

Open Source Scripting in Second Life: Get Ready for LSL-Mono

by Gigs Taggart, Correspondent



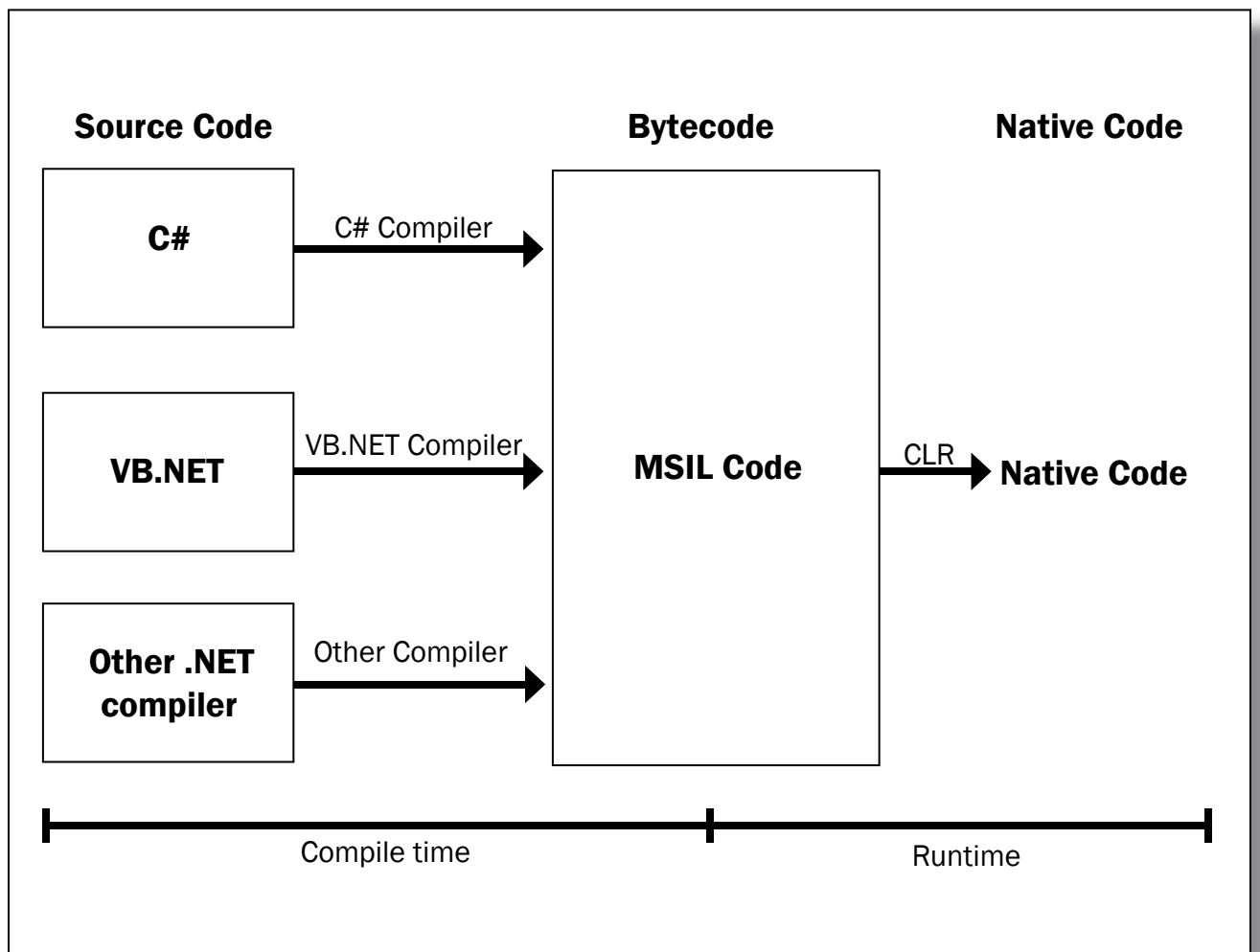
Gigs Taggart (Jason Giglio) is an active member of the Second Life community as a scripter and contributor to the OpenSL project, land developer, and proprietor of GT HQ Store and Arcade.

Several years back, Microsoft introduced .NET. One of the benefits of .NET is the [Common Language Runtime \(CLR\)](#). This allows many different languages to be compiled into a universal language called Intermediate Language (IL). This IL, in turn, runs on any platform where the CLR is implemented. Microsoft, however, only provided the CLR for MS Windows. The Mono project is an open source project to port the .NET CLR to other platforms, such as Linux and Mac OS X.

When Linden Lab looked to overhaul the aging Linden Scripting Language (LSL), Mono emerged as a potential solution. Linden Lab could simply write a compiler to translate LSL into IL, and then run it on the Linux implementation of the CLR provided by Mono. This project has progressed and is now available on the [beta grid](#).

A note on terminology: you will sometimes hear people use the term "Virtual Machine" or VM. The CLR is very similar to a Virtual Machine, however it runs the higher level Object Oriented IL code, rather than machine code. This aside, you may hear people speak of the "LSL2 VM" vs the "Mono VM". While this is slightly inaccurate, it's close enough.

So what does this all mean? In the short run, it means only that LSL scripts compiled to the new Mono CLR target will run up to 200 times faster. Eventually, Linden Lab may allow other languages to be used, such as C#.



Common Language Runtime (CLR)

Scripts That Will Never Work Right under Mono

The original plan was to migrate all LSL2 VM scripts to Mono. This plan is flawed, however. There are certain programming techniques, bugs, and problems that were tolerated by the LSL2 VM, but will never work correctly under the Mono CLR. This is in addition to any potential bugs in the Mono translation, that is, even if LSL-Mono were bug free, these scripts will still break. Be aware of these caveats when writing your scripts, and they will be much more likely to smoothly transition to Mono.

1. Visual Effects That Rely On Slow Execution

Here's a little snippet that causes part of a linkset to change to rainbow colors in a chasing effect. This will work fine under the old LSL2 VM:

```

for (x=0; x < gNumButtons; x++)
{
    llSetLinkColor(x + gButtonOffset, get_color(x), ALL_SIDES);
}
  
```

It will break entirely under Mono, going way too fast. The solution here is to add `llSleep(0.1)` in the loop. This will allow the effect to look good under mono. Note that `llSetLinkColor` has no built-in delay. An alternate solution would be to use the `timer()` event.

2. Link Message Race Condition

LSL2 was slow. Really slow. This low speed can hide race conditions that are in your code. Link Messages are asynchronous, which makes them particularly prone to race conditions

Here we have a race condition. Depending on how slow the code in the middle of the first script is, this script will either report the correct name, or report an incorrect name:

Correct:

one: Name is one
two: Name is two

Incorrect:

two: Name is one
two: Name is two

This is obviously a contrived example, but situations similar to this one can crop up in code. Never rely on the delay created by the execution of some code to allow for time for other processing to happen, and this won't affect you.

Conclusion

These are only a few caveats. There are others such as the Gray Goo Fence (self-replicating objects), looping based on user input events, and games that become too fast to play. These problems are reminiscent of the upgrade from 8088 IBM PCs to 286 AT clones. These clones included a turbo button, not to speed up the system, but to allow it to be slowed down to 8088 levels for compatibility with older programs that relied on the 4.77Mhz speed of the IBM PC.

An early plan was to recompile all the LSL2 VM scripts to the Mono CLR, on the fly. This operation would copy the state of the running script so that no data would be lost. This plan is also flawed – because of the above problems. It remains to be seen if Linden Lab will offer a “Compatibility Mode” for Mono that executes more slowly. This would allow LSL2 VM scripts to be ported, even if they have these sorts of problems.

All of these problems are a result of bad coding practices and assumptions made on the part of the programmer. They are, however, still common. Think to yourself, “How would this program run if the code were running 500 times faster?” Ensure that your code does not make assumptions about execution speed, and you'll be ready for LSL-Mono. ☞

Script 1:

```
default
{
    touch_start(integer total_number)
    {
        llSetObjectName("one");
        llMessageLinked(LINK_SET, 0, "one", NULL_KEY);

        integer x;
        for (x=1; x < 100; x++) //this emulates some code
                                //being in the middle

            1+1;

        llSetObjectName("two");
        llMessageLinked(LINK_SET, 0, "two", NULL_KEY);
    }
}
```

Script 2:

```
default
{
    link_message(integer sender, integer num, string str, key id)
    {
        llSay(0, "Name is " + str);
    }
}
```

3. Event Queue Overrun

Event queues in LSL are 64 elements deep. Because LSL running under Mono runs so much faster, it's entirely possible that a script generating events for another script will feed it too quickly and the event queue will overrun, causing messages to be dropped:

Script 1:

```
default
{
    touch_start(integer total_number)
    {
        integer x;
        integer y;
        for (x=0; x<100; x++)
        {
            for (y=0;y<100;y++) //this emulates some code
                                // running here

                1+1;

            llMessageLinked(LINK_SET, x, "", NULL_KEY);
        }
    }
}
```

Script 2:

```
default
{
    link_message(integer sender, integer num, string str, key id)
    {
        llSay(0, (string)num);
    }
}
```

Under LSL2 VM this outputs every number correctly. Under Mono this outputs 0-66, 74, and no more. The event queue was overrun.