turning wildcards on will allow for more flexible searches. So, if someone searches for the name "cube", objects whose name is "CUBE" or "Cube" will also be considered as valid matches.

When wildcards are turned on, the metacharacters * and **?** can also be used.

The character * stands for any sequence of characters. A search for "C*" will return as valid matches all names starting with a letter "C". A search for "*01" will return as valid matches all names ending with "01". A search for "Ca*01" will return as valid matches all names starting with the letters "Ca" and ending with "01".

The character ? stands for any single character. A search for "P?ll" will return as valid matches names like "Pill", "poll", "pull", etc. A search for "Cube??" will return as valid matches all names starting with "Cube" and that include two extra characters at the end, no matter what they are.

A search for "?????" will return as valid matches all names with five letters in their names.

The characters * and **?** can be combined in a single search like, for example "C??e*". This search string could consider as valid matches, names like "Cube", Cube01", "Cube_child", "Cone", "Cone-001", "Cake", "Codename", etc.

A search string can contain as many **?** characters as needed but only a single * character.

Wildcards are turned on by default each time you execute a script.

create

examples: **create** cube
create cube **named** "Block"
create cube **named** "Block" **as child**
create cube **named** "Block" **below**

Parameters

required	*primitive_type*
optional	**named** *"name"*
optional	**[as] child**
optional	**below**

Creating primitives in DeCaf is very easy. You just have to type **create** *primitive_type*. This command will create a default primitive of the specified type. The default primitive will have the default name, unless an optional parameter (**named**) is provided, followed by the desired name, between quotes. For example:

```
create sphere
```

will create a sphere primitive named "Sphere" (without the quotes).

```
create sphere named "Globe"
```

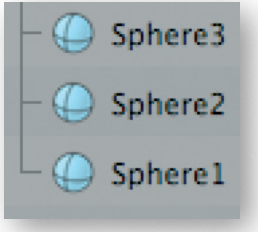
will create a sphere primitive named "Globe" (without the quotes).

As soon as you create a primitive, the **#current** variable will point to it. If you need a pointer to the newly created object, assign the value of the **#current** variable to another variable as soon as you create a new primitive.

All new primitives are created at the top of the hierarchy, by default. This means that, if you run the following DeCaf script:

```
create sphere named "Sphere1"
create sphere named "Sphere2"
create sphere named "Sphere3"
```

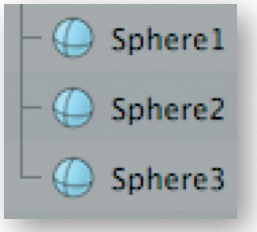
You will get the following hierarchy in your document:



If you want to create new objects below the newly created one, add the parameter below to the create command, like this:

```
create sphere named "Sphere1"
create sphere named "Sphere2" below
create sphere named "Sphere3" below
```

This will create the following hierarchy:



search

examples: **search** "cube01"
search VARNAME
search "cube01" from #current
search "cube01" from VARNAME backwards

Parameters

required	*name*
optional	**from** *object* **[backwards]**

The **search** instruction will perform a search from the beginning of your document for an object with a specific name.

The name can be provided as a literal expression or as a variable containing a name.

If an object matching the name is found, the **#current** variable will point to it. If no object is found, the **#current** variable will contain a null value.

The optional parameter **from** should be followed by a variable pointing to an object. When this parameter is provided, the search will be performed starting from the object pointed by the variable instead of from the beginning of the document. The **from** parameter can also include the optional sub-parameter **backwards**. When this sub-parameter is provided the search is performed in the direction of the top of the document instead of to the end.

The search names can be provided as wildcards, if that option is turned on. Please refer to the **wildcard** instruction, at page §§§, for more about wildcard naming.