25th of August 2007

Python
for
mIRC

Contents

1. Introduction

This mIRC DLL initializes a Python interpreter, which can be used to execute Python code. It also contains a Python class, providing functionality to communicate with mIRC (via SendMessage). It supports virtually every Python module you can think of, including threads. If that's not enough to knock you off your chair, you'll have a second chance right now: you can embed Python code into your MSL files.

I'll use standard mIRC notation for parameters: <param> stands for a required parameter "param" while [param] is used for optional parameters.

2. Requirements

- Python 2.5 properly installed. (Look for a "python25.dll" in your %PATH% environment variable)
- PyWin32 for Python 2.5 (http://sourceforge.net/project/showfiles.php?group_id=78018)
- A clue. This DLL is intended to be compatible with competence. Failure to adhere to this requirement may lead to unforeseen consequences which include but are not limited to exploding devices (e.g. CPUs - or your brain), traffic accidents, tornados, heart attacks and diarrhoea. I (the author) can not be made responsible for any kind of damage or benefit gained from downloading and using this DLL.

3. Functions

All function names are **case-sensitive**!

The Python interpreter is automatically initialized once you call any of the following functions.

- "**Python**"

  This function executes Python code in the existing context

  1) Parameters
     1. <string> = Python code to be executed
  2) Returns 1 or 0, indicating success or failure.
  3) Examples

```
1. echo -s $dll(Python4mIRC.dll, Python, x = 1)
2. echo -s $dll(Python4mIRC.dll, Python, ordstr = lambda
   string: ' '.join([ord(char) for char in string]))
```

- I suggest that you use an alias to make this a little shorter. I'll stick to this one:

```
alias py {
  var %dll $mircdirPython4mIRC.dll
  if $isid { return $dll(%dll, Python, $1-) }
  else { /dll %dll Python  $1- }
}
```

4. Embed Python into MSL

Use the parse Python function that is automatically loaded during initialization of the interpreter.

IMPORTANT NOTE: Python4mIRC.dll does not support .ini script files. Save your scripts as plain text!

1) Parameters
    1. <filename>, a string containing the Python-ready script file name.
    2. <lineno>, the number of the line containing the parse function call (parse automatically adds 1 to the line number).
    3. [blockname], used for tracebacks.
2) Returns nothing. However, you'll be calling this with the "**Python**" DLL function and thus get 1 or 0. If used in combination with goto, use the jump points :1 and :0 for error handling / cleanup.

- Insert Python code in the following line(s). Use ctrl+i (tab character) for indentation.
- End the Python code block with a jump point. If you use goto, you should insert :1 and :0 here.
- This alias will help (a lot!). The third parameter is optional just as it is in the parse function.

```
alias startpy {
 var %s = $+(parse,$chr(40),", $replace($1,\,\\) , ", $&
 $chr(44),$2,$iif($3, $chr(44) $3),$chr(41))
 return $py(%s)
}
```

- All of this sounds stupidly complicated and complex. Here's an example:

```
alias testpython {
  echo -s Testing Python!
  goto $startpy($script, $scriptline)
  print 'Testing Python #2'
  for x in reversed(xrange(3)):
       print x
  def testfunc():
       print 'Testing Python #3'
  testfunc()
  :0
  echo -s An exception occurred somewhere!!
  return
  :1
  echo -s Test successful!
}
```

This should echo 7 lines ("Testing Python!", "Testing Python #2", "2", "1", "0", "Testing Python #3" and "Test successful").

5. mIRC Python class

- I never intended to fully document this class, so it contains a lot of dirty hacks and workarounds. I, for one, do not care and neither should you. I will provide you with the necessary information to make full use of this class.
- The mIRCObject class is an interface to a running mIRC instance, which is identified by its main window handle. It manages communication between Python and mIRC via SendMessage and mapped memory files. Each instance of this class must have a unique "map number" ("mapno" parameter of __init__; defaults to 69) to ensure functionality of communication.
- A default instance of this class is created during initialization of the interpreter ("mIRC"), which is linked to the mIRC instance loading the DLL.

- It is possible to create objects pointing to other running instances of mIRC by specifying another main window handle (Python4mIRC.dll does not require any scripts or the like).
- Functions of mIRCObject are (only relevant ones listed)
  1) __init__(handle, mapno=69)
     Creates an instance of the class (YA RLY) and initializes namespaces for commands, identifiers and variables. It also redirects stdout and stderr output to the mIRC status window.
  2) newthread(mapno = self.mapno+1)
     Creates and returns a new instance intended to be used in threads. Provided for convenience.
- Attributes:
  1) stderr
     This object manages incoming errors and, by default, redirects them to the mIRC status window. This behaviour can be changed by setting the "func" attribute of stderr to a function object taking exactly one parameter. This function will then be called whenever there is a new line of an error message waiting.
     Example:
     ```
     mIRC.stderr(lambda msg: mIRC.cmd.echo("@python",
     msg))
     ```
     This redirects any output to the window named @python. See below for explanation of the mysterious mIRC.cmd.echo part!
  2) stdout (same as above.. This is for standard output)
  3) cmd
     This is the namespace to execute mIRC commands. Use cmd.<alias name> to receive a function which can then be used to execute the given alias. Parameters given will be appended to the call (including space delimiters). All parameters are str()'d. Its return value is None.
     Example:
     ```
     mIRC.cmd.echo("-s test")
     mIRC.cmd.echo("-s", "test")
     ```
     These statements are equivalent.
     To call aliases whose names contain special characters and other stuff Python doesn't really like, use mIRC.cmd("alias_name") instead of cmd.aliasname.
  4) id
     This is the namespace to evaluate identifiers. For instructions see cmd. id automatically inserts comma delimiters between parameters. Unlike cmd, functions returned by id have a return value containing the result of the identifier call.
     Example:
     ```
     nick = mIRC.id.me()
     chans = mIRC.id.chan('0')
     someone = mIRC.id.nick('#somechannel, 1')
     thesameguy = mIRC.id.nick('#somechannel', 1)
     ```
     The last two statements are equivalent. Parameters for the identifier can be specified as one, continuing, comma-delimited string, as several arguments, which will then be merged to one comma-delimited string or

any combination of these two methods. Use mIRC.id("identifier_name") for pythonophobic identifier names.

5) var

   Namespace for (global) variables. (I'm really getting tired of typing this shit!)

   No functions here, just straight forward attributes:

```
somevar = mIRC.var.somevar
somevar = mIRC.var.get('somevar')
mIRC.var.somevar = 'this will be in %somevar'
mIRC.var.set('somevar', 'and this one, too')
```

   First and last two statements are equivalent. Workaround functions obviously included in these examples.

6) handle and mapno (both integers)

   Self-explanatory.

6. Credits

I'd like to thank the following persons/groups/terroristic organizations:
starGaming, for being patient enough to explain whatever I needed to know.
#ich-sucke (Quakenet) for listening (or not listening) to my rants.
Everyone who contributed (input, criticism, proof reading and everything else)**.**