```
.-----===---------------------------------===-----------===------.
|    /__/|         ___        ___         /  /\        /  /\     |
|   |  |:|        /__/\      /  /\       /  /::\      /  /:/     |
|   |  |:|        \  \:\    /  /:/      /  /:/\:\    /  /:/      |
| __|  |:|         \  \:\  /__/::\     /  /:/~/:/    /  /:/  ___  |
|/__/\_|:|____ ___  \__\:\ \__\/\:\__ /__/:/ /:/___ /__/:/  /  /\ |
|\  \:\/:::::/ /__/\ |  |:|    \  \:\/\ \  \:\/:::::/ \  \:\ /  /:/ |
| \  \::/~~~~  \  \:\|  |:|     \__\::/  \  \::/~~~~   \  \:\  /:/  |
|  \  \:\       \  \:\__|:|     /__/:/    \  \:\        \  \:\/:/   |
|   \  \:\       \__\::::/      \__\/      \  \:\        \  \::/    |
|    \__\/          ~~~~                    \__\/         \__\/     |
|                                                                  |
`------------------------------------------------------------------'
.----------------,--------------------------------.------------------.
|                (     The Art of Scripting  Vol.2 )      by Grifisx |
`----------------`--------------------------------'------------------'
Version: etherea-0.1a/20062201
```

Objects, Classes, Array and Dictionaries

Start:

In this tutorial we'll have an eye on the main concept of *objects*, *classes*, *arrays*, *dictionaries* and *files writing and reading*; with much code and small scripts, as that's the best way to learn coding.
First, let's introduce the concept of object, a very important lesson on how to write more complex code.
What is an object? Have a look at the following code:

```
%myobject = $new(object,0,objectname);
if(%myobject)echo "Object created!";
else echo "Object creation failed!";
```

Put it in the “Script Tester” (yes, the one with the black bomb icon) and run it to see the result.

Here's a deep look of what we have yet wrote:
%myobject = $new(object,0,object_name);

To create an object you need to use the *$new()* function that requires three parameters:
- an object class (there's a forthcoming issue on Classes) ;
- father object ID (can also be “0” for toplevel objects -ie no father ones- but we'll have a look later);
- the name of the onject (can be empty).

Objects can be treated as pseudo-structures (like C structures); don't be afraid not to be familiar with a structure, code will clarify everything. Want to create an user object containing many fields such as Nick, Username and so on?

```
# User object creation
%User = $new(object,0,user_description)
# Let's set some more parameters
%Utente->%nickname = Grifisx
%Utente->%username = mDm-Team
%Utente->%hostname = do.not-disturb.net
%Utente->%info = Grifisx took this example from KVIrc handbook
```

```
%Utente->%info << Tnx to Pragma
```

Put this code in the Script Tester and run it, then in the input line of any window type:
/echo %User->%nickname

As you can see it returned nickname field echo, as for every other single field requested; while doing an object echo we have:
/echo %User

it returns in output the object ID (in this case 1710.1128236637).
As you can see it is very, very snuggy to use objects, especially if there's the need to create very complex scripts.

An eye on classes now.
A class, as stated in the official handbook, is a collection of methods that defines any object properties:

```
class (calc,object)
{
      constructor()
      {
            echo Usage\:
            echo To compute the sum of two numbers:
            echo \/\%Calc\-\>\$somma2\(\x\,\y\)
            echo To compute the quotient of two numbers:
            echo \/\%Calc\-\>\$div\(\x\,\y\)
      }
      somma2()
      {
# $0 and $1 are the values to assign the function by recall it
# for example "\%Calc->$somma2(8,6)" $0=8 e %1=6
            echo $($0+$1)
      }
      div()
      {
            echo $($0/$1)
      }

}
%Calc=$new(calc)
```

The execution of this script will give you the chanche either to sum or to divide two numbers using those teo functions you have created: *$somma2()* and *$div()*.
From the input line try this:
/%Calc->$div(10,5)

As it shows, there has been the creation of a *calc* class, that is *object* type (also it could be a *widget* type for the graphical objects, but will see this a little more later) and on its inside there are three newly created functions (or methods if you prefer): *$constructor()*, *$somma2()* and *$div();* later create the object with the code *"%Calc=$new(calc)"* and all the functions created inside the class are ready to be used, recalling them by the symbol "->".

Arithmetic function *$(<arithmetic_expression>)* does execute the compute of the expression within the brackets.

Furthermore notice the code inside *$constructor()* is added upon the creation of the class itself automatically and that's the right place to initialize variables.

For example:

```
class(addressbook,object)
{
      addName()
      {
            $$->%name=$0-
      }
      addSurname()
      {
            $$->%surname=$0-
      }
      addTel()
      {
            $$->%tel=$0-
      }

}
%AddressBook=$new(addressbook)
```

What are those *$$->* and *$0-*?
$0-* is the parameter the function receives (remembering aliases? It works out the same way: when we are going to recall the function *$addName() we'll need to give it the name to be added).

$$* means "Variable belonging to this class", ie if you need to create a class variable (showed only inside the class itself) you have to create it with *$$->* or *$this->; same way if you want to recall a function of a given class inside another function: *$$->$function().*

Array and D*ictionary* are really important concepts, because those allow the creation of a collection of values, strings or objects, mean data.

An array is a collection of variables data indexed by number; the first index of the array is "0" while the last index is equal to the lenght of the array itself minus one (as it starts from zero).
To obtain the number of the elements contained in an array the right expression to use is %*ArrayExample[]#.*
It is not necessary to declare the lenght of the array as in many other programming languages, going on adding a number the lenght of the array will vary by itself and if the first assigned element will have an index greater than "0" all the other positions will be left empty.

For example:

```
%Array[0]=Grifisx
%Array[1]=Noldor
%Array[2]=Pragma
#Print the whole content of the array
echo %Array[]
#Print the lenght of the array
echo %Array[]#
#Print only the first element
```

```
echo %Array[0]
```

Or try this one:

```
%Array[0]=Grifisx
%Array[1]=Do not show this
%Array[2]=Noldor
%Array[5]=Secret shhhh..
%Array[8]=Pragma
for(%i=0;%i < %Array[]#;%i+=2)echo Entry %i: \"%Array[%i]\";
```

As you can see is almost easy to create collections indexed by numbers, as it is very simple to move inside them; here is a *for* cycle but there was the possibility to use a *foreach(%item,%Array[])echo %item* one or a *while*.

It is possible to initialize an array this way:
%Array[]=$array(Grifisx,Noldor,Pragma,Madero);
namely using the *$array(<el1>,<el2>,<el3>,<el4>,..)* function.

And now Dictionary takes its turn.
Dictionaries are associative arrays of strings; let's have a look with an example taken from the official handbook:

```
%Songs{Jimi Hendrix} = Voodo child
%Songs{Shawn Lane} = Gray piano's flying
%Songs{Imogen Heap} = Hide and Seek
%Songs{Greg Howe} = "Full Throttle"
# Show everything in a string
echo %Songs{}
# Show every element of the dictionary
foreach(%var,%Songs{})echo %var
```

As in the array *%Songs{}#* will return the number of the elements of the dictionary; while *%Songs{}@* whill return a list divided by commas.
Dictionaries and arrays can be used together, to have a dictionary of arrays.
We can manage those collections in the way we like best. Maybe they may be seem such difficult concepts but will end it "easy" to use.

Now it is useful to know how to save and read from a text file, as we have some collections and we want to store them, first of all it's necessary to put them into files.

$file.read("file_to_be_read")
file.write("file_to_be_written")

Thee first function allows to read from a text file while the second one (it is a command, not a function as it don't starts with a *$*) allows to write in.
If we have a text file (labeled "*usersdatabase*") containing a list of names separated by commas (like *"Grifisx,Noldor,Pragma,Madero"*) and if we want to put all those nicknames in an array we need to do:

%users[] = $split(",",$file.read($file.localdir(usersdatabase)))

The *$split* function will be changed into $str.split() -so pay

attention is it gives out an error- and does the split of a string followiing the separator definition given (in this case is comma “,”); while the function *$file.localdir()* returns the local configuration of KVIrc (quick try *“/echo $file.localdir()”*).
In this way we have created the array *%users[]*, containing on its inside the nicknames read by the text file.

While to write on a file we have:
file.write $file.localdir(usersdatabase) %users[]

Now in the Script Tester:

```
%Users[]=$array(Grifisx,Noldor,Pragma,Madero);
file.write $file.localdir(usersdatabase) %Users[];
%UsersNew[] = $split(",",$file.read($file.localdir(usersdatabase)));
%idx=0;
while(%idx!=%UsersNew[]#)
{
      echo User: %UsersNew[%idx];
      %idx++;
}
file.remove $file.localdir(usersdatabase)
```

By running the script the file has been created, read and then removed with the command *file.remove* (to see other available commands check the KVIrc manual in *Commands* -> letter *f*).

We do know enough to create something a little more complex: a tiny clipboard manager.

```
class(notes,object)
{
      constructor()
      {
# In the constructor all instructions are on display, remembering
# the constructor is the first thing to be execute as the class is
# created
            echo $k(5,8) \-\-\-Notes\-\-\-\
            echo $k(5,8) Manual comomands:
            echo $k(5,8) \/\%Notes\-\>\$viewApp      $b  Show
available notes
            echo $k(5,8) \/\%Notes\-\>\$addApp\(note\)      $b  Adds a
new note
            echo $k(5,8) \/\%Notes\-\>\$delApp\(note\)      $b  Erases
a note
      }
# Function to add a note
      addApp()
      {
# Creation of an array filled with file content
            $$->%appuntos[] =
$split(",",$file.read($file.localdir(kvircAppuntidb)))
            $$->%ap = 0
# Now the creation os a simple dictionary, as it is more easy to work
# with strings and are easily characterizing, in the same time
# there's a practical application of the dictionaries uses; then it
# will be filled with the array content and that will result in:
# %Array{hello}=hello , so that by the time to erase it I won't be
# worry about its numeric index as my index is equal to the element
# and I'll be able to catch in every moment.
            while($$->%ap < $$->%appuntos[]#)
            {
                  $$->%ListNotes{$$->%appuntos[$$->%ap]} = $$-
>%appuntos[$$->%ap]
                  $$->%ap++;
            }
            $$->%ProvaAP =  $0-
            $$->%ListNotes{$$->%ProvaAP} = $$->%ProvaAP
# All notes on display
            foreach($$->%var,$$->%ListNotes{}) echo $k(5,8)$$->%var
# File storing
            file.write $file.localdir(kvircAppuntidb) $$-
>%ListNotes{}
            echo $k(5,8) End of List : $$->%ListNotes{}# \) Notes
stored
      }

# Function to erase a note
      dellApp()
      {
# As the previous function
            $$->%appuntos[] =
$split(",",$file.read($file.localdir(kvircAppuntidb)))
            $$->%ap = 0
            while($$->%ap < $$->%appuntos[]#)
            {
                  $$->%ListNotes{$$->%appuntos[$$->%ap]} = $$-
>%appuntos[$$->%ap]
                  $$->%ap++;
            }
            $$->%ProvaAP =  $0
# Easy to delete without worrying about the index as I do use the
# same name
            $$->%ListNotes{$$->%ProvaAP} = ""
# Empty
            foreach($$->%var,$$->%ListNotes{}) echo $k(5,8)$$->%var
            file.write $file.localdir(kvircAppuntidb) $$-
>%ListNotes{}
            echo $k(5,8) End of list: $$->%ListNotes{}# \) Notes
stored
      }
# Function to see the notes
      viewApp()
      {
# Same as above
            $$->%appuntos[] =
$split(",",$file.read($file.localdir(kvircAppuntidb)))
            $$->%ap = 0
            while($$->%ap < $$->%appuntos[]#)
            {
                  $$->%ListNotes{$$->%appuntos[$$->%ap]} = $$-
>%appuntos[$$->%ap]
                  $$->%ap++;
            }
            foreach($$->%var,$$->%ListNotes{})echo $k(5,8) $$->%var
            echo $k(5,8) End of List : $$->%ListNotes{}# \) Notes
stored
```

```
        }
}
# Object %Notes created
%Notes=$new(notes)
```

There is nohing more to explain as if you did understood the concept of *array*, *dictionary* and file *read* and *write* (and *classes*), the script will explain by itself.
You can write a better script surely (or use more functional constructs) but we used only the most important ones to make a great one script.
It will be surely better when you'll have understand all the other concepts KVIrc hides =)
/ECHO STOP.

- - - - - - -- - - - - - - -- - - - - - - - - - -- - - - - - -- - - - -
"You see things; and you say `*Why?*' But I dream things that never were; and I say `*Why not?*"
(George Bernad Shaw)
- - - - - - - - - -- - - - - - - - -- -- - - - - - - - - - - - - - - - - -
Grifisx