Scripting SOS.NET

from https://bitbucket.org/grozeille/sosnet/overview

installed using the script: **Installer_SosNet.cs**

note: see references and links at the end

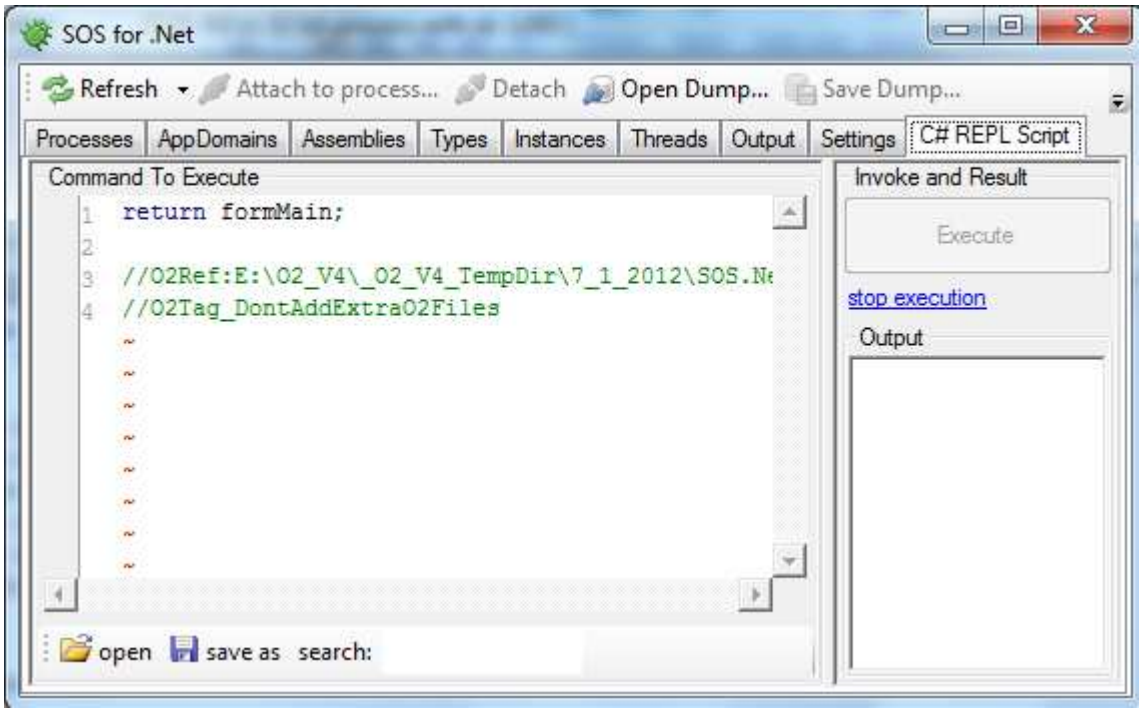
Start SosNet Gui and store its main Form object in an o2Cache object

```
//return O2.Kernel.O2LiveObjects.LiveObjects;


//var topPanel = O2Gui.open<Panel>("{name}",700,400);
//var sos= new SosController();
//sos.LoadSettings();
//
//
if ("sos".o2Cache().isNull())
{
	O2Thread.staThread(
		()=>{
				"Opening form".info();
				var mainForm= new FormMain();
				"sos".o2Cache(mainForm);
				mainForm.ShowDialog();
				"sos".o2Cache(null);
				"cleaned cache".info();
			});
	this.sleep(1000);
 }
else
	"Sos From still there".info();

var sos = (FormMain)"sos".o2Cache();
return sos;

//using SOS.Net
//using SOS.Net.Core.Cdb

//O2File:Script_ExtensionMethods.cs
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
```

Set an internal SOS property

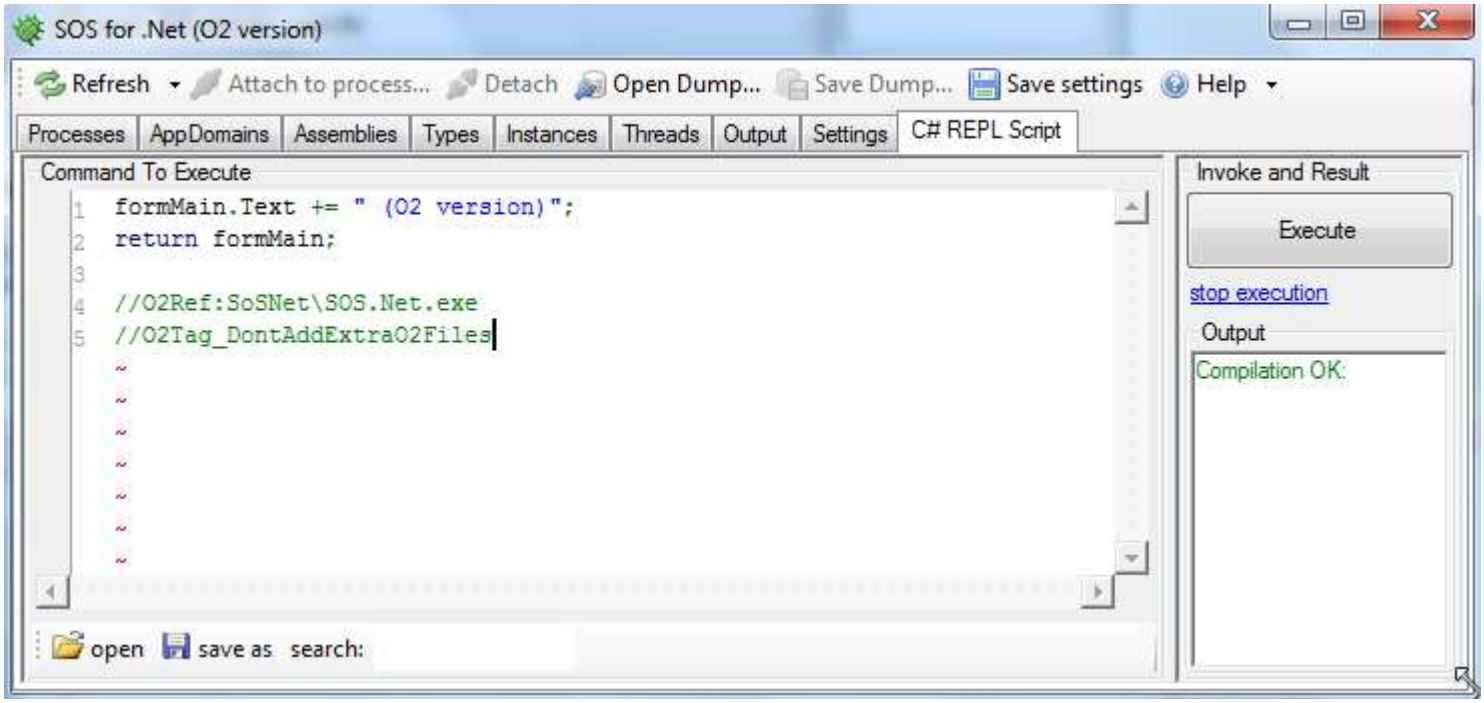
```
var sos = (FormMain)"sos".o2Cache();

//set the value of CdbPath
var controller = (SosController)sos.field("controller");
var settings = (CdbSettings)controller.field("settings");
settings.CdbPath = @"C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86";
```

Add a C# REPL environment to it:



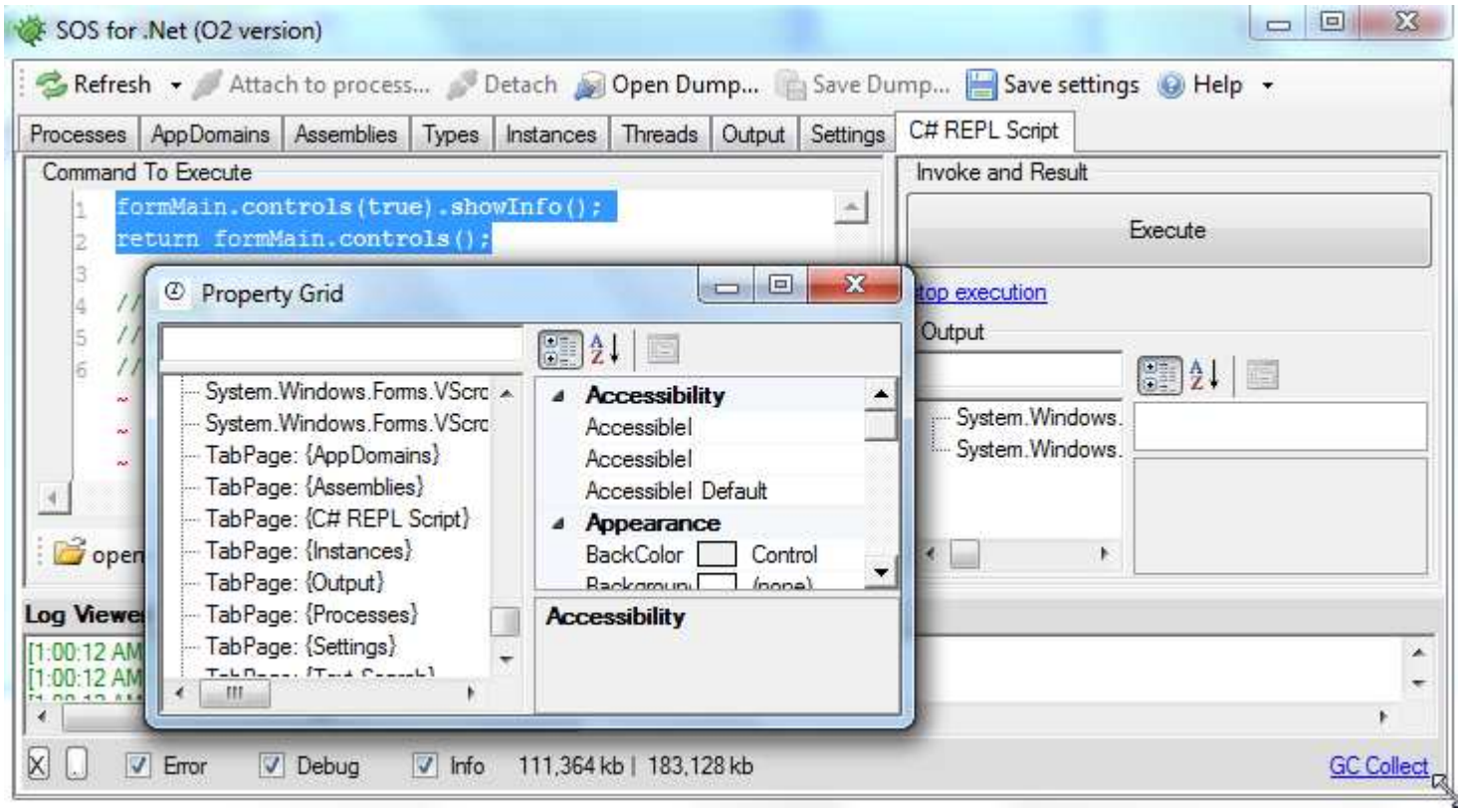
Which can now be used to control the SosNet gui

```
formMain.Text += " (O2 version)";
return formMain;

//O2Ref:SoSNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```



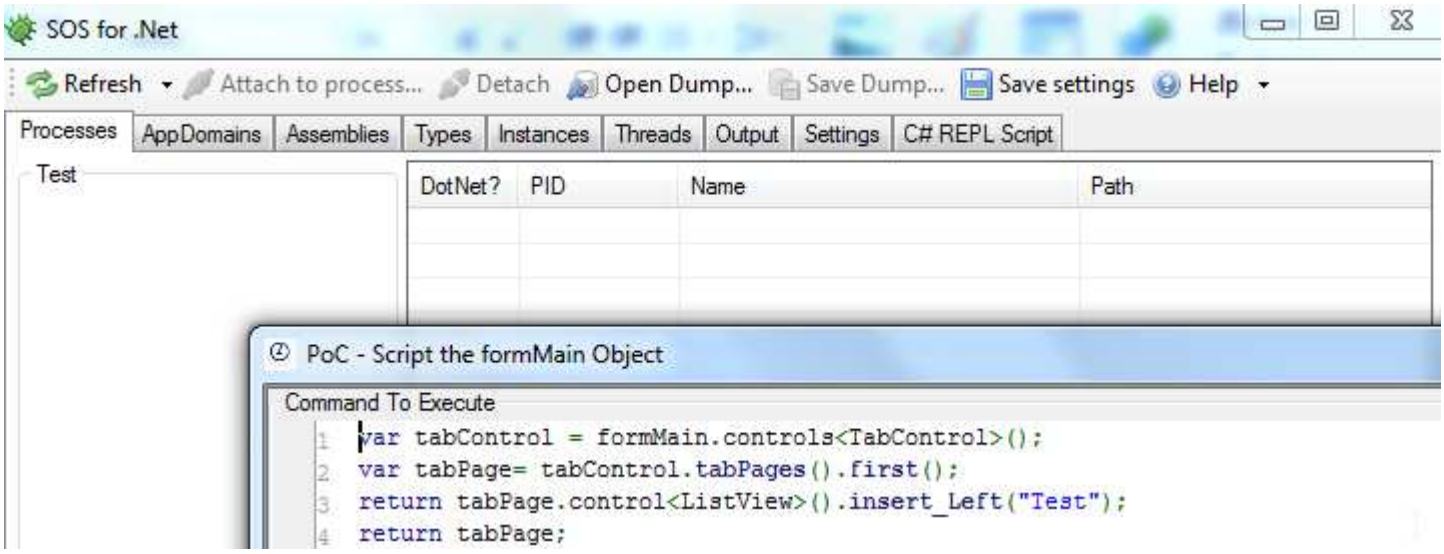
View all controls in this current form:

```
formMain.controls(true).showInfo();
return formMain.controls();
```



We can add a new panel to one of the Tabs:

```
var tabControl = formMain.controls<TabControl>();
var tabPage= tabControl.tabPages().first();
return tabPage.control<ListView>().insert_Left("Test");
```



Removing and adding the main TabControl (just in case also storring it in a cache variable)**:**

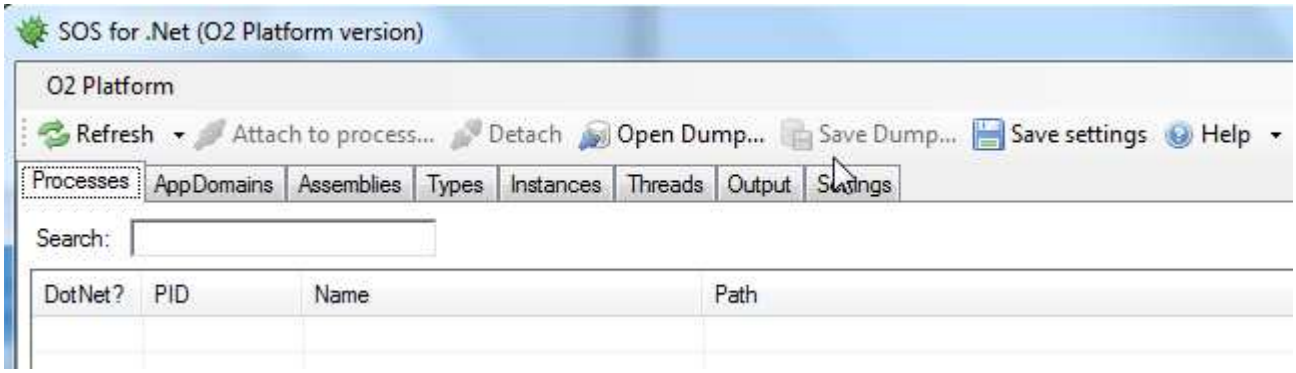
```
var tabControl = "tabControl".o2Cache(()=> formMain.controls<TabControl>());

formMain.Controls.Remove(tabControl);

this.sleep(1000);

formMain.Controls.Add(tabControl);

return tabControl;
```

Adding a new Menu with a couple useful O2 scripts:

```
sos.add_Menu().add_MenuItem("O2 Platform")
    .add_MenuItem("O2 C# REPL Script Editor", ()=> sos.script_Me())
    .add_MenuItem("O2 Log Viewer", ()=> "Util - LogViewer.h2".local().executeH2Script())
    .add_MenuItem("Open Find Scripts GUI", ()=> "Util - O2 Available scripts.h2".local().executeH2Script
())
    .add_MenuItem("Open Main O2 GUI", ()=> "Simple O2 Gui.h2".local().executeH2Script());
```

The best place to create this menu is on the main Gui creator code:

```
if ("mainForm".o2Cache().isNull())
{
	O2Thread.staThread(
		()=>{
				"Opening form".info();
				var mainForm= new FormMain();
				mainForm.Text += " (O2 Platform version)";
				mainForm.add_Menu().add_MenuItem("O2 Platform")
					.add_MenuItem("O2 C# REPL Script Editor", ()=> mainForm.script_Me
())
					.add_MenuItem("O2 Log Viewer", ()=> "Util - LogViewer.h2".local
().executeH2Script())
					.add_MenuItem("Open Find Scripts GUI", ()=> "Util - O2 Available
scripts.h2".local().executeH2Script())
					.add_MenuItem("Open Main O2 GUI", ()=> "Simple O2 Gui.h2".local
().executeH2Script());

				"mainForm".o2Cache(mainForm);
				mainForm.ShowDialog();
				"mainForm".o2Cache(null);
				"cleaned cache".info();
			});
	this.sleep(1000);
 }
```



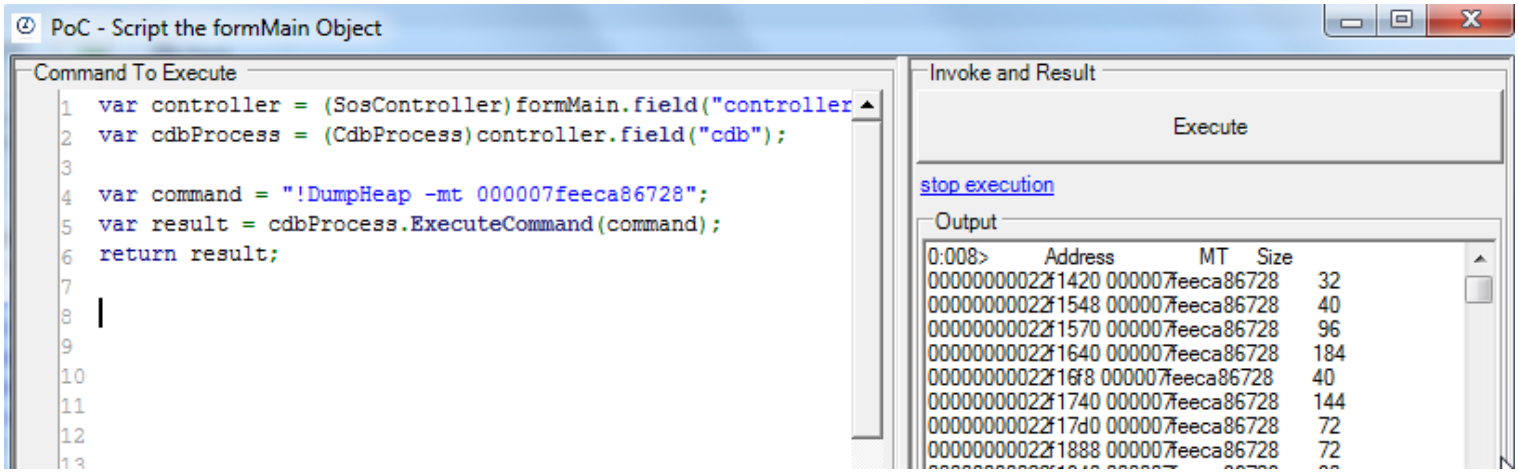
Start a process and attach to it

```
var exe = @"E:\O2_V4\_O2_V4_TempDir\6_30_2012\Util - LogViewer [08145]\Util - LogViewer.exe";
var process = exe.startProcess();

var controller = (SosController)formMain.field("controller");
controller.AttachToProcess(process.Id.str());
formMain.field("toolStripButtonAttach").prop("Enabled",false);
formMain.field("toolStripButtonDetach").prop("Enabled",true);
formMain.field("appDomainsToolStripMenuItem").prop("Enabled",true);
formMain.field("assembliesToolStripMenuItem").prop("Enabled",true);
formMain.field("typesToolStripMenuItem").prop("Enabled",true);
formMain.field("toolStripButtonDump").prop("Enabled",true);
controller.ExecuteCommand(@".load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\sos.dll");
controller.ExecuteCommand(@".load E:\O2_V4\_O2_V4_TempDir\_ToolsOrAPIs\SosNet\x64\sosex.dll");

return controller.ExecuteCommand("!help");


return controller;
return formMain;
```

```
//using SOS.Net
//using SOS.Net.Core.Cdb
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

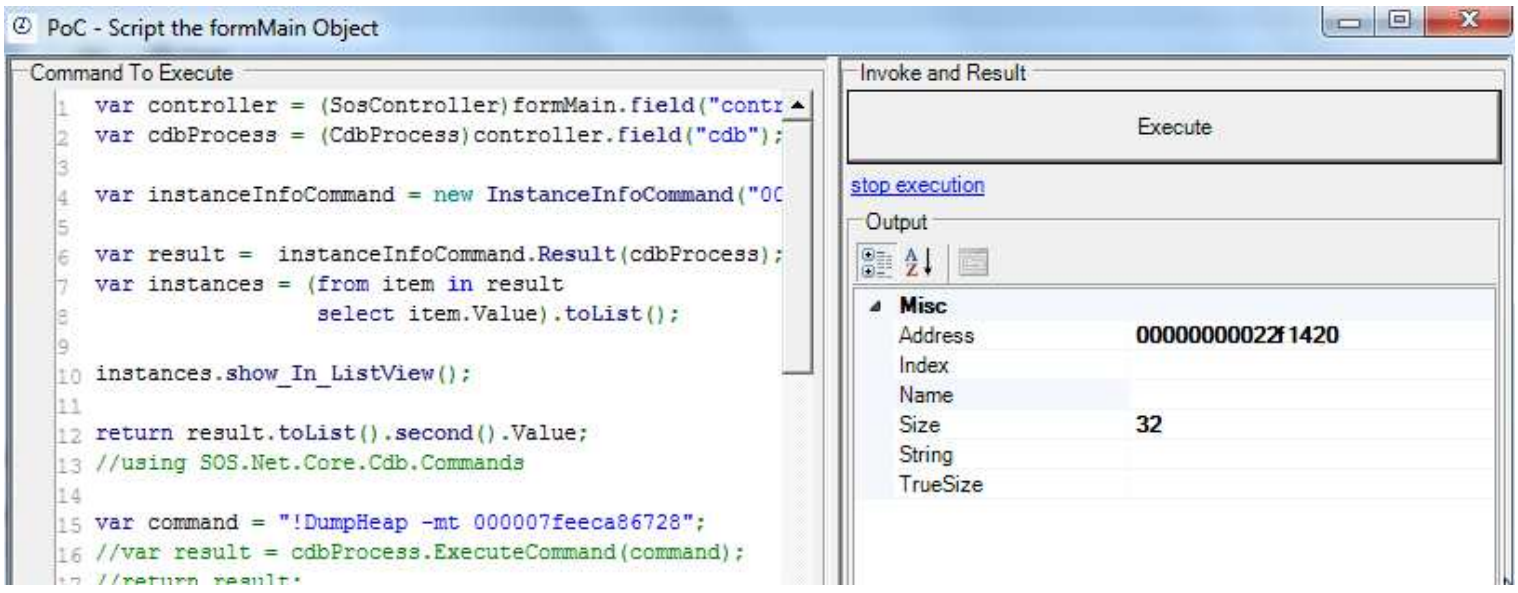
Gettting all instances of a String (directly not using SosNet objects)

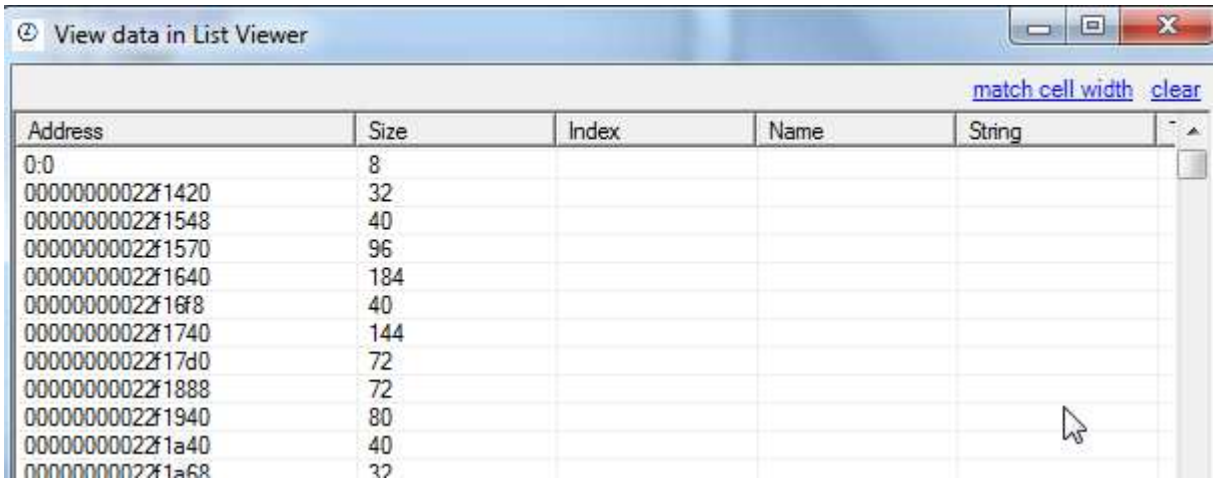
```
var controller = (SosController)formMain.field("controller");
var cdbProcess = (CdbProcess)controller.field("cdb");

var command = "!DumpHeap -mt 000007feeca86728";
var result = cdbProcess.ExecuteCommand(command);
return result;
```



Getting all instances of string using SosNet object

```
var controller = (SosController)formMain.field("controller");
var cdbProcess = (CdbProcess)controller.field("cdb");

var instanceInfoCommand = new InstanceInfoCommand("000007feeca86728");

var result =  instanceInfoCommand.Result(cdbProcess);
var instances = (from item in result
                 select item.Value).toList();

instances.show_In_ListView();

return result.toList().second().Value;
```





Get 100 first variables (making the gui output textarea empty and not visible during data fetch)

```
var textBox = (TextBox)formMain.field("textBoxCdbOuput");
textBox.set_Text("");
textBox.visible(false);

var controller = (SosController)formMain.field("controller");
var cdbProcess = (CdbProcess)controller.field("cdb");

var instanceInfoCommand = new InstanceInfoCommand("000007feeca86728");
var o2Timer = new O2Timer(".").start();

var result =  instanceInfoCommand.Result(cdbProcess);
var instances = (from item in result
                        select item.Value).Take(200).toList();
instances.remove(0)
             .remove(0);
var strings = new List<string>();

var count = 0;
foreach(var instance in instances)
{
/*      var  output = cdbProcess.ExecuteCommand("!DumpObj {0}".format(instance.Address));
       var value = output.split_onLines()[5];
       strings.add(value);*/

       var instanceInfoDetailsCommand = new InstanceInfoDetailsCommand(instance.Address);
       var value = instanceInfoDetailsCommand.Result(cdbProcess)
                                                                  .first()
                                                                  .Value;

       strings.add(value.String);
       if (count++ % 25 ==0)
              "[{0} / {1}]".info(count, instances.size());
}

o2Timer.stop();
textBox.set_Text("");
textBox.visible(true);
return strings.Distinct().toList();
```

Extract all strings in .NET process (in this case 4493)

```
var instanceInfoCommand = new InstanceInfoCommand("000007feeca86728");
var o2Timer = new O2Timer(".").start();

var result =  instanceInfoCommand.Result(cdbProcess);
var instances = (from item in result
                        select item.Value).Take(10000).toList();
instances.remove(0)
             .remove(0);
var strings = new List<string>();

var count = 0;
foreach(var instance in instances)
{
       textBox.set_Text("");
```

```
        var  output = cdbProcess.ExecuteCommand("!DumpObj {0}".format(instance.Address));
        var value = "*" + output.split_onLines()[5].subString(13);
        strings.add(value);

        if (count++ % 25 ==0)
                "[{0} / {1}]".info(count, instances.size());
}

o2Timer.stop();
textBox.set_Text("");
textBox.visible(true);
return strings.Distinct().toList();
```

Very unsafe way to replace strings values in memory:

```
if (value == "Util - LogViewer")
        {
                matches++;
                cdbProcess.ExecuteCommand("ezu {0}+a \"{1}\"".format(instance.Address, ("edit " +
matches).str()));
        }
```

You can load the sosex.dll extension like this:

```
var controller = (SosController)formMain.field("controller");
var cdbProcess = (CdbProcess)controller.field("cdb");

controller.ExecuteCommand(@".load E:\O2_V4\_O2_V4_TempDir\_ToolsOrAPIs\SosNet\x64\sosex.dll");
```

```
.load E:\O2_V4\_O2_V4_TempDir\_ToolsOrAPIs\SosNet\x64\sosex.dll
0:011> 0:011> 590a4fa1-9979-4f84-8bee-b57426bcce23

!help
0:011> SOSEX - Copyright 2007-2012 by Steve Johnson - http://www.stevestechspot.com/
To report bugs or offer feedback about SOSEX, please email sjjohnson@pobox.com
Quick Ref:
-------------------------------------------------
bhi       [filename]                                    BuildHeapIndex - Builds an index file for heap objects.
bpsc      (Deprecated.  Use !mbp instead)
chi                                                     ClearHeapIndex - Frees all resources used by the heap index and removes it from memory.
dlk       [-d]                                          Displays deadlocks between SyncBlocks and/or ReaderWriterLocks
dumpgen   <GenNum> [-free] [-stat] [-type <TYPE_NAME>]  Dumps the contents of the specified generation
                [-nostrings]
finq      [GenNum] [-stat]                              Displays objects in the finalization queue
frq       [-stat]                                       Displays objects in the Freachable queue
gcgen     <ObjectAddr>                                  Displays the GC generation of the specified object
gch       [HandleType]...                               Lists all GCHandles, optionally filtered by specified handle types
help      [CommandName]                                 Display this screen or details about the specified command
lhi       [filename]                                    LoadHeapIndex - load the heap index into memory.
mbc       <SOSEX breakpoint ID | *>                     Clears the specified or all managed breakpoints
mbd       <SOSEX breakpoint ID | *>                     Disables the specified or all managed breakpoints
mbe       <SOSEX breakpoint ID | *>                     Enables the specified or all managed breakpoints
mbl       [SOSEX breakpoint ID]                         Prints the specified or all managed breakpoints
mbm       <Type/MethodFilter> [ILOffset] [Options]      Sets a managed breakpoint on methods matching the specified filter
mbp       <SourceFile> <nLineNum> [ColNum] [Options]    Sets a managed breakpoint at the specified source code location
mdso      [Options]                                     Dumps object references on the stack and in CPU registers in the current context
mdt       [TypeName | VarName | MT] [ADDR] [Options]    Displays the fields of an object or type, optionally recursively
mdv       [nFrameNum]                                   Displays arguments and locals for a managed frame
mfrag     [-stat] [-mt:<MT>]                            Reports free blocks, the type of object following the free block, and fragmentation statistics
mframe    [nFrameNum]                                   Displays or sets the current managed frame for the !mdt and !mdv commands
mgu       // TODO: Document
mk        [FrameCount] [-l] [-p] [-a]                   Prints a stack trace of managed and unmanaged frames
mln       [expression]                                  Displays the type of managed data located at the specified address or the current instruction pointer
mlocks    [-d]                                          Lists all managed lock objects and CriticalSections and their owning threads
mroot     <ObjectAddr>                                  Displays GC roots for the specified object
mt        (no parameters)                               Steps into the managed method at the current position
mu        [address] [-s] [-il] [-n]                     Displays a disassembly around the current instruction with interleaved source, IL and asm code
muf       [MD Address | Code Address] [-s] [-il] [-n]   Displays a disassembly with interleaved source, IL and asm code
mwaits    [-d]                                          Lists all waiting threads and, if known, the locks they are waiting on
mx        <Filter String>                               Displays managed type/field/method names matching the specified filter string
refs      <ObjectAddr> [-target|-source]                Displays all references from and to the specified object
rwlock    [ObjectAddr | -nd]                            Displays all RWLocks or, if provided a RWLock address, details of the specified lock
sosexhelp [CommandName]                                 Display this screen or details about the specified command
strings   [ModuleAddress] [Options]                     Search the managed heap or a module for strings matching the specified criteria

ListGcHandles - See gch
```

And the Psscor4.dll like this:

```
var controller = (SosController)formMain.field("controller");
var cdbProcess = (CdbProcess)controller.field("cdb");
controller.ExecuteCommand(@".load E:\O2_V4\_O2_V4_TempDir\_ToolsOrAPIs\SosNet\x64\psscor4.dll");
```

```
!help
0:008> ------------------------------------------------------------------------------
SOS is a debugger extension DLL designed to aid in the debugging of managed
programs. Functions are listed by category, then roughly in order of
importance. Shortcut names for popular functions are listed in parenthesis.
Type "!help <functionname>" for detailed info on that function.

Object Inspection                  Examining code and stacks
-----------------------------     -----------------------------
DumpObj (do)                       Threads
DumpArray (da)                     ThreadState
DumpStackObjects (dso)             IP2MD
DumpHeap                           U
DumpVC                             DumpStack
GCRoot                             EEStack
ObjSize                            CLRStack
FinalizeQueue                      GCInfo
PrintException (pe)                EHInfo
TraverseHeap                       BPMD
DumpField (df)                     COMState
DumpDynamicAssemblies (dda)
GCRef
DumpColumnNames (dcn)
DumpRequestQueues
DumpUMService

Examining CLR data structures      Diagnostic Utilities
-----------------------------     -----------------------------
DumpDomain                         VerifyHeap
EEHeap                             VerifyObj
Name2EE                            FindRoots
SyncBlk                            HeapStat
DumpMT                             GCWhere
DumpClass                          ListNearObj (lno)
DumpMD                             GCHandles
Token2EE                           GCHandleLeaks
EEVersion                          FinalizeQueue (fq)
DumpModule                         FindAppDomain
ThreadPool                         SaveModule
DumpAssembly                       ProcInfo
DumpSigElem                        StopOnException (soe)
DumpRuntimeTypes                   DumpLog
DumpSig                            VMMap
RCWCleanupList                     VMStat
DumpIL                             MinidumpMode
PrintIPAddress                     AnalyzeOOM (ao)
DumpHttpRuntime                    FindDebugTrue
DumpIL                             FindDebugModules
PrintDateTime                      Analysis
DumpDataTables                     CLRUsage
DumpAssembly                       CheckCurrentException (cce)
RCWCleanupList                     CurrentExceptionName (cen)
DumpHttpContext
ASPXPages
DumpASPNETCache (dac)
ConvertVTDateToDate (cvtdd)
ConvertTicksToDate (ctd)
DumpRequestTable
DumpHistoryTable
```

Consuming SosNET APIs directly (without using its GUI)

Attach and detach to a running process:

```
var exe = @"E:\O2_V4\_O2_V4_TempDir\6_30_2012\Util - LogViewer [08145]\Util - LogViewer.exe";
var process = exe.startProcess();
process.closeInNSeconds(5);
this.sleep(2000);
var controller = new SosController();

var settings = new CdbSettings()
                                {
                                        CdbPath = (clr.x64())
                                        ? @"C:\Program Files\Windows Kits\8.0\Debuggers\x64"
                                        : @"C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86"
                                };
controller.field("settings",settings);
controller.AttachToProcess(process.Id.str());
"Attached at the moment:{0}".info(controller.Attached);
this.sleep(2000);
controller.Detach();
"Attached at the moment:{0}".info(controller.Attached);
return controller;
```

```
//using SOS.Net
//using SOS.Net.Core.Cdb
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

checking attach and detach times

```
Action attachAndDetach =
       ()=>{
                     var o2Timer = new O2Timer("Attach and detach time").start();

                     controller.field("settings",settings);
                     controller.AttachToProcess(process.Id.str());
                     "Attached at the moment:{0}".info(controller.Attached);
                     //this.sleep(200);
                     controller.Detach();
                     o2Timer.stop();
              };
5.loop(1000,()=>{
                                  attachAndDetach();
                           });
```

Attach and detach times seem to be in the < 100ms range (which are not noticable in the GUI). Note: need to test this on larger projects

```
[12:30:18 PM] DEBUG: Attach and detach time in 0s:70ms
[12:30:18 PM] INFO: Attached at the moment:True
[12:30:17 PM] INFO: Sleeping for: 1000 mili-seconds
[12:30:17 PM] DEBUG: Attach and detach time in 0s:59ms
[12:30:17 PM] INFO: Attached at the moment:True
[12:30:16 PM] INFO: Sleeping for: 1000 mili-seconds
[12:30:16 PM] DEBUG: Attach and detach time in 0s:61ms
[12:30:16 PM] INFO: Attached at the moment:True
[12:30:15 PM] INFO: Sleeping for: 1000 mili-seconds
[12:30:15 PM] DEBUG: Attach and detach time in 0s:59ms
[12:30:15 PM] INFO: Attached at the moment:True
[12:30:14 PM] INFO: Sleeping for: 1000 mili-seconds
[12:30:14 PM] DEBUG: Attach and detach time in 0s:80ms
```

Attach execute command and detach:

```
controller.AttachToProcess(process.Id.str());
var result = controller.ExecuteCommand("!help");
controller.Detach();
return result;
```

load Sos.dll in memory

```
controller.AttachToProcess(process.Id.str());
controller.ExecuteCommand(@".load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\sos.dll");
var result = controller.ExecuteCommand("!help");
controller.Detach();
return result;
```

load correct Sos and dump Domain

```
var sosFile = (clr.x64()) ? @"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\sos.dll"
                                    : @"C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll";
controller.AttachToProcess(process.Id.str());
controller.ExecuteCommand(".load " + sosFile);
var result = controller.ExecuteCommand("!DumpDomain");
controller.Detach();
return result;
```

loading sos.dll in the attached session is also very fast

```
Action attach =
       ()=>{
                     var o2Timer = new O2Timer("Attach time").start();
                     controller.AttachToProcess(process.Id.str());
                     controller.ExecuteCommand(".load " + sosFile);
```

```
                "Attached at the moment:{0}".info(controller.Attached);
                o2Timer.stop();
        };
```

[12:47:00 PM] DEBUG: Attach time in 0s:59ms

Dumping all strings takes about 200ms

```
Action attach =
      ()=>{
                var o2Timer = new O2Timer("Attach time").start();
                controller.AttachToProcess(process.Id.str());
                controller.ExecuteCommand(".load " + sosFile);
                //controller.ExecuteCommand("!DumpDomain");
                //controller.ExecuteCommand("!DumpHeap");
                controller.ExecuteCommand("!dumpheap -mt 000007feeca86728");
                "Attached at the moment:{0}".info(controller.Attached);
                o2Timer.stop();
        };
```

[12:47:15 PM] DEBUG: Attach time in 0s:299ms

Loading all strings directly (there must be a more effecient way to do this, but it works :))

```
//var topPanel = O2Gui.open<Panel>("{name}",700,400);
var exe = @"E:\O2_V4\_O2_V4_TempDir\6_30_2012\Util - LogViewer [08145]\Util - LogViewer.exe";
var process = exe.startProcess();
process.closeInNSeconds(2);
this.sleep(200);
var controller = new SosController();
var settings = new CdbSettings()
                                {
                                        CdbPath = (clr.x64())
                                        ? @"C:\Program Files\Windows Kits\8.0\Debuggers\x64"
                                        : @"C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86"
                                };

controller.field("settings",settings);
var clr2 = clr.clr2();
var x64 = clr.x64();
var sosFile = (clr2) ?     (x64) ? @"C:\Windows\Microsoft.NET\Framework64\v2.0.50727\sos.dll"
                                                : @"C:\Windows\Microsoft.NET\Framework64\v2.0.50727
\sos.dll"
                                :      (x64) ? @"C:\Windows\Microsoft.NET\Framework64\v4.0.30319
\sos.dll"
                                                : @"C:\Windows\Microsoft.NET\Framework\v4.0.30319
\sos.dll";

Action attach =
      ()=>{
                controller.AttachToProcess(process.Id.str());
                controller.ExecuteCommand(".load " + sosFile);
                "Attached at the moment:{0}".info(controller.Attached);
        };

var o2Timer = new O2Timer("get data").start();
attach();
var cdbProcess = (CdbProcess)controller.field("cdb");
var instanceInfoCommand = new InstanceInfoCommand("000007feeca86728");

var result =  instanceInfoCommand.Result(cdbProcess);
var instances = (from item in result
                        select item.Value).Take(10000).toList();

var strings = new List<string>();

var count = 0;
foreach(var instance in instances)
{
      var  output = cdbProcess.ExecuteCommand("!DumpObj {0}".format(instance.Address));
      if (output.split_onLines().size()>6)
      {

            var value = output.split_onLines()[5].subString(13);
            strings.add(value);

      }
```

```
        if (count++ % 25 ==0)
                "[{0} / {1}]".info(count, instances.size());
}


//var result = controller.ExecuteCommand("!dumpheap -mt 000007feeca86728");
//result = controller.ExecuteCommand("!DumpHeap");

controller.Detach();
o2Timer.stop();
return strings;
return controller;


//using SOS.Net
//using SOS.Net.Core.Cdb
//using SOS.Net.Core.Cdb.Commands
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

[1:06:25 PM] DEBUG: get data in 14s:966ms
[1:06:02 PM] DEBUG: get data in 13s:997ms

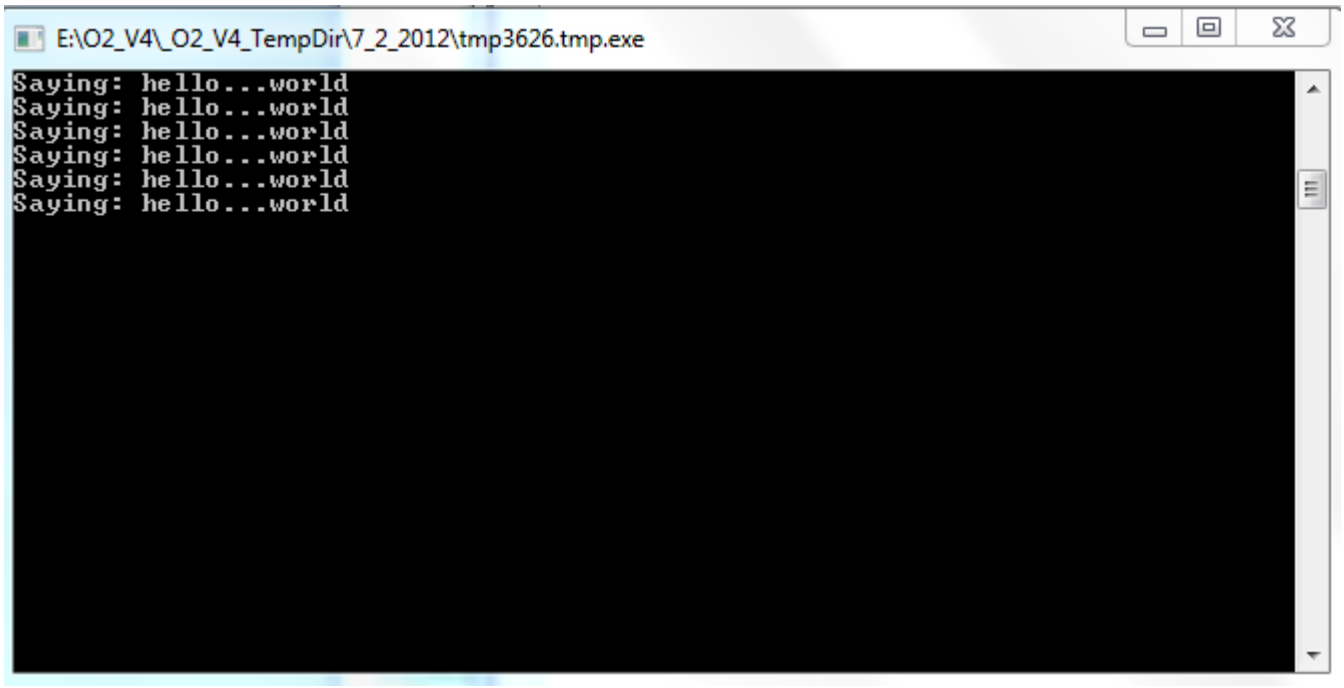
Changing the value of a in memory string

Script that creates a new exe with a continous loop and opens up a script window with the sosnet controller

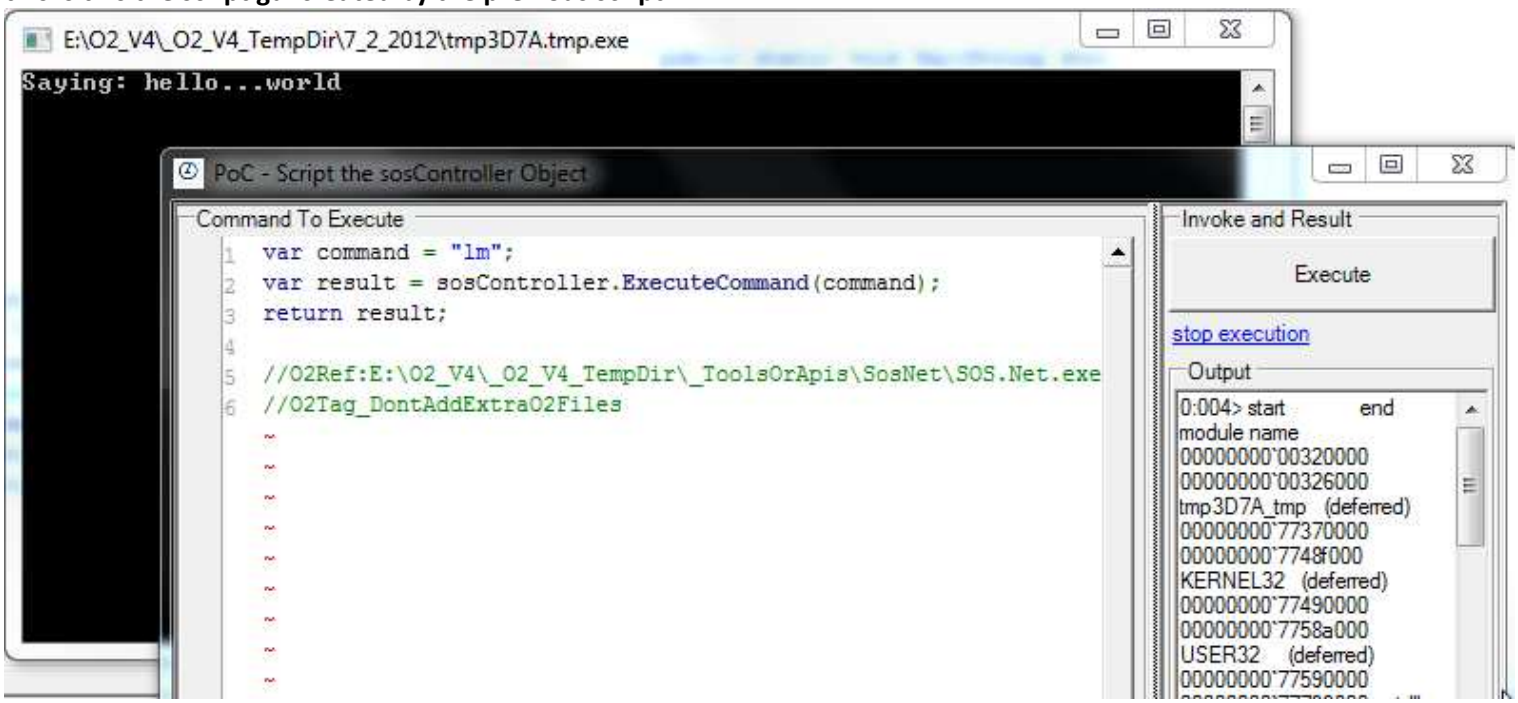
```
//var topPanel = O2Gui.open<Panel>("{name}",700,400);
"exeToDebug".o2Cache(null);
var exeToDebug = "exeToDebug".o2Cache<string>(()=>{
                                                                        "Creating temp
exe".info();
                                                                        var code = @"using
System;
                                                                        using
System.Diagnostics;
                                                                        public class
Program
                                                                        {
                                                                                public
static string msg1 = ""hello"";
                                                                                public
static void Main(String[] args)
                                                                                {


        string msg2 = ""world"";
        while(true)
                                                                                        {


        Say(msg1 + ""..."" + msg2);
        System.Threading.Thread.Sleep(500);
                                                                                        }
                                                                                }


                                                                                public
static void Say(String str)
                                                                                {

        Console.WriteLine(""Saying: "" + str);
                                                                                }
                                                                        }";


                                                                        return
code.createExe();
                                                                }});


if (exeToDebug.isNull())
return "compilation error";
var process = exeToDebug.startProcess();
//process.closeInNSeconds(2);
```

```
this.sleep(200);
var controller = new SosController();
var settings = new CdbSettings()
                                {
                                        CdbPath = (clr.x64())
                                        ? @"C:\Program Files\Windows Kits\8.0\Debuggers\x64"
                                        : @"C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86"
                                };

controller.field("settings",settings);
var clr2 = clr.clr2();
var x64 = clr.x64();
var sosFile = (clr2) ?     (x64) ? @"C:\Windows\Microsoft.NET\Framework64\v2.0.50727\sos.dll"
                                                : @"C:\Windows\Microsoft.NET\Framework64\v2.0.50727
\sos.dll"
                                :      (x64) ? @"C:\Windows\Microsoft.NET\Framework64\v4.0.30319
\sos.dll"
                                                : @"C:\Windows\Microsoft.NET\Framework\v4.0.30319
\sos.dll";

Action attach =
       ()=>{
                        controller.AttachToProcess(process.Id.str());
                        controller.ExecuteCommand(".load " + sosFile);
                        "Attached at the moment:{0}".info(controller.Attached);
                };

var o2Timer = new O2Timer("get data").start();
attach();



var result = controller.ExecuteCommand("!help");
result = controller.ExecuteCommand("!DumpHeap");
controller.script_Me();
//controller.Detach();

o2Timer.stop();
return controller;

//O2File:API_ConsoleOut.cs
//O2File:_Extra_methods_Roslyn_API.cs

//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll


//using SOS.Net
//using SOS.Net.Core.Cdb
//using SOS.Net.Core.Cdb.Commands
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

This is what the original exe executes like:

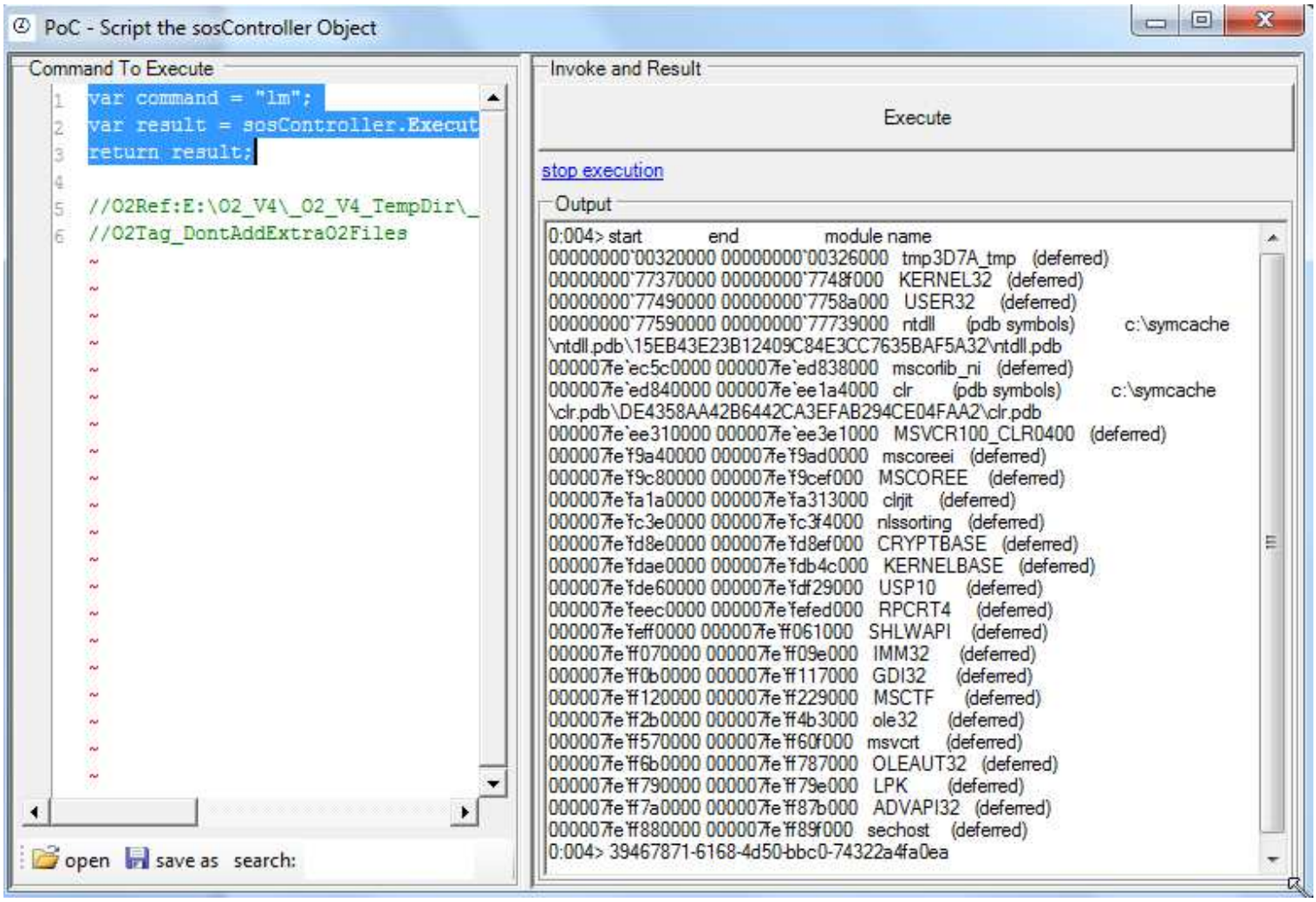


this is this the script gui created by the previous script:



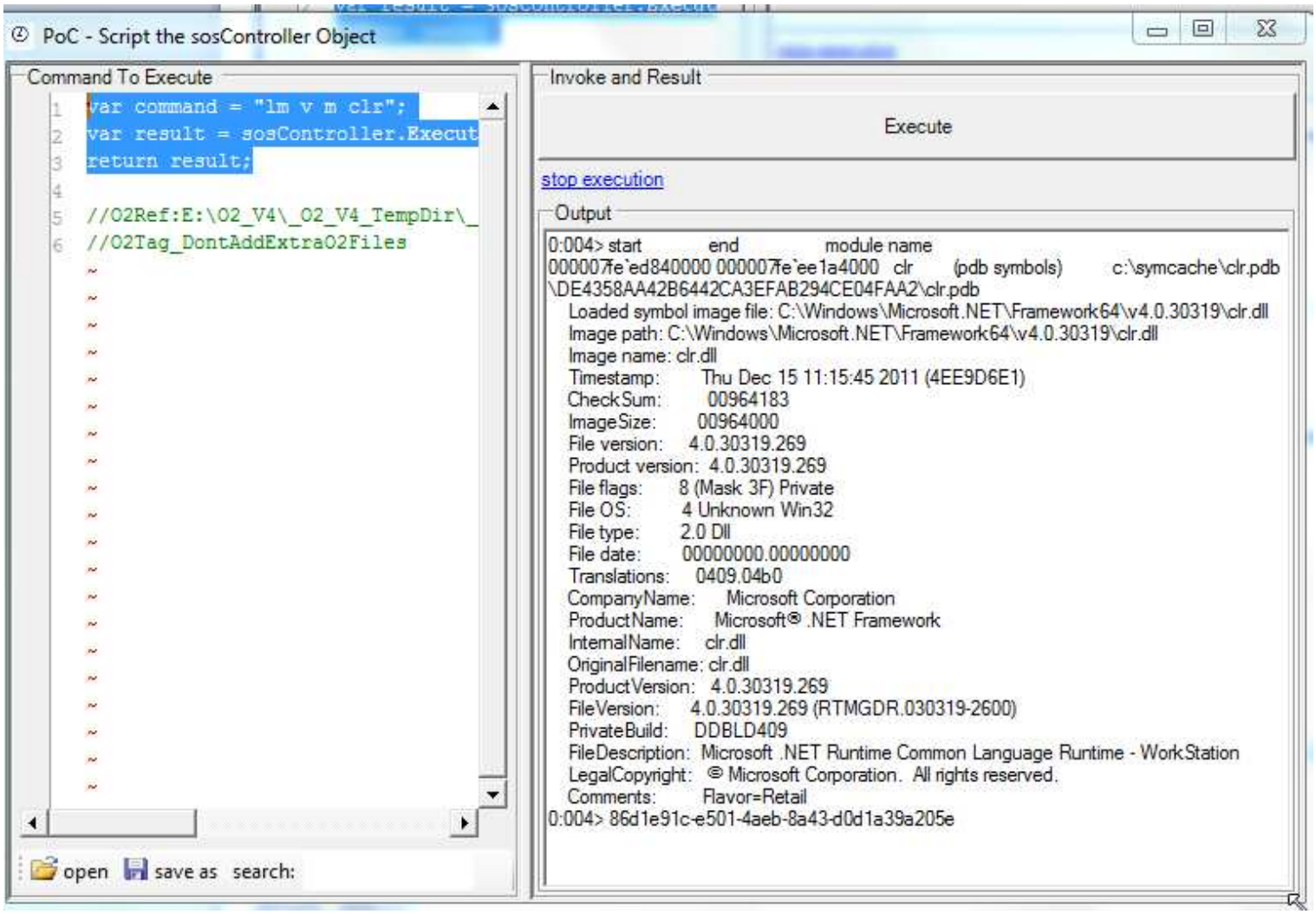
Loaded Modules

```
var command = "lm";
var result = sosController.ExecuteCommand(command);
return result;
```

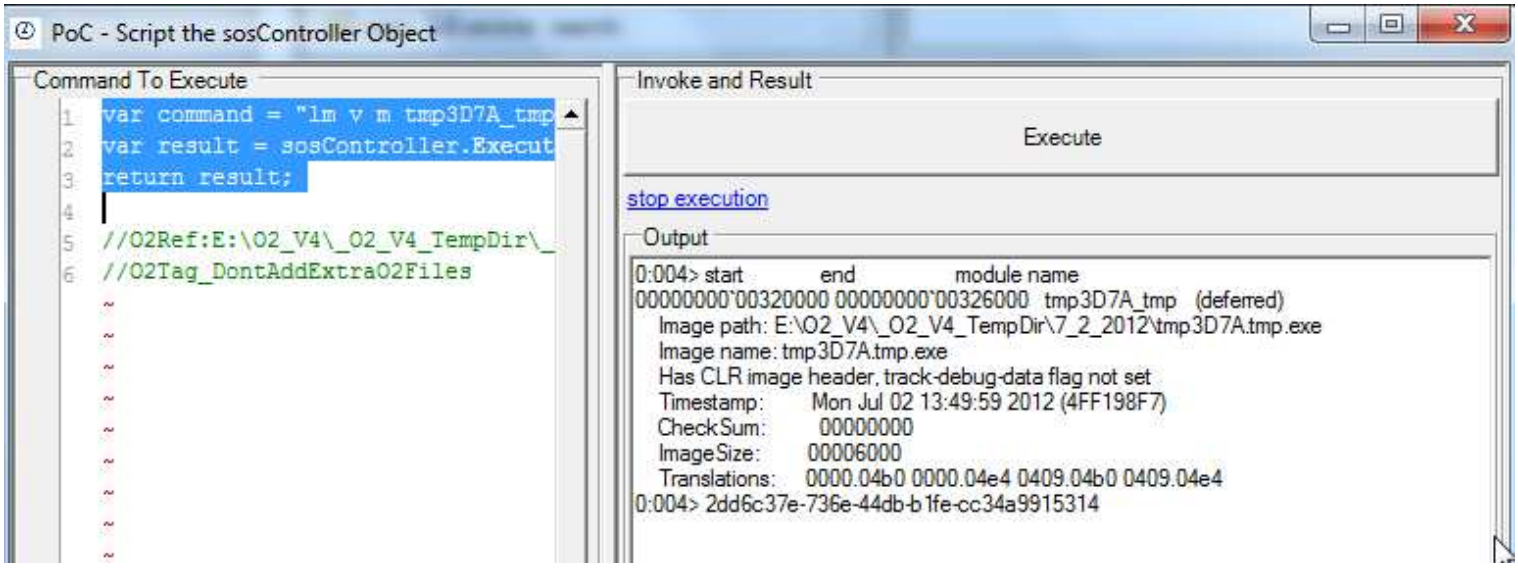


view a particular module details:

```
var command = "lm v m clr";
var result = sosController.ExecuteCommand(command);
return result;
```

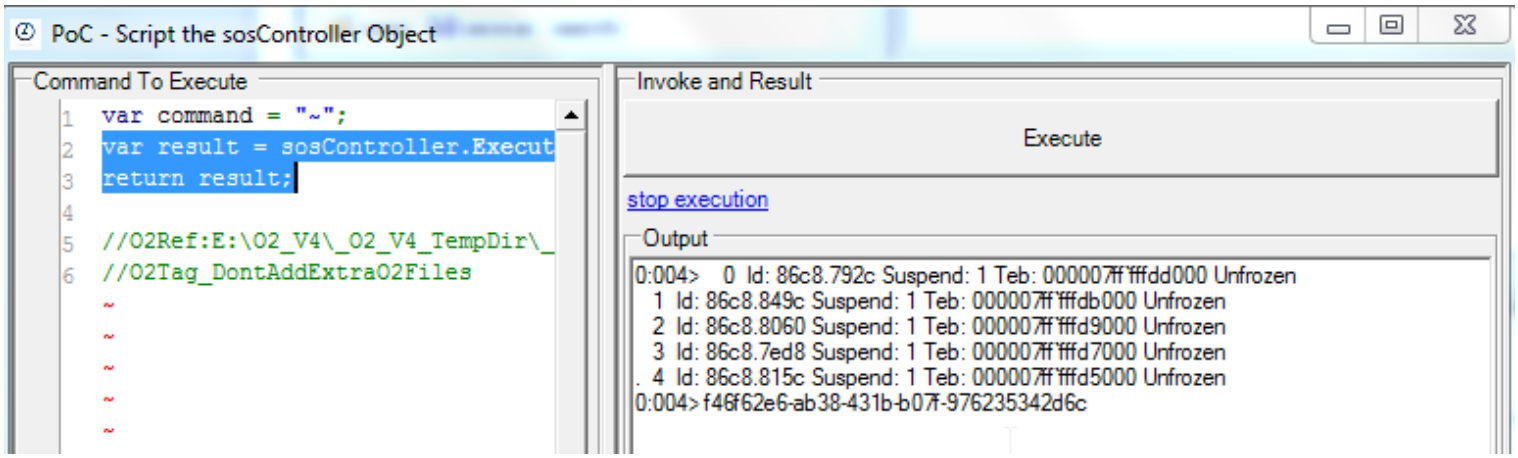



```
var command = "lm v m tmp3D7A_tmp";
var result = sosController.ExecuteCommand(command);
return result;
```



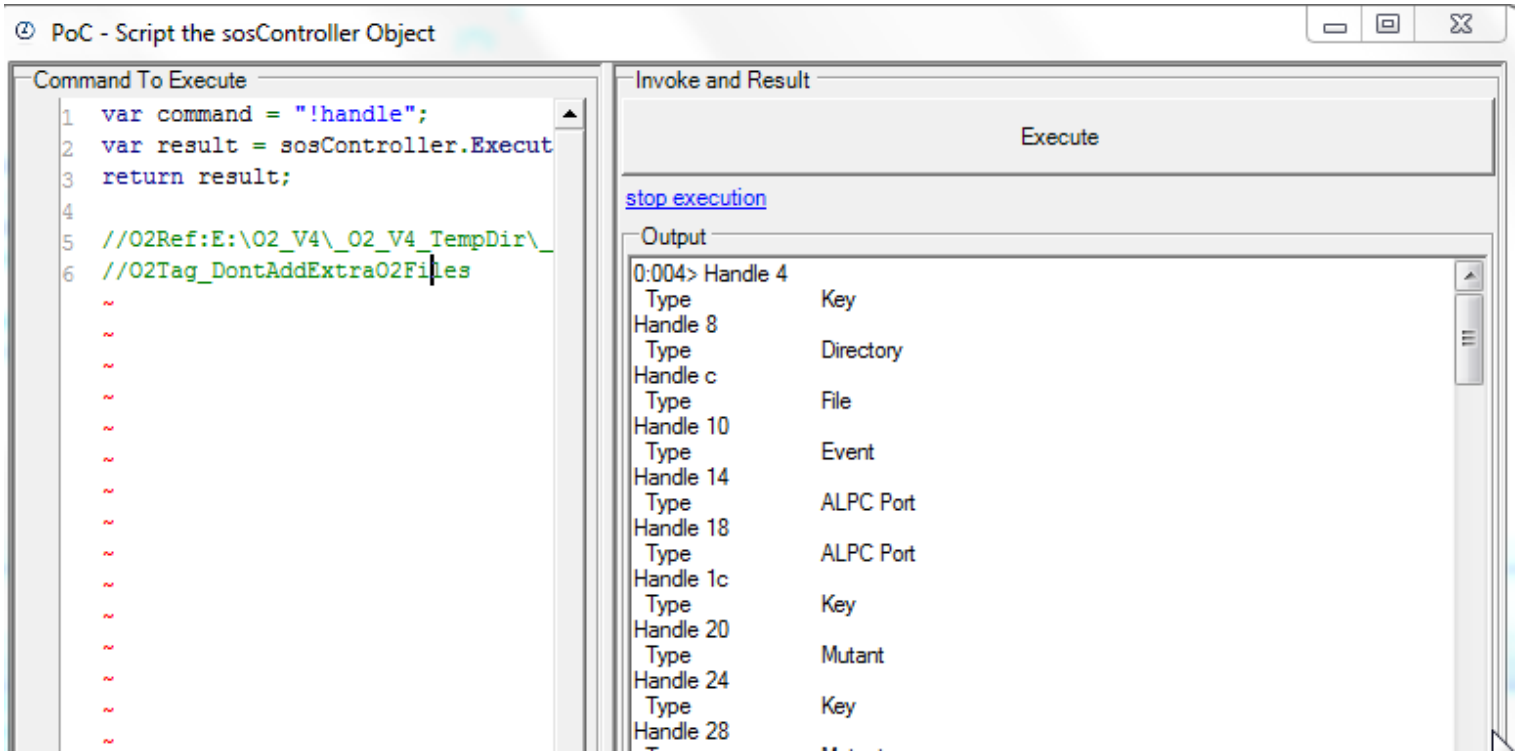
View native threads

```
var command = "~";
var result = sosController.ExecuteCommand(command);
return result;
```



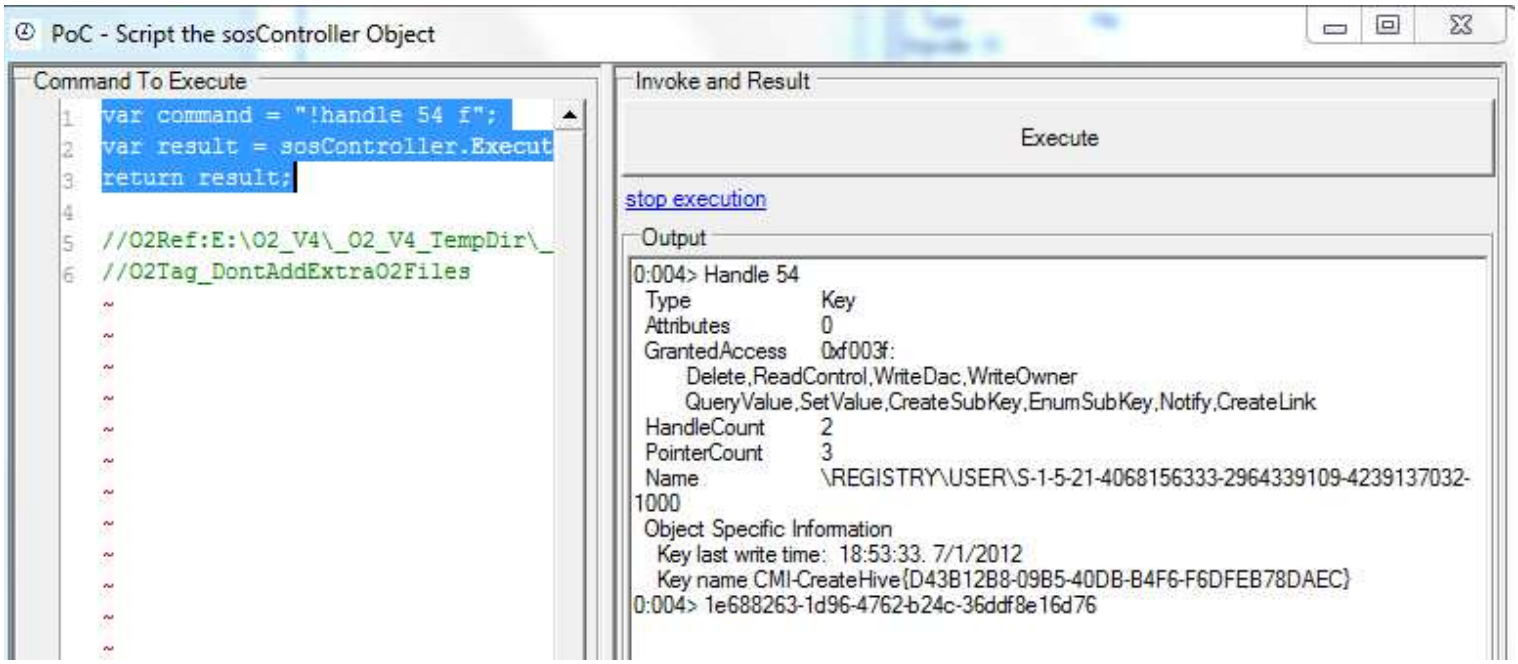
view native handles:

```
var command = "!handle";
var result = sosController.ExecuteCommand(command);
return result;

//O2Ref:E:\O2_V4\_O2_V4_TempDir\_ToolsOrApis\SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```



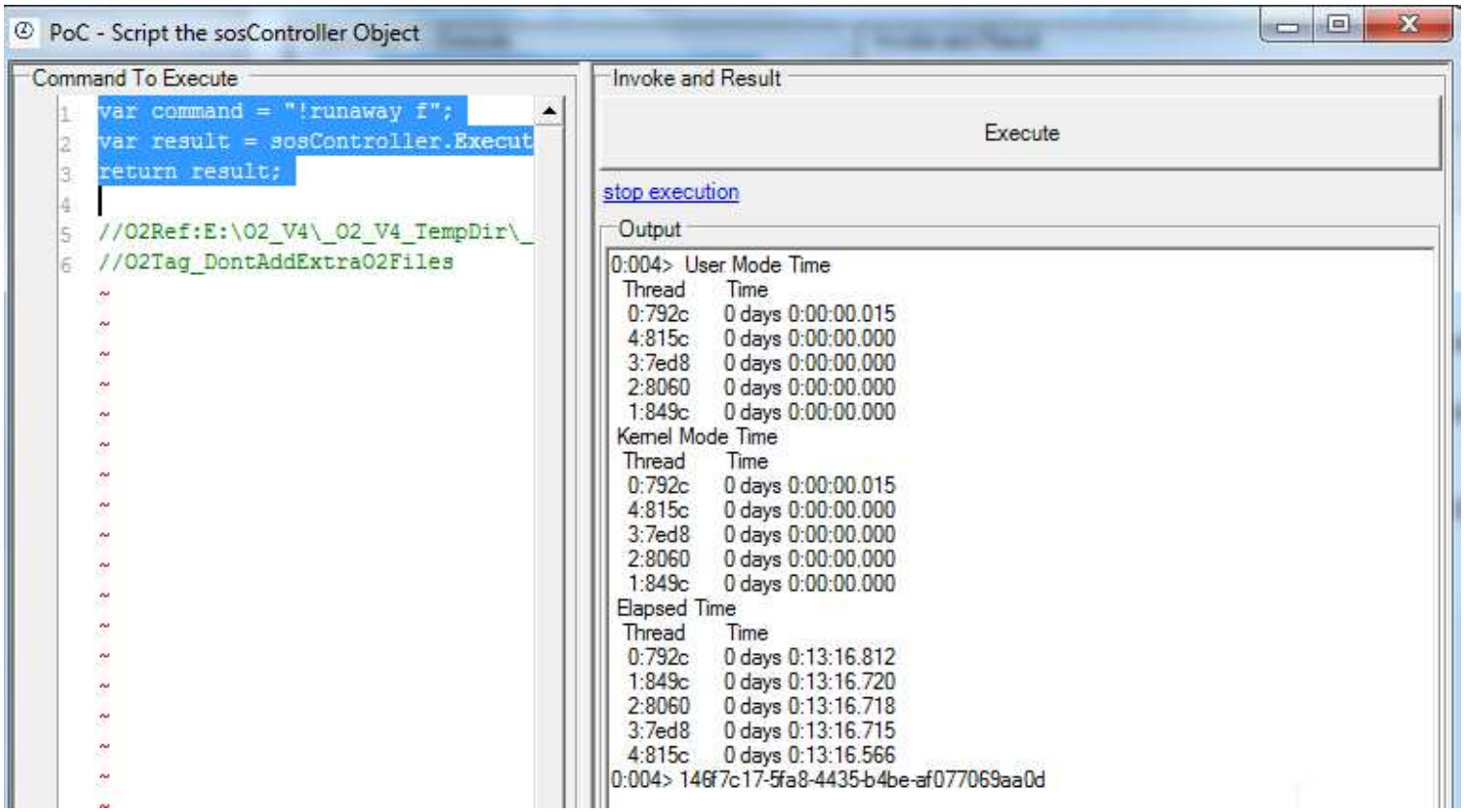
view handle details:

```
var command = "!handle 54 f";
var result = sosController.ExecuteCommand(command);
return result;
```



View thread times:

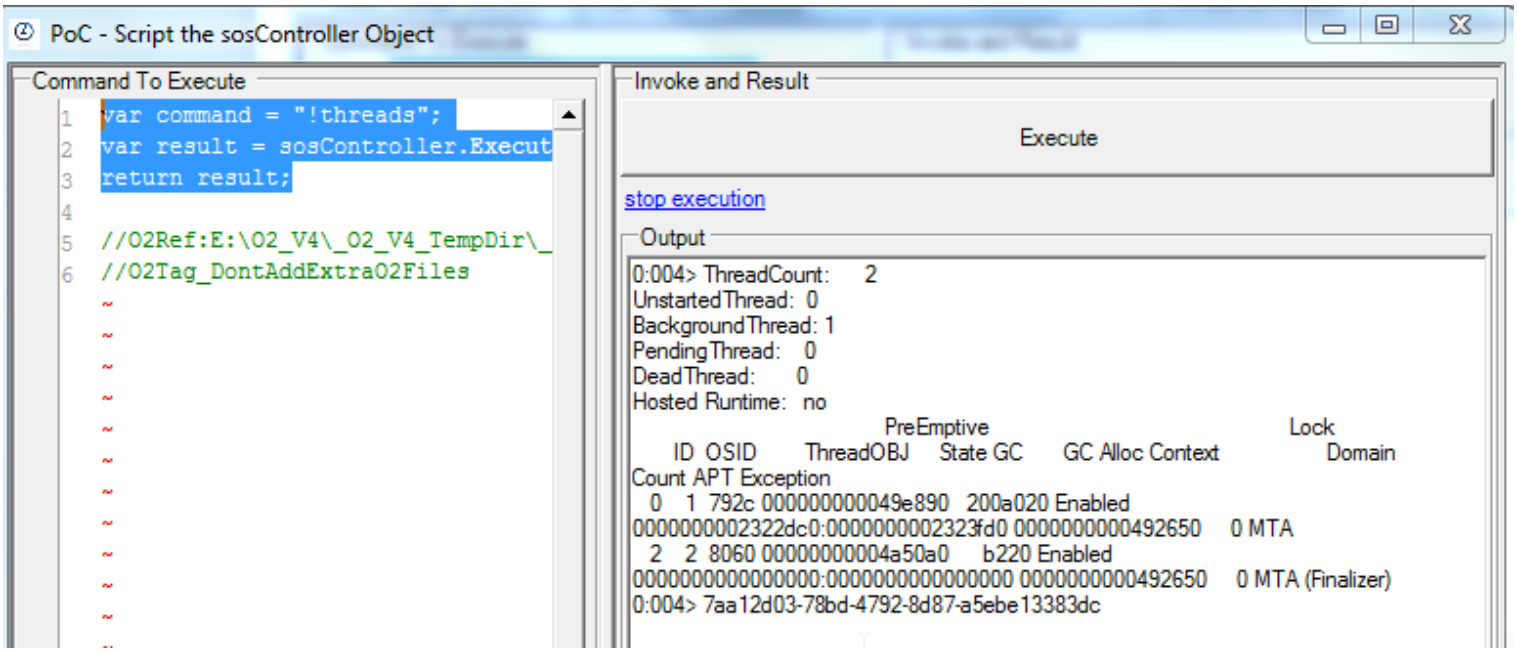
```
var command = "!runaway f";
var result = sosController.ExecuteCommand(command);
return result;
```



These now use the methods provided by sos.dll

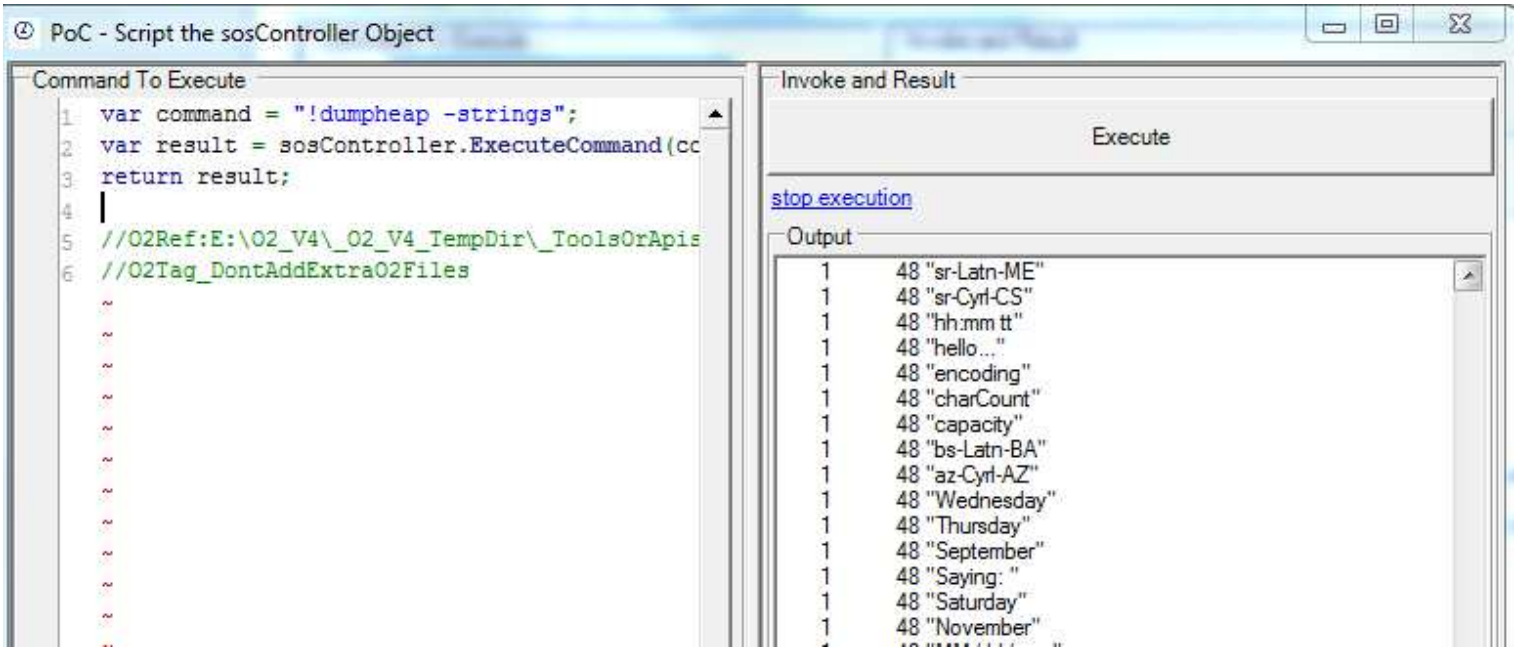
Managed threads

```
var command = "!threads";
var result = sosController.ExecuteCommand(command);
return result;
```



viewing all strings

```
var command = "!dumpheap -strings";
var result = sosController.ExecuteCommand(command);
return result;
```

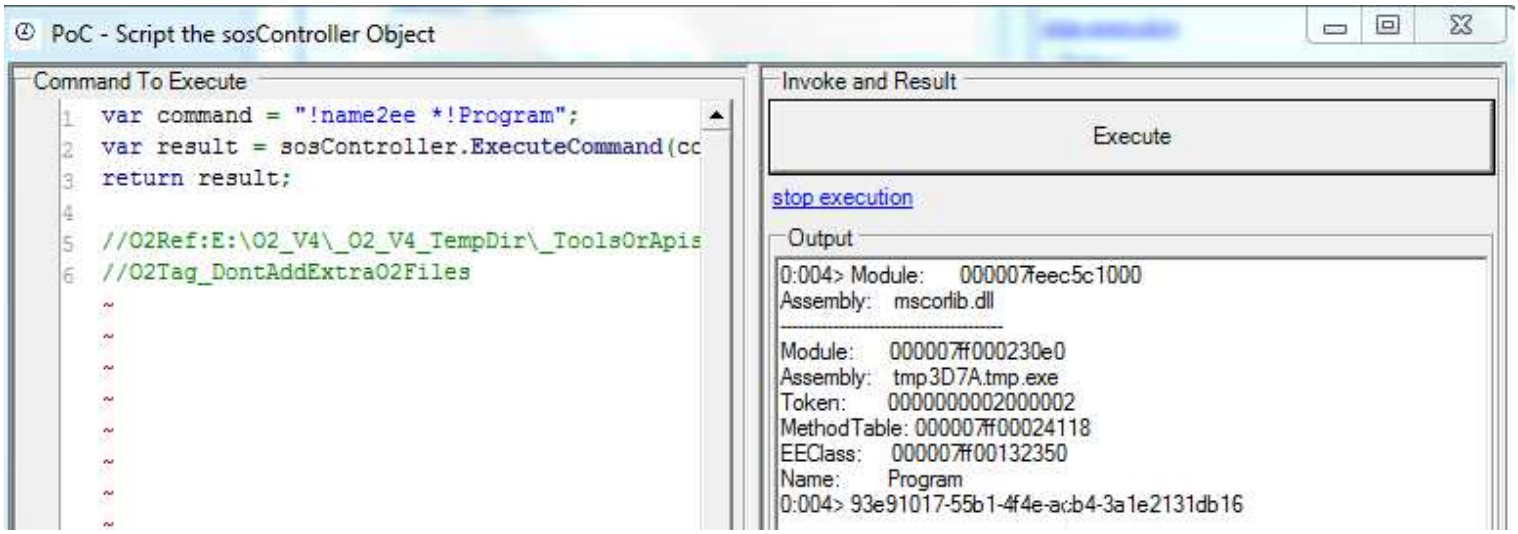


Find type location

```
var command = "!name2ee *!Program";
var result = sosController.ExecuteCommand(command);
return result;
```

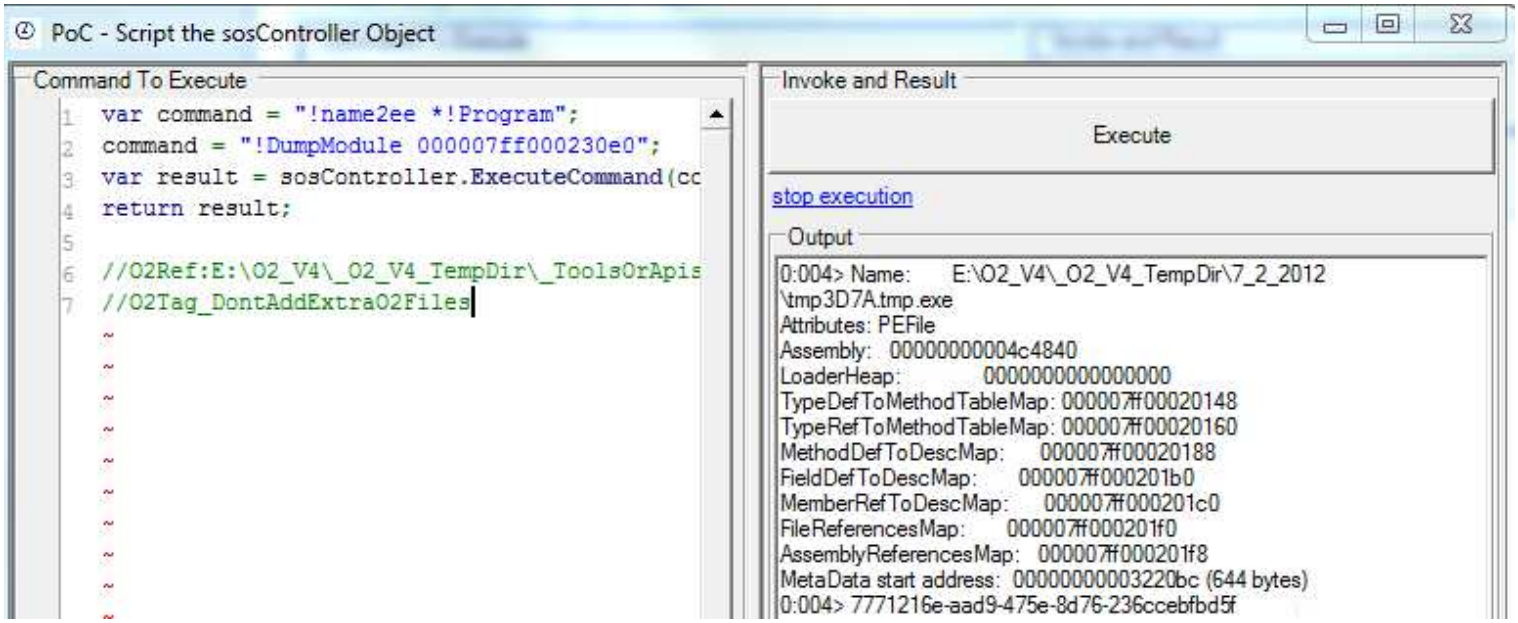
or

```
var command = "!name2ee tmp3D7A.tmp.exe!Program";
var result = sosController.ExecuteCommand(command);
return result;
```



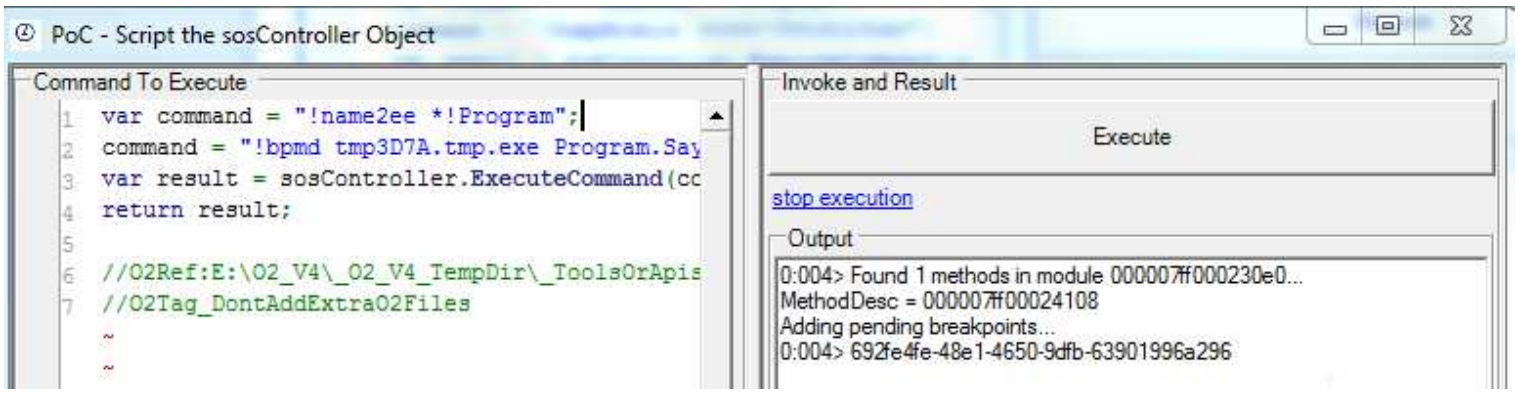
Dumping module:

```
var command = "!name2ee *!Program";
command = "!DumpModule 000007ff000230e0";
var result = sosController.ExecuteCommand(command);
return result;
```



adding breakpoint

```
command = "!bpmd tmp3D7A.tmp.exe Program.Say";
var result = sosController.ExecuteCommand(command);
return result;
```

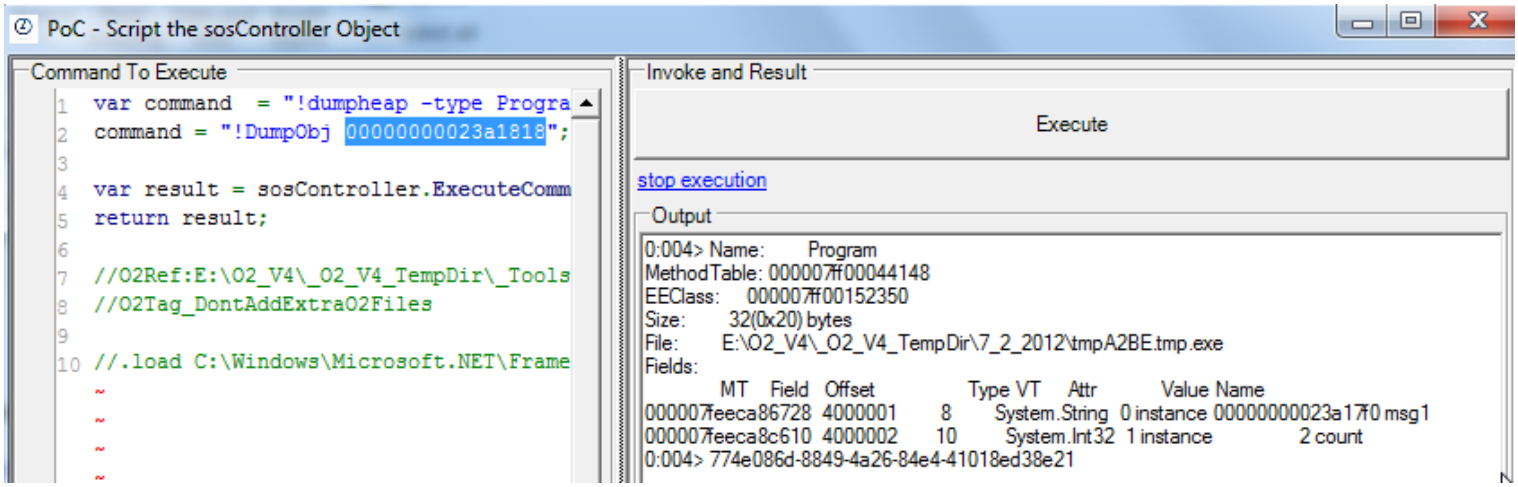


Finding a type's object:

```
var command= "!dumpheap -type Program";
var result = sosController.ExecuteCommand(command);
return result;
```

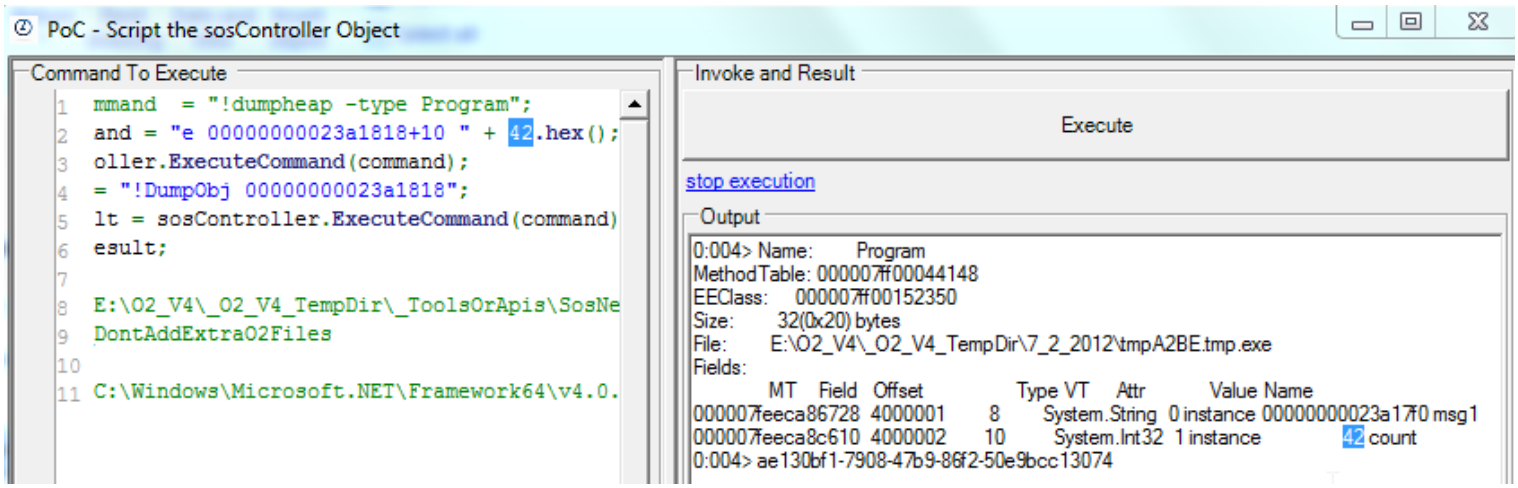
dump the Address value to see the object details:

```
command = "!DumpObj 00000000023a1818";

var result = sosController.ExecuteCommand(command);
return result;
```



Change the value of an Int in menory (from 2 to 42)

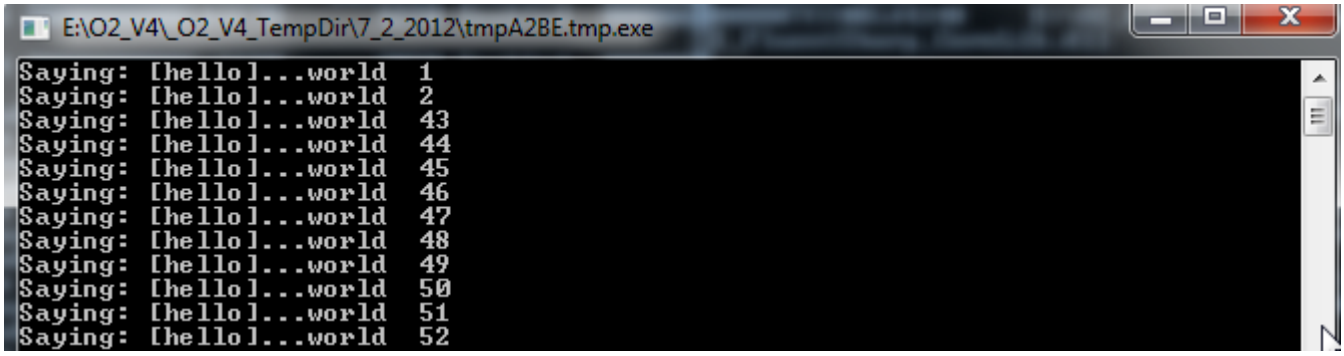
```
var command = "ed 00000000023a1818+10 " + 42.hex(); //42 = 2A;
sosController.ExecuteCommand(command);
command = "!DumpObj 00000000023a1818";
var result = sosController.ExecuteCommand(command);
return result;
```



Continue execution

```
command= "g";
var result = sosController.ExecuteCommand(command);
return result;
```

And the count id will jump from 2 to 42:

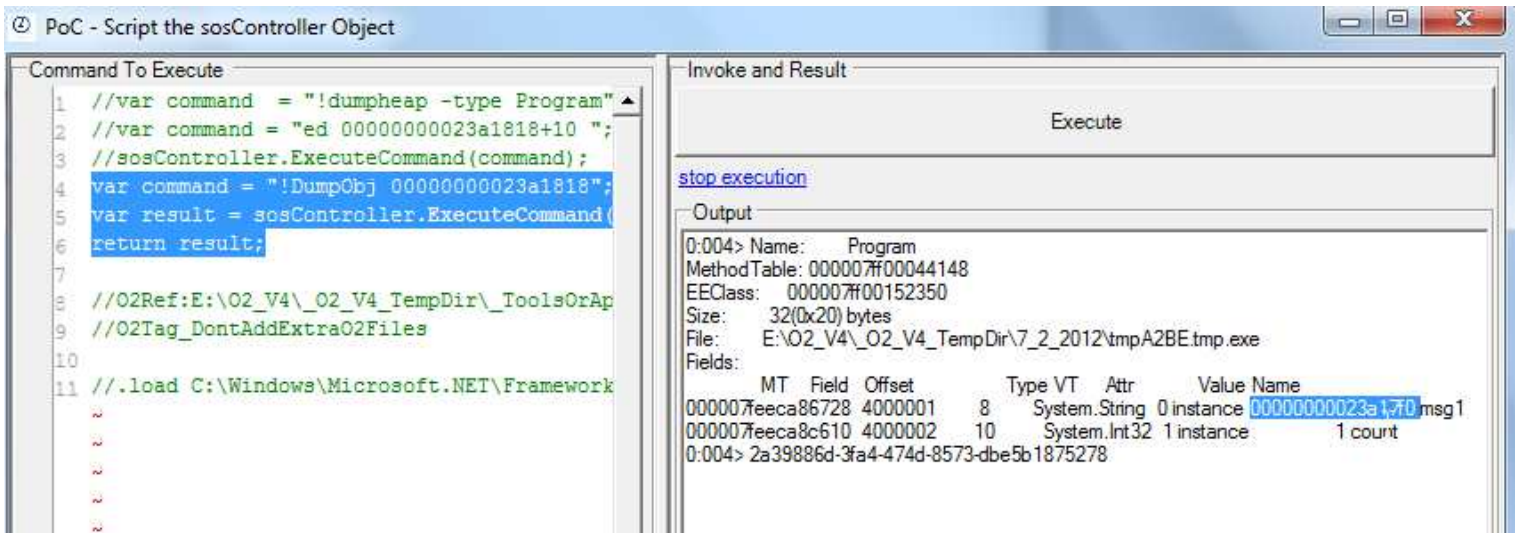


note that 'g' command will hang the script GUI until Ctrl+C is pressed on the console window

To change the value of the string, we need to go back to the dump of the Program object

```
var command = "!DumpObj 00000000023a1818";
var result = sosController.ExecuteCommand(command);
return result;
```

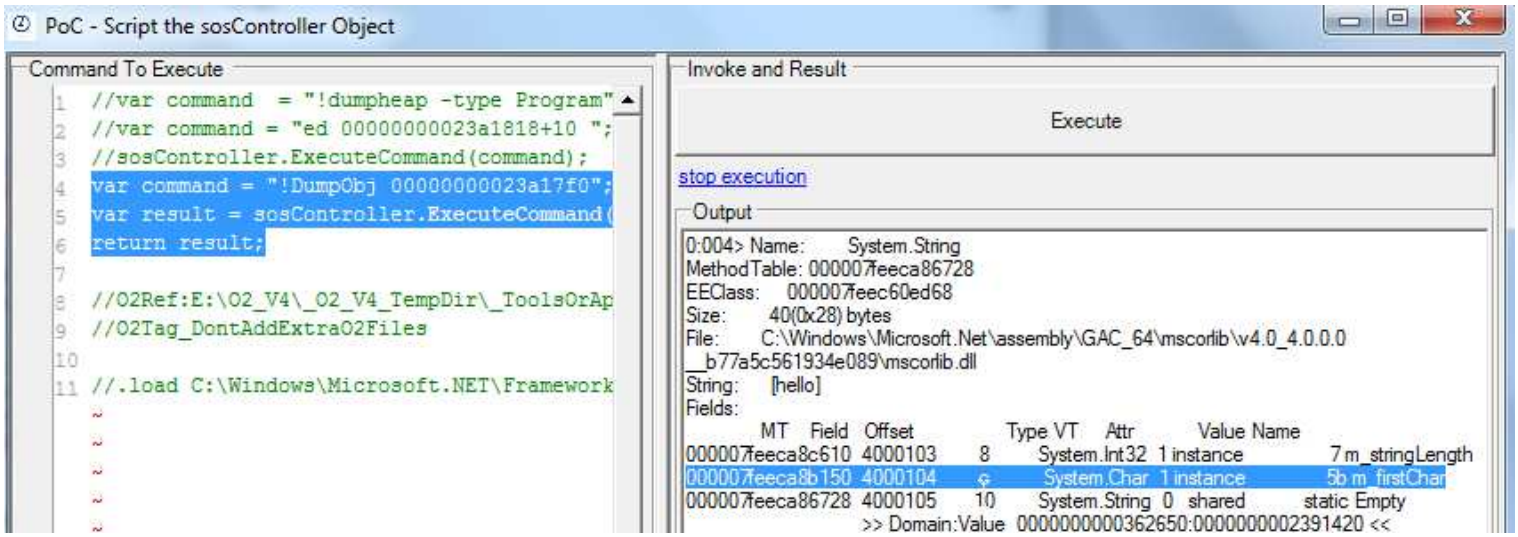
And get the address value of the string object



Dump its value:

```
var command = "!DumpObj 00000000023a17f0";
var result = sosController.ExecuteCommand(command);
return result;
```

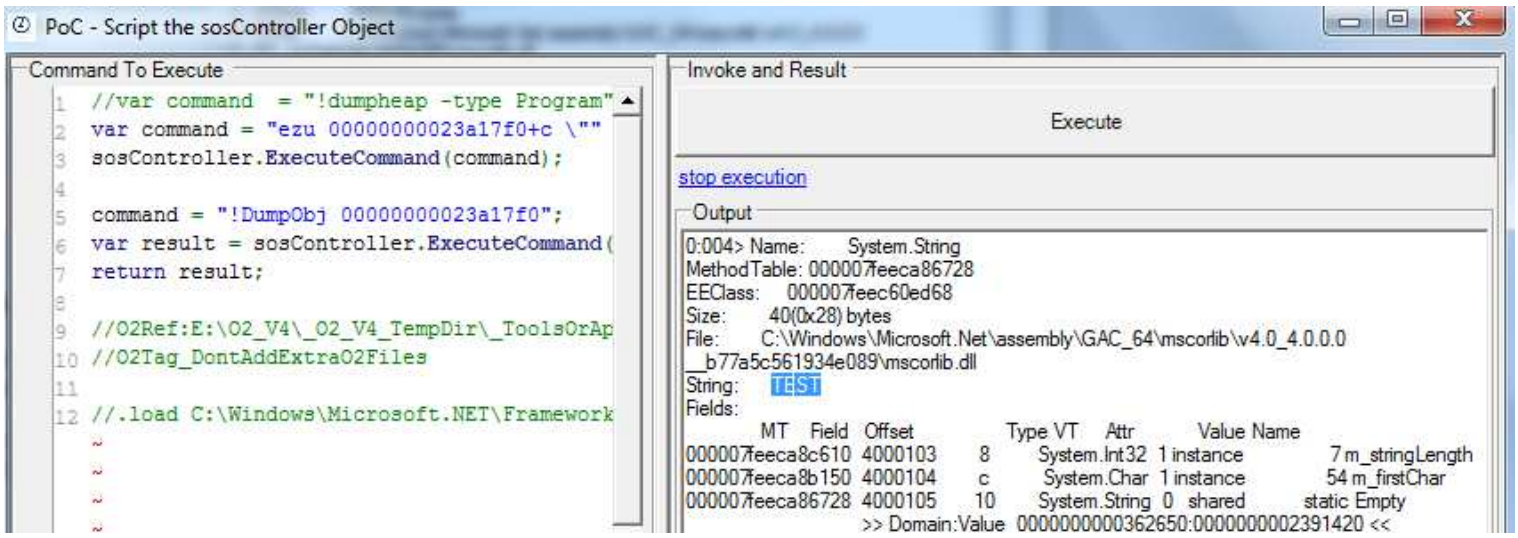
Note the offset where the char array starts (in this case 'C' , ie 12):



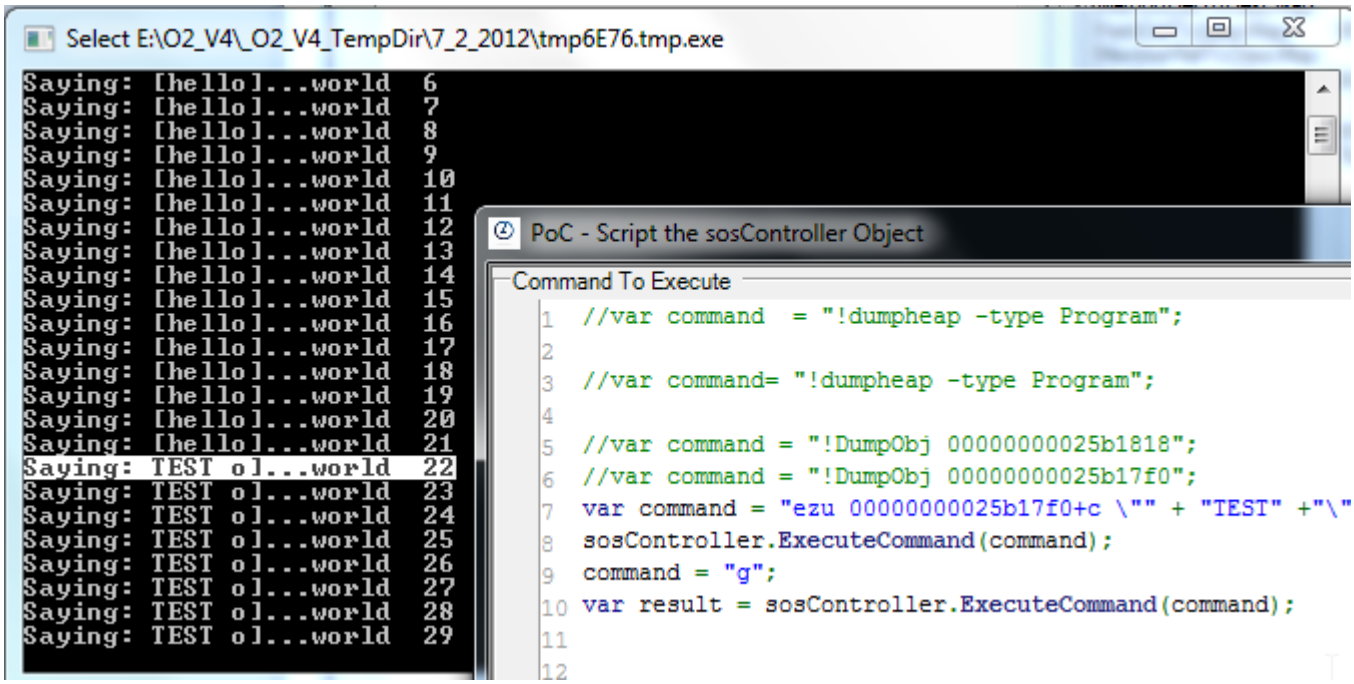
Change the string value (note that the new string has to be smaller than the existing one)

```
var command = "ezu 00000000023a17f0+c \"" + "TEST" +"\"";
sosController.ExecuteCommand(command);

command = "!DumpObj 00000000023a17f0";
var result = sosController.ExecuteCommand(command);
return result;
```

After execution the value is changed:



and now the string value was changed



Here are the sequence of commands to be executed:

```
//var command= "!dumpheap -type Program";
//var command = "!DumpObj 00000000025b1818";
//var command = "!DumpObj 00000000025b17f0";
var command = "ezu 00000000025b17f0+c \"" + "TEST" +"\"";
sosController.ExecuteCommand(command);
command = "g";
var result = sosController.ExecuteCommand(command);
```

Creating ExtensionMethods to make this easier

Execute Command

```
var result = sosController.executeCommand("!DumpHeap -type Program");
return result;
```

Execute command (for chained requests)

```
var result = "";
sosController.execute("!help", ref result);
return result;
```

or just (with the result shown on the Log Viewer)

```
sosController.execute("!DumpHeap");
```

```
21 sosController.execute("!DumpHeap");
22
```

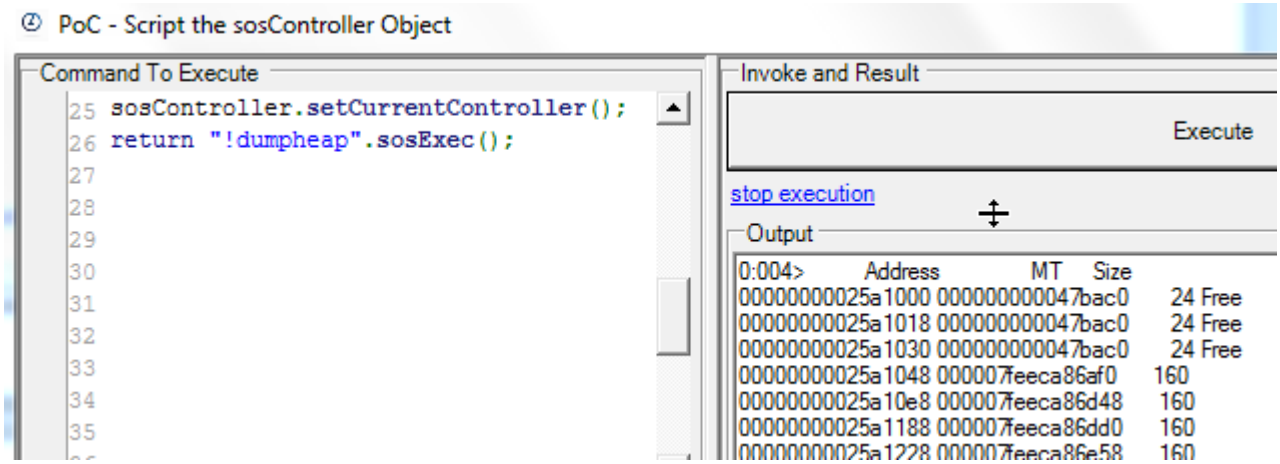
open save as search:

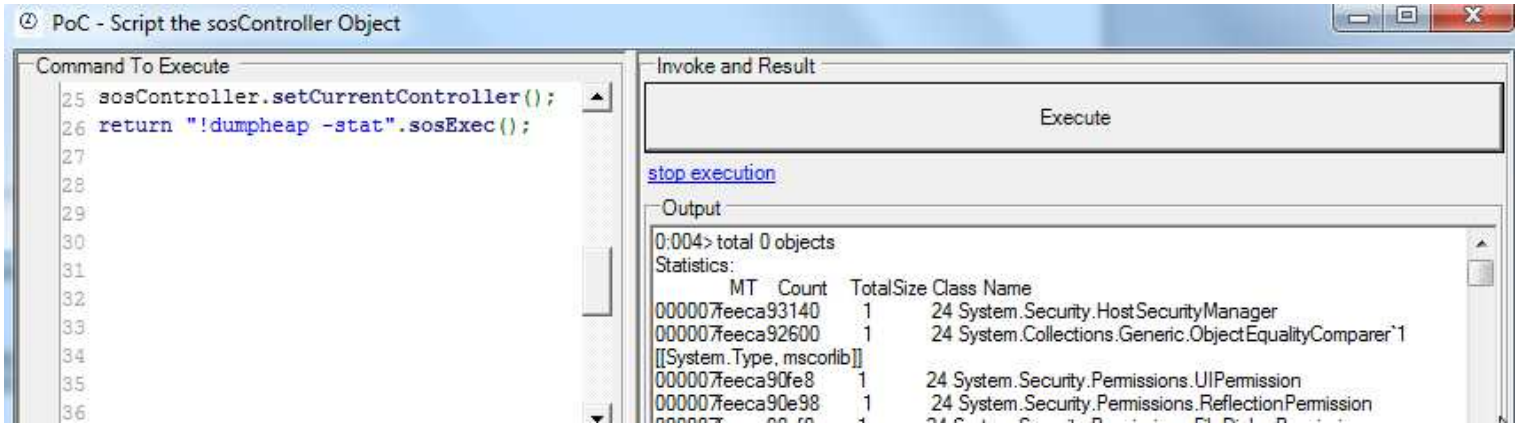
Log Viewer

```
[5:37:58 PM] INFO: [SosController] ExecuteCommand: !DumpHeap

0:004>        Address           MT     Size
00000000025a1000 000000000047bac0       24 Free
00000000025a1018 000000000047bac0       24 Free
00000000025a1030 000000000047bac0       24 Free
00000000025a1048 000007feeca86af0      160
00000000025a10e8 000007feeca86d48      160
```

Execute from string (not Thread safe the SosController is stored on a global static variable)

```
sosController.setCurrentController();
return "!dumpheap".sosExec();
```

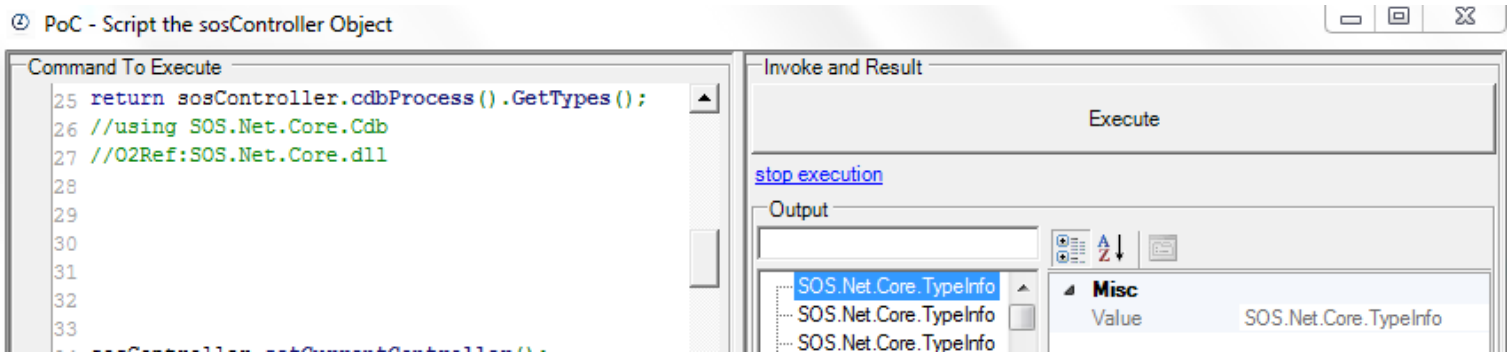




Get CdbProcess object

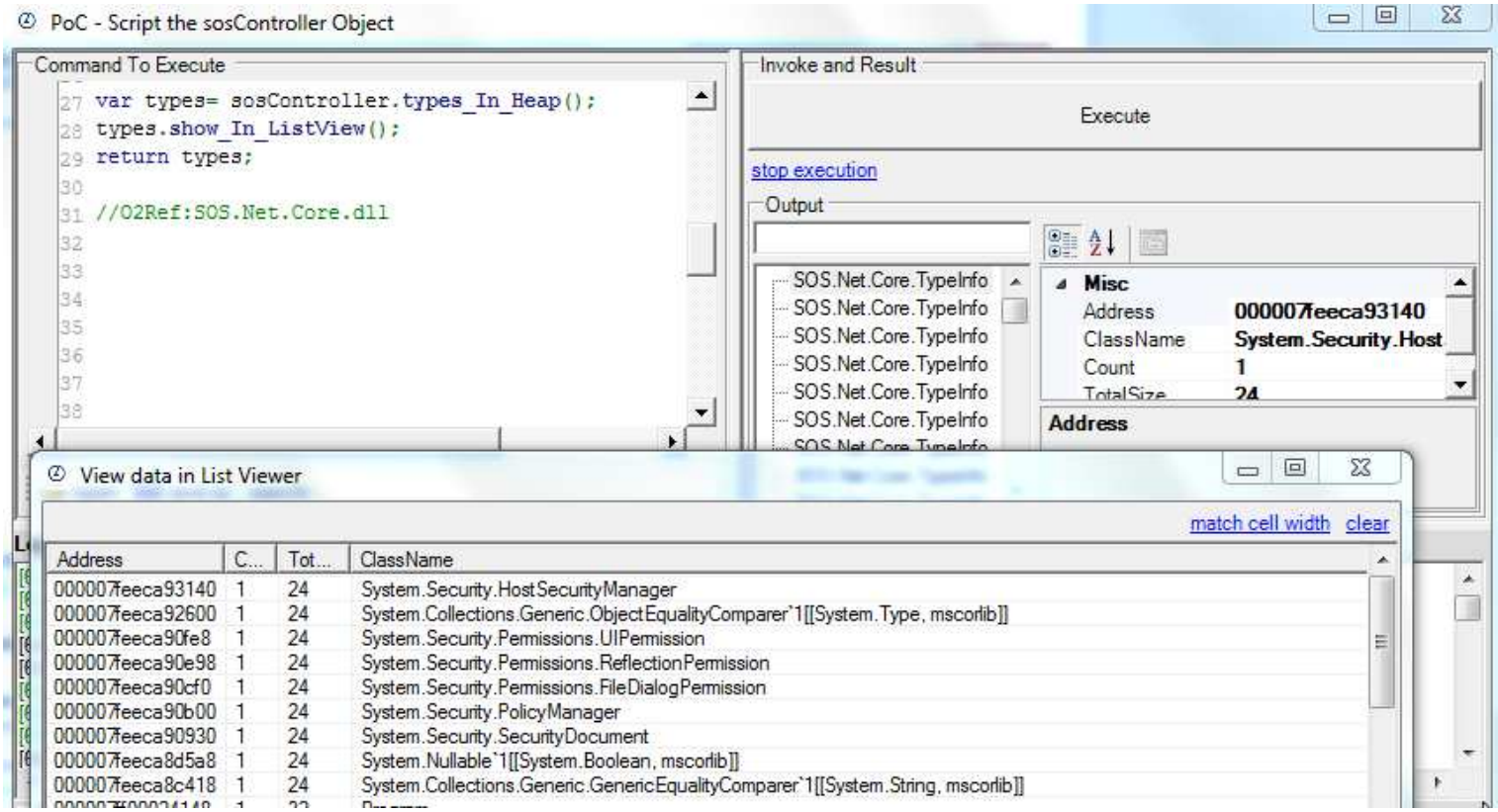
```
return sosController.cdbProcess();
//O2Ref:SOS.Net.Core.dll
```

There is an SoSNet extension method to get the current Types (in a nice TypeInfo object)



Which is easier to consume like this:

```
var types= sosController.types_In_Heap();
types.show_In_ListView();
return types;

//O2Ref:SOS.Net.Core.dll
```



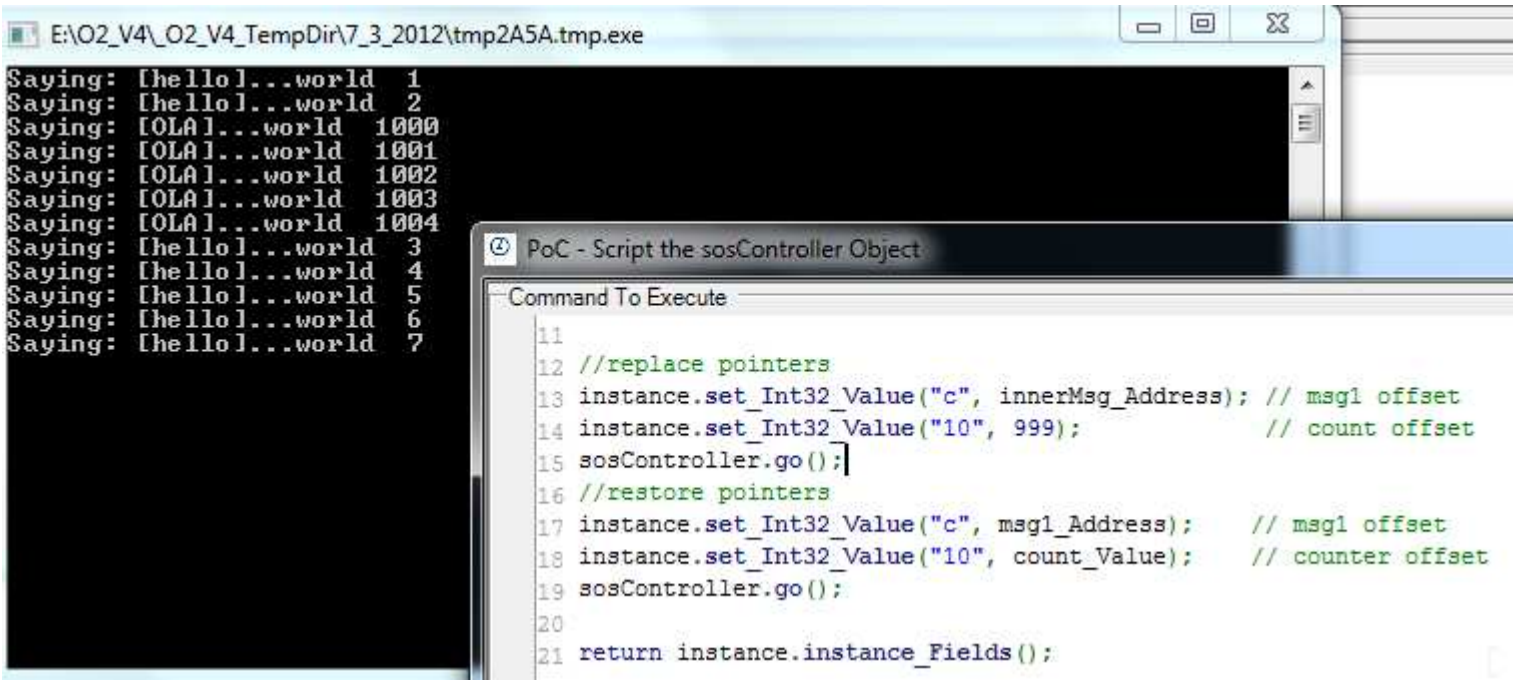
Dynamically changing string and int in real time:

```
sosController.setCurrentController();
var rawHeapTypes = sosController.types_In_Heap_Raw("Program");
var programType = rawHeapTypes.first() ;

var msg1 = programType.instance_Fields().name("msg1");

programType.set_Int32_Value("10", 1);

msg1.set_String_Value_DangerousWay("aaa");
sosController.go();
sosController.setCurrentController();
programType.set_Int32_Value("10", 1);
msg1.set_String_Value_DangerousWay("bbb");
sosController.go();
sosController.setCurrentController();
```

```
msg1.set_String_Value_DangerousWay("ccc");
var count = programType.instance_Fields().name("count");
sosController.go();

//O2File:SoSNet_ExtensionMethods.cs

//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

Dynamically repointing Strings values

```
//press Ctrl+C twice on the cmd window to break from the go() commands

sosController.setDefaultController();
var instance = sosController.type_In_Heap("Program")
                                        .instances().first();
var result = instance.instance_Fields();

var msg1_Address =  instance.instance_Field("msg1").Value;
var innerMsg_Address = instance.instance_Field("innerMsg").Value;
var count_Value  = instance.instance_Field("count").Value;

//replace pointers
instance.set_Int32_Value("c", innerMsg_Address); // msg1 offset
instance.set_Int32_Value("10", 999);                 // count offset
sosController.go();
//restore pointers
instance.set_Int32_Value("c", msg1_Address);   // msg1 offset
instance.set_Int32_Value("10", count_Value);   // counter offset
sosController.go();

return instance.instance_Fields();

//return sosController.types_In_Heap();
//return sosController.sos_Help();

//O2Ref:SOS.Net.exe
//O2Ref:SOS.Net.Core.dll
//O2Tag_DontAddExtraO2Files
//O2File:SoSNet_ExtensionMethods.cs
```



Adding a breakpoint

!bpmd E:\O2_V4_O2_V4_TempDir\7_3_2012\tmpD0D0.tmp.exe Program.Say

Poc to add breakpoint make changes and resume

script that creates the exe, starts the process and gives a script editor for it

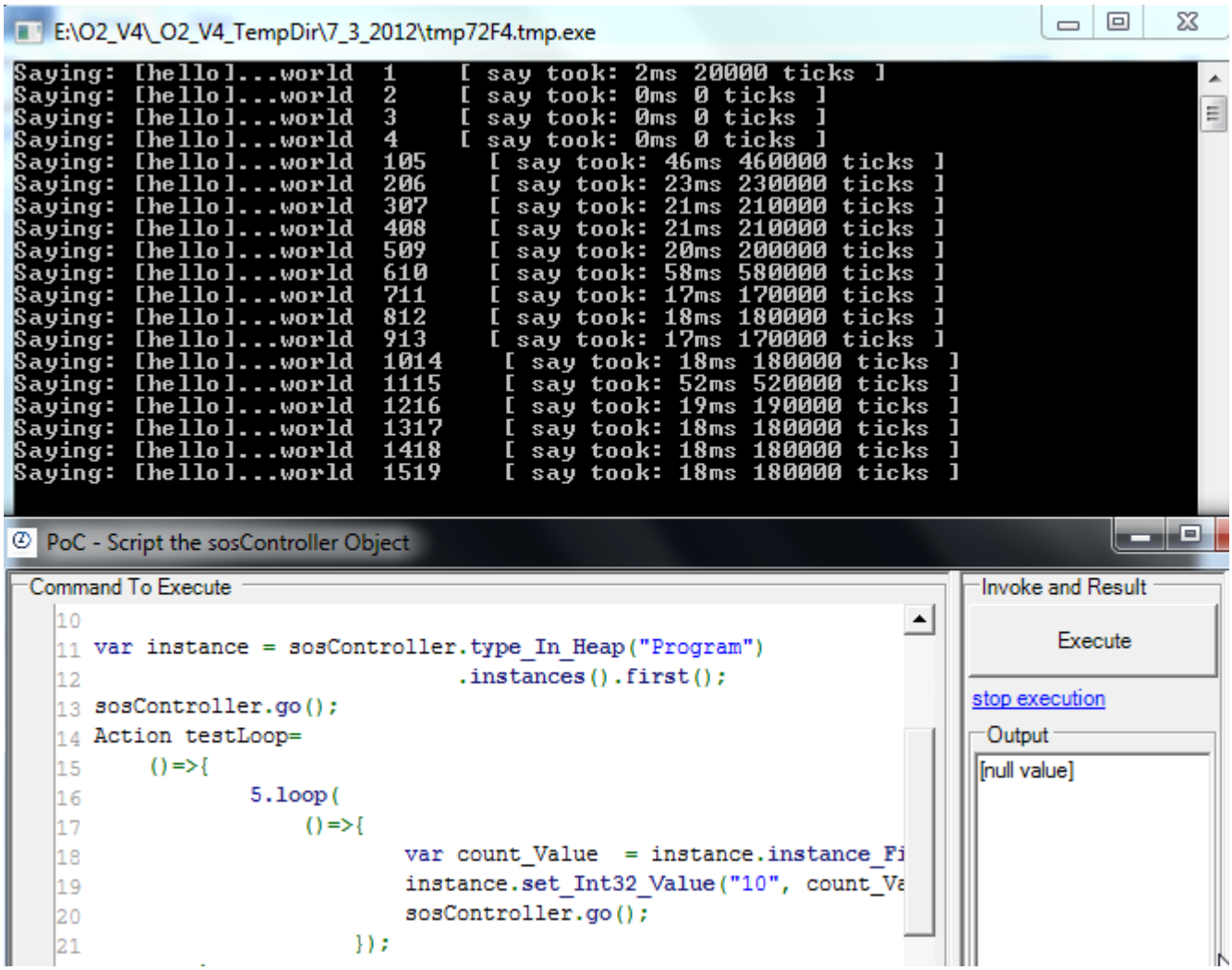
```
//var topPanel = O2Gui.open<Panel>("{name}",700,400);
"exeToDebug".o2Cache(null);
var exeToDebug = "exeToDebug".o2Cache<string>(()=>{
	"Creating temp exe".info();
	var code = @"using System;
	using System.Diagnostics;
	public class Program
	{
		public string here     = ""Here"";
		public string innerMsg = ""[OLA]"";
		public string msg1 = ""[hello]"";
		public int count = 0;
		public int delay = 250;
		public static void Main(String[] args)
		{
			new Program().test();
		}
		public void test()
		{
			string msg2 = ""world"";
			while(true)
			{
				DateTime now = System.DateTime.Now;
				Say(msg1 + ""..."" + msg2);

				var timeSpan = (DateTime.Now - now);
				Console.WriteLine(""    [ say took: "" + timeSpan.TotalMilliseconds +""ms "" + timeSpan.Ticks + "" ticks ] "");
				System.Threading.Thread.Sleep(delay);
			}
		}

		public void Say(String str)
		{
			count++;
			var local = ""Saying: "" + str + ""  "" + count;
			Console.Write(local);
		}
	}";
	return code.createExe(true);
});
```

```
if (exeToDebug.isNull())
	return "compilation error";

var process = exeToDebug.startProcess();
//process.closeInNSeconds(2);
this.sleep(1000);
var controller = new SosController();
var settings = new CdbSettings()
                                {
                                        CdbPath = (clr.x64())
                                        ? @"C:\Program Files\Windows Kits\8.0\Debuggers\x64"
                                        : @"C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86"
                                };

controller.field("settings",settings);

Action attach =
	()=>{
                        controller.AttachToProcess(process.Id.str());
                        controller.setDefaultController();
                        controller.loadSoS();
                        "Attached at the moment:{0}".info(controller.Attached);
                };

var o2Timer = new O2Timer("get data").start();

attach();


var result = controller.ExecuteCommand("!help");

result = controller.ExecuteCommand("!DumpHeap");

var script = controller.script_Me();
script.Code =
@"sosController.setDefaultController();
var result = sosController.type_In_Heap(""Program"")
                                          .instances().first()
                                          .instance_Fields();
return result;
 //return sosController.types_In_Heap();
//return sosController.sos_Help();

//O2Ref:SOS.Net.exe
//O2Ref:SOS.Net.Core.dll
//O2Tag_DontAddExtraO2Files
//O2File:SoSNet_ExtensionMethods.cs
";
//controller.Detach();
script.onClosed(
	()=>{
                        "Parent Form closed, so detaching".info();
                        controller.detach();
                        process.closeInNSeconds(2);
                });
o2Timer.stop();

script.insert_Right(40).add_LogViewer()
        .insert_Above(45,"Execute SoS Command")
        .add_TextBox().fill().set_Text("!help").onEnter((text)=>"{0}".info(text.sosExec()));
return controller;

//O2File:API_ConsoleOut.cs
//O2File:_Extra_methods_Roslyn_API.cs

//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll


//using SOS.Net
//using SOS.Net.Core.Cdb
//using SOS.Net.Core.Cdb.Commands
//O2File:SoSNet_ExtensionMethods.cs
//O2Ref:SosNet\SOS.Net.Core.dll
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
```

Script to run in side the popped-up script editor

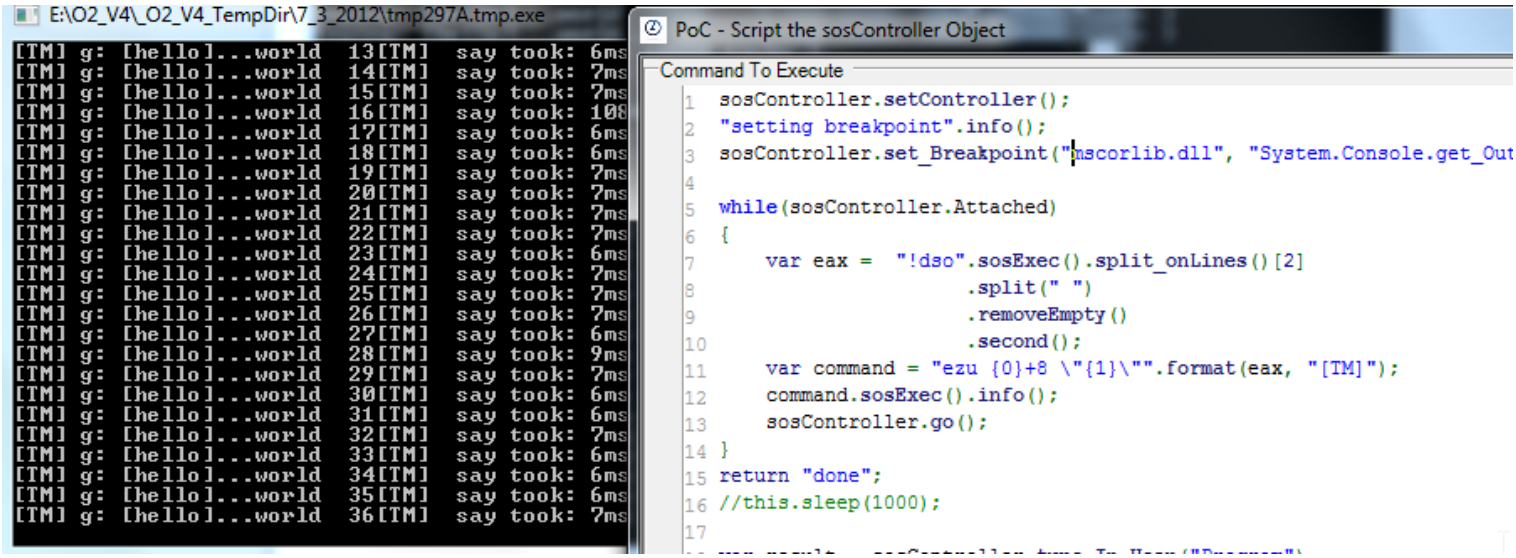
```
var sos = sosController;
var mainAssembly = sosController.assemblies().notInGac().first();
var name = mainAssembly.Name.fileName();
//return sosController.type_In_Heap("Program").Address.toInt().hex();
sosController.setDefaultController();
sosController.set_Breakpoint(name , "Program.Say");
//return sosController.breakpoints();
//@"!bpmd E:\O2_V4\_O2_V4_TempDir\7_3_2012\tmp7F03.tmp.exe Program.Say".sosExec();

var instance = sosController.type_In_Heap("Program")
                                                .instances().first();
sosController.go();
Action testLoop=
       ()=>{
                     5.loop(
                            ()=>{
                                          var count_Value  = instance.instance_Field("count").Value.toInt
();
                                          instance.set_Int32_Value("10", count_Value+100);
                                          sosController.go();
                                   });
              };
instance.set_Int32_Value("14", 1); //set delay to 10;
testLoop();
instance.set_Int32_Value("14", 100); //set delay to 10;
testLoop();
instance.set_Int32_Value("14", 500); //set delay to 100;
testLoop();

//sosController.breakpoints_Clear();
//sosController.detach();

//O2Ref:SOS.Net.exe
//O2Ref:SOS.Net.Core.dll
//O2Tag_DontAddExtraO2Files
//O2File:SoSNet_ExtensionMethods.cs
```

Execution Result (note that the hook time goes from 0 ms to 20ms)



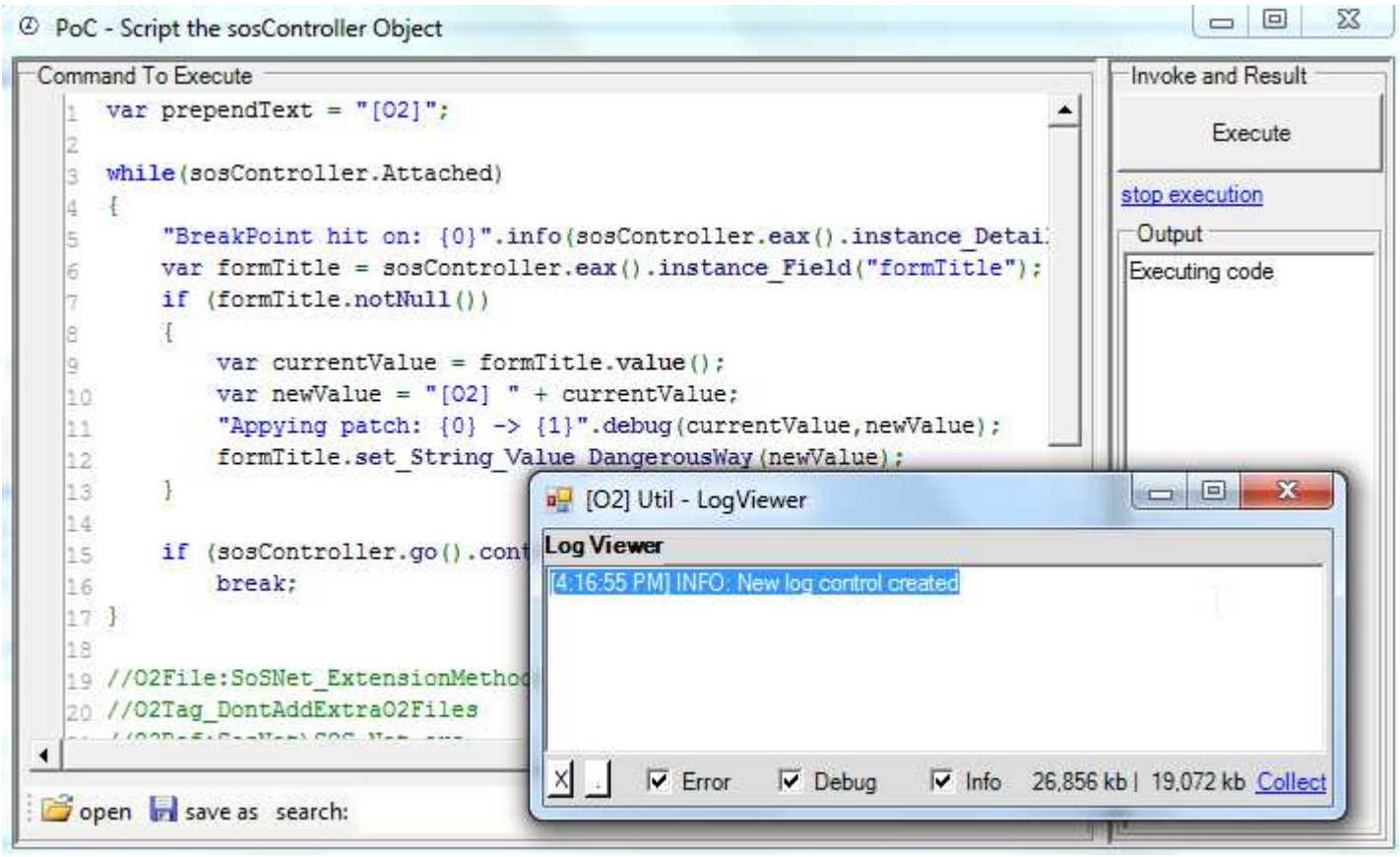
Setting a breakpoint on Console.get_Out (Console.Write line is optimized when in debug mode)

```
sosController.setController();
"setting breakpoint".info();
sosController.set_Breakpoint("mscorlib.dll", "System.Console.get_Out");

while(sosController.Attached)
{
	var eax =  "!dso".sosExec().split_onLines()[2]
	                        .split(" ")
	                        .removeEmpty()
	                        .second();
	var command = "ezu {0}+8 \"{1}\"".format(eax, "[TM]");
	command.sosExec().info();
	sosController.go();
}
```



Script to hook System.Windows.Forms.set_Text

```
var exe = @"file:///E:/O2_V4/_O2_V4_TempDir/7_3_2012/Util - LogViewer [35630]/Util - LogViewer.exe";
var process = exe.startProcess();
var sosController = new SosController();

sosController.attach(process.Id);

sosController.set_Breakpoint("System.Windows.Forms.dll", "System.Windows.Forms.Control.set_Text");
sosController.go();
var script = sosController.script_Me();
var code = @"var prependText = ""[O2]"";

while(sosController.Attached)
{
	""BreakPoint hit on: {0}"".info(sosController.eax().instance_Details().Name.trim());
	var formTitle = sosController.eax().instance_Field(""formTitle"");
	if (formTitle.notNull())
	{
		var currentValue = formTitle.value();
		var newValue = ""[O2]"" + currentValue;
		""Appying patch: {0} -> {1}"".debug(currentValue,newValue);
		formTitle.set_String_Value_DangerousWay(newValue);
	}

	if (sosController.go().contains(""No runnable debuggees error in""))
		break;
}

//O2File:SoSNet_ExtensionMethods.cs
//O2Tag_DontAddExtraO2Files
//O2Ref:SosNet\SOS.Net.exe
//O2Ref:SOS.Net.Core.dll
";
script.Code = code;
return "done";

//using SOS.Net
//using SOS.Net.Core.Cdb

//O2File:SoSNet_ExtensionMethods.cs
//O2File:Scripts_ExtensionMethods.cs
//O2Ref:SosNet\SOS.Net.exe
//O2Tag_DontAddExtraO2Files
//O2Ref:SOS.Net.Core.dll
```



.

.

.

.

.

.

.

.

.

Good References

- SOS.dll (SOS Debugging Extension) http://msdn.microsoft.com/en-us/library/bb190764(v=VS.100).aspx
- sos cheatsheet: http://geekswithblogs.net/.NETonMyMind/archive/2006/03/14/72262.aspx
- Sending an SOS http://codenasarre.wordpress.com/2011/06/22/sending-an-sos/
- Special Command—Editing memory with a, eb, ed, ew, eza, ezu http://blogs.msdn.com/b/debuggingtoolbox/archive/2010/01/06/special-command-editing-memory-with-a-eb-ed-

ew-eza-ezu.aspx

- Updating .NET String in memory with Windbg http://naveensrinivasan.com/2011/06/14/updating-net-string-in-memory-with-windbg/
- How to set breakpoint in windbg for managed code http://asher2003.wordpress.com/2010/08/11/how-to-set-breakpoint-in-windbg-for-managed-code/
- Setting breakpoints in .net code using !bpmd http://blogs.msdn.com/b/tess/archive/2008/04/28/setting-breakpoints-in-net-code-using-bpmd.aspx