

NeoLua

- readme -

Inhalt

1	Introduction.....	3
2	Basic Concepts.....	4
2.1	Values and Types.....	4
2.2	Environment.....	4
2.3	Error Handling	4
2.4	Metatables and Metamethods.....	4
2.5	Garbage Collection	4
2.6	Coroutines	5
3	Language	6
3.1	New Keywords.....	6
3.2	Not implemented	6
3.3	cast	6
3.4	foreach.....	6
4	Application Program Interface	7
4.1	Getting started	7
4.2	Lua script engine.....	8
4.3	LuaChunk	8
4.4	LuaGlobal/LuaTable.....	8
4.5	Debug Information	8
5	The Auxiliary Library.....	9
5.1	Packages	9
5.2	Extensions to LuaGlobal	9
5.3	clr Library (gateway to .net)	10
6	Standard Libraries.....	11
6.1	Basic library	11
6.2	Coroutine library (coroutine)	11
6.3	Package library (package).....	11
6.4	String manipulation (string).....	12
6.5	Table manipulation (table)	12
6.6	Mathematical functions (math)	12
6.7	Bitwise operations (bit32)	13
6.8	Input and output (io)	13
6.9	Operating system facilities (os)	14
6.10	Debug facilities (debug).....	14

1 Introduction

NeoLua is an implementation of the Lua language. Currently the implementation is on the level of Lua 5.2 (<http://www.lua.org/manual/5.2/manual.html>). The target is to follow the reference of the C-Lua implementation and combine this with full .net framework support.

NeoLua is implemented in C# and uses the Dynamic Language Runtime (<https://dlr.codeplex.com>). So it integrates very well in the .net framework.

The reason why I started this project was, that LualInterface is only a bridge to the C-Lua implementation.

Drawbacks of LualInterface:

- You have two memory managers and so you have to marshal every data between these two worlds. That takes time and there are a lot pitfalls to get memory leaks.
- C-Lua interprets the code. The code is not compiled to machine code.

Advantages of LualInterface:

- 100% compatible to the Lua reference.

Drawbacks of NeoLua:

- It is not 100% compatible to Lua, there is a lot of stuff to do. Missing parts I list in this document.
- There is no handling of upvalues possible.
- Debugging is more complicated, because we have to debug machine code.

Advantages of NeoLua:

- NeoLua is based on the DLR. Compiled code. The code is collectable.
- .Net Framework Garbage Collector. Well tested. Fast.

This document has the same structure like the official reference, so it should be easy to compare the two worlds.

2 Basic Concepts

2.1 Values and Types

NeoLua is a dynamically typed language. Variables have no type (internally all declared as an object), only the values have a type. NeoLua supports all CLR types. If you call none dynamic types of the CLR, the values will be automatic converted.

Declaration of Constants:

Lua	Example	CLR	Difference
nil		System.Object	
false		System.Boolean	
true		System.Boolean	
number	1.45	System.Double	
	23	System.Int32	
string	"Test"	System.String	Difference to 8bit string of lua.
function	function() end;	System.Delegate	Can CLR-functions or Lua-Functions. Because they are also compiled to delegates.
userdata			Does not exists. The value has always a CLR-Type. In Lua it is always a reference.
thread			Not implemented.
table	{1}	Neo.IronLua.LuaTable	

2.2 Environment

Lua code is always compiled in delegates that take as first parameter the Environment (Neo.IronLua.LuaGlobal). All global variables are part of this table and easy to exchange between host process and script.

So, it is possible to compile the code one time and execute it multiple times in different environments.

2.3 Error Handling

NeoLua uses the CLR exception handling and introduces two new exception classes. LuaParseException and LuaRuntimeException.

The parse exception is thrown during the parsing, to inform about syntactical errors. The runtime exception is for all runtime errors. Overflow- or DivByZero exception will not catch or convert. The lua error-function creates a LuaRuntimeException.

If you want a stack trace to the exception, you need to compile the script with debug information. With the method LuaExceptionData.GetData you can retrieve for all exceptions a lua stacktrace.

2.4 Metatables and Metamethods

Not implemented.

2.5 Garbage Collection

We fully trust the .net framework garbage collector.

2.6 Coroutines

Not implemented.

But the runtime is thread safe, so you can use the CLR way.

3 Language

The language is nearly the same. I will only show the differences.

3.1 New Keywords

NeoLua introduces two new keywords:

- `cast` for static typing
- `foreach` for the easy access of `IEnumerable`'s

3.2 Not implemented

Hex values are always integers. Numbers like `0x0.1E` will raise a conversion error.

3.3 cast

```
cast ::= 'cast' '(' type ',' expr ')'

type ::= 'byte' | 'sbyte' | 'short' | 'ushort' | 'int' | 'uint' |
'long' | 'ulong' | 'float' | 'double' | 'decimal' | 'datetime' |
'string' | 'object' | 'type' | name { '.' name }
```

Cast enforces a type conversation on a CLR type. This is normally needed to get the correct overloaded CLR method or index.

Example:

```
cast(int, variable);
cast(System.Int32, variable);
```

3.4 foreach

```
foreach ::= 'foreach' loopvar 'in' enumvar 'do' block 'end'
```

Loops types they implement the `System.Collection.IEnumerable` interface.

3.5 do loop

```
doloop ::= 'do' '(' name { ',', name } '=' expr { ',', expr }) block
'end'
```

Do has a using extension. Declare variables within the brackets and they are automatically disposed at the end of the loop.

4 Application Program Interface

4.1 Getting started

Host program:

```
public static class Program
{
    public static void Main(string[] args)
    {
        // create lua script compiler
        using (Lua l = new Lua())
        {
            // create an environment that is associated to the lua scripts
            LuaGlobal g = l.CreateEnvironment();

            // register new functions
            g.RegisterFunction("print", new Action<object[]>(Print));
            g.RegisterFunction("read", new Func<string, string>(Read));

            foreach (string c in args)
            {
                using (LuaChunk chunk = l.CompileChunk(c, true)) // compile the script with debug formations,
                that is needed for a complete stacktrace
                try
                {
                    object[] r = g.DoChunk(chunk); // execute the chunk
                    ...
                }
                catch (TargetInvocationException e)
                {
                    Console.WriteLine("Exception: {0}", e.InnerException.Message);
                    LuaExceptionData d = LuaExceptionData.GetData(e.InnerException); // get stack trace
                    Console.WriteLine("StackTrace: {0}", d.GetStackTrace(0, false));
                }
            }
        }
    } // Main

    private static void Print(object[] texts)
    {
        foreach (object o in texts)
            Console.Write(o);
        Console.WriteLine();
    } // proc Print

    private static string Read(string sLabel)
    {
        Console.Write(sLabel);
        Console.Write(": ");
        return Console.ReadLine();
    } // func Read
} // class Program
```

Lua-Scrip:

```
local a, b = tonumber(read("a")), tonumber(read("b"));

function PrintResult(o, op)
    print(o .. ' = ' .. a .. op .. b);
end;

PrintResult(a + b, " + ");
PrintResult(a - b, " - ");
PrintResult(a * b, " * ");
PrintResult(a / b, " / ");
```

4.2 Lua script engine

The lua class holds all together, that is needed for the compile and the runtime process. Normally you need only one instance per application. But sometimes it is useful to have more than one e.g. multithread scenarios.

Clear	Removes all compiled chunks and clears the script compile cache.
CompileChunk	Compiles lua code to a LuaChunk. A chunk can be compiled with debug information and arguments.
CreateEnvironment	Creates a new LuaTable that is associated to the lua script engine (see more under LuaGlobal)

4.3 LuaChunk

Represents compiled lua code.

Dispose	Removes this chunk from the lua script engine and is assigned chunks.
ChunkName	
IsCompiled	
HasDebugInfo	
IsEmpty	

4.4 LuaGlobal/LuaTable

LuaTable is the implementation of the table type for lua. It is a dynamic type and holds different values. It implements enumeration interface and property changing.

RegisterFunction	Registers a function to the table. It is equal to <code>table[name] = function;</code>
this[index]	Get/Set set an index or member.
Length	Length of the array.

LuaGlobal is an extension from the LuaTable to provide the Basic Functions and Packages.

RegisterPackage	see Packages
DoChunk	Executes a chunk on this environment.

4.5 Debug Information

It is possible to add debug information to the compiled script. But the handling of the DLR is different.

If you compile a script with debug information the lua script engine creates a dynamic assembly, and for each compiled chunk a type. The compile time is nearly ten times slower, and memory usage increases (nearly four times). It does not affect the runtime.

Without debug information you get no stacktrace for exceptions, but it compiles really fast and has less memory usage.

Retrieve StackTrace:

```
LuaExceptionData led = LuaExceptionData.GetData(e);
```

```
Console.WriteLine(led.StackTrace);
```

5 The Auxilary Library

5.1 Packages

It is possible to add more packages to the environment. There are two ways to add packages. Static packages and dynamic packages.

Static Packages are basically static classes. The static members will be available to lua.

Restrictions:

- No overload allowed.
- No Instance members allowed.

Example declaration:

```
public static class Test
{
    public static int add(int a, int b)
    {
        return a + b;
    }
}
```

Example register:

```
g.RegisterPackage("test", typeof(Test));
```

Example usage:

```
return test.add(1, 2);
```

For dynamic packages you have to implement `IDynamicMetaObjectProvider`. More information can be find under <http://dlr.codeplex.com/documentation> ([library-authors-introduction.pdf](#))

5.2 Extensions to LuaGlobal

The easiest way to add new functions or members to the lua environment or table is to add a member. It is also the only way to override the standard implementation of the function. If you want the standard back, just delete the member by setting it to nil/null.

Example:

```
LuaTable t = new LuaTable();
t["var"] = 2;
t["foo"] = new Action(() => {});
```

or

```
dynamic t = new LuaTable();
t.var = 2;
t.foo = new Action(() => {});
```

It is also possible to create your own lua environment. For this you have to subclass the standard environment `LuaGlobal`.

Extent Basic Functions:

- Add a private function to your environment class and mark it with `LuaFunctionAttribute`.
- Example:

```
[LuaFunction("rawset")]
private LuaTable LuaRawSet(LuaTable t, object index, object value)
```

The third way to extend the environment is to change the dynamic interface. For this you have to override the `GetMetaObject`-Method and you **must** inherit the `IDynamicMetaObjectProvider` from `LuaCoreMetaObject`.

5.3 clr Library (gateway to .net)

clr is the interface to the .net framework common language runtime. You can access .net types and namespaces.

Example (create a `StringBuilder`-class):

```
local StringBuilder = clr.System.StringBuilder;
-- invokes the constructor to create an instance
local sb = StringBuilder:ctor();
```

Example (get newline):

```
return clr.System.Environment.NewLine;
```

Example (String Concat):

```
function a()
    return 'Hallo ', 'Welt', '!';
end;
local sys = clr.System;
return sys.String:Concat(a());
```

6 Standard Libraries

This sections gives an overview of the implementation state of the lua system libraries.

States:

- **done**: Is compatible to the lua reference.
- **noco**: Not full compatible to the lua reference.
- **noti**: Not implemented.

6.1 Basic library

Function	State	Remark
<code>assert</code>	done	
<code>collectgarbage</code>	done	Only the parameter "count" and "collect" are supported.
<code>dofile</code>	done	Redirects to DoChunk.
<code>error</code>	done	Throws a LuaRuntimeException.
<code>_G</code>	done	
<code>getmetatable</code>	noco	
<code>ipairs</code>	done	
<code>load</code>	noti	
<code>loadfile</code>	noti	
<code>next</code>	noti	
<code>pairs</code>	done	
<code>pcall</code>	done	
<code>print</code>	done	Prints on the debug output.
<code>rawequal</code>	done	
<code>rawget</code>	done	
<code>rawlen</code>	done	
<code>rawset</code>	done	
<code>select</code>	done	
<code>setmetatable</code>	noco	
<code>tonumber</code>	done	
<code>tostring</code>	done	
<code>type</code>	done	"type" is extended with a second boolean parameter, that replaces "userdata" with the clr-type-name: type(obj, true);
<code>_VERSION</code>	done	Cannot be replaced.
<code>xpcall</code>	noco	

6.2 Coroutine library (coroutine)

Function	State	Remark
<code>create</code>	noti	
<code>resume</code>	noti	
<code>running</code>	noti	
<code>status</code>	noti	
<code>wrap</code>	noti	
<code>yield</code>	noti	

6.3 Package library (package)

Function	State	Remark

require	noti
config	noti
cpath	noti
loaded	noti
loadlib	noti
path	noti
preload	noti
searchers	noti
searchpath	noti

6.4 String manipulation (string)

Function	State	Remark
byte	done	
char	done	
dump	noti	
find	noco	.net regex syntax with an exchange of the escape symbol % is \.
format	done	
gmatch	noco	
gsub	noti	
len	done	
lower	done	
match	noco	
rep	done	
reverse	done	
sub	done	
upper	done	

6.5 Table manipulation (table)

Function	State	Remark
conat	done	
insert	done	
pack	done	
remove	done	
sort	done	
unpack	done	

6.6 Mathematical functions (math)

Function	State	Remark
abs	done	
acos	done	
asin	done	
atan	done	
atan2	done	
ceil	done	
cos	done	
cosh	done	
deg	done	

e	done
exp	done
floar	done
fmod	done
frexp	noti
huge	noti
ldexp	noti
log	done
max	done
min	done
modf	done
pi	done
pow	done
rad	done
random	done
randomseed	done
sin	done
sinh	done
sqrt	done
tan	done
tanh	done

6.7 Bitwise operations (bit32)

Function	State	Remark
arshift	done	
band	done	
bnot	done	
bor	done	
btest	done	
bxor	done	
extract	done	
replace	done	
lrotate	done	
lshift	done	
rrotate	done	
rshift	done	

6.8 Input and output (io)

Function	State	Remark
close	noti	
flush	noti	
input	noti	
lines	noti	
open	noti	
output	noti	
popen	noti	
read	noti	
tmpfile	noti	

type	noti
write	noti
:close	noti
:flush	noti
:lines	noti
:read	noti
:seek	noti
:setvbuf	noti
:write	noti

6.9 Operating system facilities (os)

Function	State	Remark
clock	noco	Returns Environment.TickCount.
date	noti	
difftime	noti	
execute	noti	
exit	noti	
getenv	done	
remove	noti	
rename	noti	
setlocale	noti	
time	noti	
tmpname	noti	

6.10 Debug facilities (debug)

Function	State	Remark
debug	noti	
gethook	noti	
getinfo	noti	
getlocal	noti	
getmetatable	noti	
getregistry	noti	
getupvalue	noti	
setuservalue	noti	
traceback	noti	
upvalueid	noti	
upvaluejoin	noti	