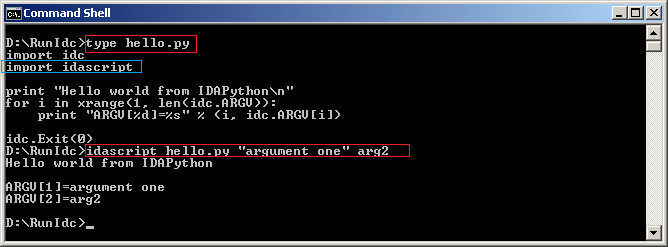
**Running scripts from the command line with idascript**

Posted on [July 8, 2010](http://www.hexblog.com/?p=128) by [Elias Bachaalany](http://www.hexblog.com/?author=3)

In this blog post we are going to demonstrate how the ‘-S’ and ‘-t’ switches (that were introduced in [IDA Pro 5.7](http://www.hex-rays.com/idapro/57/index.htm)) can be used to run IDC, Python or other supported scripts from the command line as if they were standlone scripts and how to use the **idascript** utility



**Background**

In order to run a script from the command line, IDA Pro needs to know which script to launch. We can specify the script and its argument via the “-S” switch:

idag **-Sfirst.idc** mydatabase.idb

Or:

idag **-S"first.idc *arg1* *arg2*"** mydatabase.idb

In case the script does not require a database (for example, it works with the debugger and attaches to existing processes), then IDA Pro will be satisfied with the "-t" (create a temporary database) switch:

idag -S"first.idc *arg1* *arg2*" **-t**

Where **first.idc**:

#include <idc.idc>

static main()

{

Message("Hello world from IDC!\n");

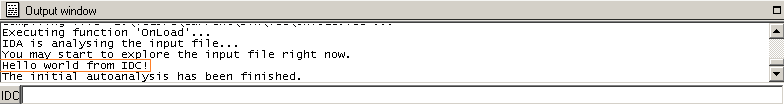
return 0;

}

If we run IDA Pro with the following command:

idag -Sfirst.idc -t

We notice two things:

1. Nothing is printed in the console window: This is because the message will show in the output window instead:  
     
     
     
     
   (It is possible to save all the text in the output window by using the **IDALOG** environment variable.)
2. IDA Pro remains open and does not close: To exit IDA Pro when the script finishes, use Exit() IDC function.

In the following section, we will address those two problems with *idascript.idc* and *idascript.py* helper scripts and the *idascript* utility.

**Running scripts from the command line**

In order to print to the console window, we will not use IDC / Message() instead we will write to a file and when IDA Pro exits we will display the contents of that file.

Our second attempt with **second.idc**:

extern g\_idcutil\_logfile;

static LogInit()

{

g\_idcutil\_logfile = fopen("**idaout.txt**", "w");

if (g\_idcutil\_logfile == 0)

return 0;

return 1;

}

static LogWrite(str)

{

if (g\_idcutil\_logfile != 0)

return fprintf(g\_idcutil\_logfile, "%s", str);

return -1;

}

static LogTerm()

{

if (g\_idcutil\_logfile == 0)

return;

fclose(g\_idcutil\_logfile);

g\_idcutil\_logfile = 0;

}

static main()

{

LogInit(); // Open log file

LogWrite("Hello world from IDC!\n"); // Write to log file

LogTerm(); // Close log file

Exit(0); // Exit IDA Pro

}

Now let us run IDA Pro:

idag -Ssecond.idc -t

and type afterwards:

type idaout.txt

to get the following output:

Hello world from IDC!

To simplify this whole process, we wrote a small win32 command line utility called **idascript:**

IDAScript 1.0 (c) Hex-Rays - A tool to run IDA Pro scripts from the command line

It can be used in two modes:

a) With a database:

idascript database.idb script.(idc|py|...) [arg1 [arg2 [arg3 [...]]]]

b) With a temporary database:

idascript script.(idc|py|...) [arg1 [arg2 [arg3 [...]]]]

Since we will be using LogInit(), LogTerm(), LogWrite() and other helper functions over and over, we moved those common functions to **idascript.idc**.

The script **first.idc** can now be rewritten like this:

#include <idc.idc>

#include "idascript.idc"

static main()

{

InitUtils(); // calls LogInit()

Print(("Hello world from IDC!\n")); // Macro that calls LogWrite()

Quit(0); // calls LogTerm() following by Exit()

}

As for IDAPython, we wrote a small class to redirect all output to **idaout.txt**:

import sys

class ToFileStdOut(object):

def \_\_init\_\_(self):

self.outfile = open("**idaout.txt**", "w")

def write(self, text):

self.outfile.write(text)

def flush(self):

self.outfile.flush()

def isatty(self):

return False

def \_\_del\_\_(self):

self.outfile.close()

sys.stdout = sys.stderr = ToFileStdOut()

Thus, **hello.py** can be written like this:

import idc

**import idascript**

print "Hello world from IDAPython\n"

for i in xrange(1, len(**idc.ARGV**)):

print "ARGV[%d]=%s" % (i, idc.ARGV[i])

idc.Exit(0)

**Sample scripts**

**Process list**

The sample script **listprocs.idc** will enumerate all processes and display their ID and name:

#include <idc.idc>

#include <idascript.idc>

static main()

{

InitUtils();

**LoadDebugger**("win32", 0);

auto q = **GetProcessQty**(), i;

for (i=0;i<q;i++)

Print(("[%08X] %s\n", **GetProcessPid**(i), **GetProcessName**(i)));

Quit(0);

}

**Kill process**

The **killproc.idc** script illustrates how to find processes by name and terminate them one by one:

#include <idc.idc>

#include "idascript.idc"

#include "procutil.idc"

static main()

{

InitUtils();

// Load the debugger

LoadDebugger("win32", 0);

// Get parameters

if (**ARGV**.count < 1)

QuitMsg(0, "Usage: killproc.idc ProcessName\n");

auto procs = **FindProcessByName**(ARGV[1]), i;

if (procs.count == 0)

QuitMsg(-1, "No process(es) with name " + ARGV[1]);

for (i=procs.count-1;i>=0;i--)

{

auto pid = procs[i];

Print(("killing pid: %X\n", pid));

**KillProcess**(pid);

}

Quit(0);

}

To test the script, let us suppose we have a few instances of notepad.exe we want to kill:

D:\idascript>idascript killproc.idc notepad.exe

killing pid: 878

killing pid: 14C8

D:\idascript>

We used here the “ARGV” variable that contains all the parameters passed to IDA Pro via the -S switch, FindProcessByName() utility function and KillProcess() (check procutil.idc)

The trick behind terminating a process is to attach and call StopDebugger(). The following is an excerpt from **procutil.idc** utility script:

static KillProcess(pid)

{

if (!AttachToProcess(pid))

return 0;

StopDebugger(); // Terminate the current process

// Normally, we should get a PROCESS\_EXIT event

GetDebuggerEvent(WFNE\_SUSP, -1);

}

**Process information**

The **procinfo.idc** script will display thread count, register information and the command line arguments of the process in question:

#include "idascript.idc"

#include "procutil.idc"

static **DumpProcessInfo**()

{

// Retrieve command line via Appcall

Print(("Command line: %s\n", **GetProcessCommandLine**()));

// Enum modules

Print(("Module list:\n------------\n"));

auto x;

for (x = **GetFirstModule**();x!=BADADDR;x=**GetNextModule**(x))

Print(("Module [%08X] [%08X] %s\n", x, **GetModuleSize**(x), **GetModuleName**(x)));

Print(("\nThread list:\n------------\n"));

for (x=GetThreadQty()-1;x>=0;x--)

{

auto tid = GetThreadId(x);

Print(("Thread [%x]\n", tid));

SelectThread(tid);

Print((" EIP=%08X ESP=%08X EBP=%08X\n", **Eip**, **Esp**, **Ebp**));

}

}

static main()

{

InitUtils();

// Load the debugger

LoadDebugger("win32", 0);

// Get parameters

if (ARGV.count < 2)

QuitMsg(0, "Usage: killproc.idc ProcessName\n");

auto procs = FindProcessByName(ARGV[1]), i;

for (i=procs.count-1;i>=0;i--)

{

auto pid = procs[i];

if (!AttachToProcess(pid))

{

Print(("Could not attach to pid=%x\n", pid));

continue;

}

DumpProcessInfo();

DetachFromProcess();

}

Quit(0);

}

The function **GetProcessCommandLine** is implemented (using [Appcall](http://hexblog.com/2010/01/practical_appcall_examples_1.html)) like this:

static GetProcessCommandLine()

{

// Get address of the GetCommandLine API

auto e, **GetCmdLn** = LocByName("kernel32\_GetCommandLineA");

if (GetCmdLn == BADADDR)

return 0;

// Set its prototype for Appcall

SetType(GetCmdLn, "char \* \_\_stdcall x();");

try

{

// Retrieve the command line using Appcall

return **GetCmdLn()**;

}

catch (e)

{

return 0;

}

}

**Extracting function body**

So far we did not really need a specific database to work with. In the following example (**funcextract.idc**) we will demonstrate how to extract the body of a function from a given database:

#include <idc.idc>

#include "idascript.idc"

static main()

{

InitUtils();

if (**ARGV.count** < 2)

QuitMsg(0, "Usage: funcextract.idc FuncName OutFile");

// Resolve name

auto ea = **LocByName**(ARGV[1]);

if (ea == BADADDR)

QuitMsg(0, sprintf("Function '%s' not found!", ARGV[1]));

// Get function start

ea = **GetFunctionAttr**(ea, **FUNCATTR\_START**);

if (ea == BADADDR)

QuitMsg(0, "Could not determine function start!\n");

// size = end - start

auto sz = **GetFunctionAttr**(ea, **FUNCATTR\_END**) - ea;

auto fp = fopen(ARGV[2], "wb");

if (fp == 0)

QuitMsg(-1, "Failed to create output file\n");

**savefile**(fp, 0, ea, sz);

fclose(fp);

Print(("Successfully extracted %d byte(s) from '%s'", sz, ARGV[1]));

Quit(0);

}

To test the script, we use idascript utility and pass a database name:

D:\idascript>**idascript** *ar.idb* funcextract.idc start start.bin

Successfully extracted 89 byte(s) from 'start'

D:\idascript>

**Other ideas**

There are other ideas that can be implemented to create useful command line tools:

* Process memory read/write: Check the **rwproc.idc** script that allows you to read from the process memory to a file or the other way round.
* Associate .IDC with idascript.exe: This allows you to double-click on IDC scripts to run them from the Windows Explorer
* Scriptable debugger: Write scripts to debug a certain process and extract needed information
* …

**Installing the idascript utility**

Please download idascript and the needed scripts from [here](http://hexblog.com/ida_pro/files/idascript_files.zip) and follow these steps:

1. Copy idascript.exe to the installation directory of IDA Pro (say %IDA%)
2. Add IDA Pro directory to the PATH environment variable
3. Copy idascript.idc and procutil.idc to %IDA%\idc
4. Copy idascript.py to %IDA%\python
5. Optional: Associate \*.idc files with idascript.exe

Comments and suggestions are welcome!