Creating a Programming Language – by i542, modified by Secure_ICT

If you have ever created a program, you have probably used about 50% of the available function abilities because it is too hard to make room for all functions *and* its parameters.

A small programming language like this is a great addition to your script. Beginners will use a GUI, and advanced users will use your "command line" to use all abilities of your program.

In addition, you can create a whole new "command line" program which will use your system.

So, let's get started.

First, create a script in ScITe or you desired text editor. The first thing you want to do is, add a GUI which will allow the users to see what is happening in the program:

```
#include <GUIConstants.au3>

$font = "courier new"

$gui = GUICreate("My Command Line", 338, 192, 192, 125)
$txt = GUICtrlCreateEdit("", 0, 0, 481, 217, $ES_READONLY+ $WS_VSCROLL)
GUICtrlSetFont(-1, 9, 400, 4, $font)
GUICtrlSetBkColor(-1, 0x000000)
GUICtrlSetColor(-1, 0xffffff)
GUICtrlSetData($txt, "> Welcome to my command line!" &@CRLF)
$cLine = GUICtrlCreateInput("clear", 0, 168, 257, 21)
$Execute = GUICtrlCreateButton("Execute", 264, 168, 74, 22,
$BS_DEFPUSHBUTTON)
GUISetState(@SW_SHOW)
```

When you have added or created a GUI, write the commands on a piece of paper or in Notepad that you want, your programming language to do when it is typed into your Command Line.

The commands that we are going to use are: "echo", "clear", "popup", "exit" and "help".

Now, decide which character you want to be a separator. Depending on your choice, commands will look like this:

1. Echo, Hello World

2. Echo; Hello World

3. Echo. Hello World

I recommend the first variant.

Now find the ASCII code for the separator in the AutoIt Help File Appendix or look here:

For	Use
; (Semicolon)	59
, (Comma)	44
. (Period)	46

Now add this to the bottom of your script:

```
While 1
  $msg = GuiGetMsg()
  Select
  Case $msg = $GUI_EVENT_CLOSE
     ExitLoop
  Case $msg = $Execute
     $string = StringSplit(GuiCtrlRead($cLine), Chr(44))
     Switch $string[1]
     EndSwitch
  EndSelect

Wend

Func _ConsoleWrite($text)
  GUICtrlSetData($txt, $text & @crlf, GUICtrlRead($txt))
EndFunc

Func _TrimSpaces(ByRef $parameter)
  Local $string
  $string = StringSplit($parameter, "")
  If $string[1] = Chr(32) then
     $string = StringtrimLeft($parameter, 1)
     Return $string
  Else
     Return $parameter
  EndIf

EndFunc
```

This piece of code assumes you are using a comma (,) for the separator and that you have not changed anything above.

Now, between Switch and EndSwitch add your first command, "echo".

Echo will add text to your "Return Space".

So, add a "case 'echo'" command between Switch and EndSwitch.

Then trim spaces using _TrimSpaces function. This will look like:

$string[2] = _TrimSpaces($string[2])

([2] is the first parameter – the one after the first comma. For the second parameter use $string[3] and so on.)

This function needs only one parameter, so we need only one _TrimSpaces function.

Then, add the text to "return space". This is done using _ConsoleWrite function. So the function should look like this:

```
Case "echo"
        $string[2] = _TrimSpaces($string[2])

        _ConsoleWrite($string[2])
```

Now run your script.

Then at the "Command line" enter:

echo, Hello World

If you have done all of the steps correctly a, "Hello World" message will appear in the "return space"!

Yay! You have just created your first function in your programming language!

Close the GUI.

Now add a "clear" command to your script. This is done using GuiCtrlSetData() function. Set the **$txt** to blank. The code will look like:

```
Case "clear"
        GUICtrlSetData($txt,"","")
```

Now run the script again and test the following functions:

echo, blah blah blah

echo, blah blah blah

clear

echo, Cleared – right?

Good job. Again this function does not need parameters.

You've probably noticed two things: text stays in the command line and when you write the wrong function nothing happens. This will soon be changed – stay with us!

On the end of every function add a "clear command line" function. This is done by using GuiCtrlSetData(). Now your functions should look similar to:

```
Case "clear"
        GuiCtrlSetData($txt,"","")
        GuiCtrlSetData($cLine,"","")
Case "echo"
        $string[2] = _TrimSpaces($string[2])
        _ConsoleWrite($string[2])
        GuiCtrlSetData($cLine,"","")
```

You can now test the script again. Enter any command. The command line is now cleaned!

What about bad functions, you maybe asking? The only thing you have to do is add a **Case Else** function to the bottom of your Cases. Now you are on your own – I will just tell you what you have to do.

1. Add a Case Else line before EndSwitch Line
2. Enter an error message you want using _ConsoleWrite function. Use $string[1] for the function name.
3. Run & Test

Seems you are good at creating programming languages. Let's go to functions with more than one parameter. We are going to do a simple Message Box, the function will be called "popup".

When you execute any dialog boxes it is good idea to have a confirmation in "return space" that I has succeeded, it could look like this:

"<Popup Executed>".

Before you get started, I will give you several tips:

$string[1] stands for the FUNCTION typed. $string[2] stands for first parameter after the comma etc.

It is a good idea not to use flags with the popup. We will later learn about numbers in functions.

Don't forgot to trim spaces

So, the function syntax will be:

popup, title, text

Now, without cheating, try to create that function in 1 minute.

Ready – steady – GO!

(1 minute…)

Ok, dude, time is now up!

Now let's see your results. Run the script and enter:

popup, Boo!, i542 is the greatest!

Worked? Yay! Crashed? EEK! You haven't passed the challenge.

Now let's see the results. If your code is just like it (you can change the variable's name!) then you have won:

```
MsgBox(0, $string[2], $string[3])
GUICtrlSetData($txt,"<Popup Executed>" & @CRLF, GuiCtrlRead($txt))
GuiCtrlSetData($cLine,"","")

; Displays a small dialog box
```

Now that you have learnt all of the functions I can teach you only one more thing.

Never, ever release a command line if it has less than 10 commands!

It's time for you to add the user's escape function to your program – exit. It is that simple, so you won't need more than 15 (ok, 30) seconds to create it. Remember this function has no parameters.

Now: All you have to do is enter Case "exit" and a simple word in a new line – exit

Good. Now let's learn our last command today – help.

The help syntax is going to be:

help

help command, <command name>

help about

As you can see, the "help" command will contain 3 functions. So start the Help System and add the first one.

When user enters help, the text must appear, which will teach the user how to use other commands. This is done with _ConsoleWrite. You don't need to clean up the Return Space so user must NOT write help again. Although it is a good idea to...

Help command requires "switch-in-switch". This is very simple. Here is the help system for the functions "echo" and "popup":

Case "help /popup"

_ConsoleWrite("> Syntax: popup, <title>, <text>")

_ConsoleWrite("> Shows a little popup window")

_ConsoleWrite("> Remarks: none")

Case "help /echo"

_ConsoleWrite("> Syntax: echo, <text>")

_ConsoleWrite("> Places text in this control")

_ConsoleWrite("> Remarks: none")

EndSwitch

And "help about" is so simple I shouldn't explain it, but just in case:

Case "about"

_ConsoleWrite("> My Programming Language")

_ConsoleWrite("> Created with i542's Programming Language System")

_ConsoleWrite("> This language is licensed to: You")

That's all for today. Hope 6 pages of text are enough for you! Enjoy creating your own programming language. Stay tuned – this tutorial will continue soon!

i542

Thank you to Secure_ICT for modifying this file and adding an example.

P.S. Don't forget to visit http://exod-soft.sittingonair.com.

P.P.S. Don't forgot to visit http://freeforall.ipbfree.com.