Modula-3 for the Web

Klaus Preschern
https://preschern.org

September 6, 2026

Abstract

This report describes experimental changes made to the Modula-3 compiler and runtime system to compile Modula-3 programs with Emscripten. Configurations to emit 32-bit and 64-bit WebAssembly are described and options available for the implementation of preemtive threads and the suspension of threads during garbage collection are explored. With the changes described it is possible to execute Modula-3 programs with Node.js and in web browsers. By generating WebAssembly Modula-3 gets the ability to compile once and run everywhere.

1 Overview

The Modula-3 compiler does use configuration files for each supported target. Therefore two new targets, **Wasm32_EM** and **Wasm64_EM**, have been created for Emscripten.

1. Common configuration settings for Emscripten
   cm3/m3-sys/cminstall/src/config/Emscripten.common (does use Unix.common)
2. **Wasm32_EM**: 32-bit with user threads
   cm3/m3-sys/cminstall/src/config/Wasm32_EM
3. **Wasm64_EM**: 64-bit with Pthreads
   cm3/m3-sys/cminstall/src/config/Wasm64_EM

A bug (from the WebAssembly point of view) has been discovered and removed in the Modula-3 code generator which has caused the **function signature mismatch** runtime error in the generated WebAssembly code. WebAssembly does check procedure

signatures at runtime and therefore the Modula-3 compiler must generate matching function signatures.

The Wasm32_EM target does use Emscripten fibers to implement user threads and the Wasm64_EM target is based on the Pthread implementation of Emscripten, but the suspension of Pthreads remains an open problem.

2 Thread support

Currently only Wasm32_EM does support the suspension of threads for garbage collection. With Pthreads (Wasm64_EM) the suspension for garbage collection does currently not work, because Emscripten does neither support the **sigsuspend** system call nor direct suspension of threads. The Modula-3 thread test program[1] does work with Wasm64_EM and can be executed with **node threadtest.js -tests tryexcept,lock @M3nogc** (no garbage collection), because these tests do not allocate a lot of memory.

Browsers have implemented security measures to mitigate the Spectre and Meltdown vulnerabilities. Pthreads (**Wasm64_EM**) require the (gated) SharedArrayBuffer to share memory between WebAssembly threads. Browsers that have implemented and enabled SharedArrayBuffer, are gating it behind Cross Origin Opener Policy (COOP) and Cross Origin Embedder Policy (COEP) headers. Pthreads do not work in such browsers unless these headers are correctly set. For more information see COOP-COEP Headers.

User threads (**Wasm32_EM**), based on Emscripten fibers, do not need the SharedArrayBuffer and hence do not require these policy headers.

Whether user threads are used or Pthreads is selected with the variable **NOPTHREAD** in the configuration file. For Wasm32_EM **NOPTHREAD = 1**.

The user threads currently require compiler support. The procedure **em_proc_enter()**[2] is called whenever a Modula-3 procedure in entered and does a **nanosleep** to keep the virtual timer alive, which is used for preemption. It may be required to extend this kind of support to include Modula-3 (potentially endless) loops, with no procedure calls.

3 Description of function signature mismatch

Here is an example which does cause the function signature mismatch runtime error.

```
1 UNSAFE MODULE RTLinker EXPORTS RTLinker, RTModule;
```

[1] Located in cm3/m3core/tests/thread.

[2] Defined in cm3/m3-libs/m3core/src/thread/Emscripten/em_ucontext.c

```
2   PROCEDURE AddUnit (b: RT0.Binder) =
3     VAR m: RT0.ModulePtr;
4   BEGIN
5     IF (b = NIL) THEN RETURN END;
6     m := b(0); (* line 122 in RTLinker.m3 *)
7     IF (m = NIL) THEN RETURN END;
8     AddUnitI(m);
9   END AddUnit;
10 BEGIN
11 END RTLinker.
```

The Modula-3 code generator[3] does generate a procedure call with signature **(int,void*)** because the passed procedure could be a nested procedure with a static link. Nested procedures are passed with a closure, but for top level procedures the address of the procedure is passed (see the call of **RTLinker__AddUnit** in the last line of the listing below). The signature of the passed top level procedure **RT0__ModulePtr RTLinker_I3(INTEGER)** has no static link. The call of this procedure does cause a signature mismatch runtime error with WebAssembly code.

```
1   void __cdecl RTLinker__AddUnit( RT0__Binder b_L_82 )
2   {
3     ...
4     /* Example of a call which does generate the runtime error (the 2nd
          void* parameter RTLinker_m_150_L_151 is a static link): */
5
6     (*(ADDRESS*)(&RTLinker_m_154_L_155))=
7       (ADDRESS)(((((ADDRESS (__cdecl*)(int,void*))b_L_82)(0,
            RTLinker_m_150_L_151))));
8     ...
9   }
10
11  /* The signature of the passed top level procedure is just INTEGER,
       without a 2nd void* parameter: */
12
13  RT0__ModulePtr __cdecl RTLinker_I3( INTEGER mode_L_14 );
14
15  /* RTLinker__AddUnit is called like this (passing procedure
       RTLinker_I3 as parameter): */
16
17  RTLinker__AddUnit(( RT0__Binder )(((ADDRESS)(RTLinker_I3)) ));
```

This problem cannot be fixed at compile time. The decision which call to take must be made at runtime and the compiler must generate both variants of such a call. One with static link and one without it (see below).

```
1  #line 118 "../src/runtime/common/RTLinker.m3"
2  void __cdecl RTLinker__AddUnit( RT0__Binder b_L_82 )
```

[3]There are different backends available for Modula-3. Only the C/C++ backend has been changed.

```
3   {
4     bool static_link_initialized = false;
5     ADDRESS RTLinker_m_148_L_149={0}; //loadBeforeStore
6     ADDRESS RTLinker_m_150_L_151;     // uninitialized static link

7     ADDRESS RTLinker_m_152_L_153={0}; //initAll
8     ADDRESS RTLinker_m_154_L_155={0}; //loadBeforeStore
9     ...
10    #line 122 "../src/runtime/common/RTLinker.m3"
11    (*(ADDRESS*)(&RTLinker_m_148_L_149))=
12        (ADDRESS)(((ADDRESS)(b_L_82))); // procedure address
13
14    if (pop_static_link()) { /* Nested procedure with closure? */
15      (*(ADDRESS*)(&RTLinker_m_150_L_151))=
16        (ADDRESS)(((*((ADDRESS*)(8+((ADDRESS)(RTLinker_m_148_L_149)))))
              )); // static link
17      static_link_initialized = true; // static link popped

18      (*(ADDRESS*)(&RTLinker_m_148_L_149))=
19        (ADDRESS)(((*((ADDRESS*)(4+((ADDRESS)(RTLinker_m_148_L_149)))))
              )); // procedure address
20    }
21    (*(ADDRESS*)(&RTLinker_m_152_L_153))=(ADDRESS)(((ADDRESS)(
          RTLinker_m_148_L_149)));
22
23    // use a lambda function with both variants
24    (*(ADDRESS*)(&RTLinker_m_154_L_155)) =
25      (ADDRESS)(((ADDRESS)( ([&]() {
26    const auto result =
27      (static_link_initialized ?
28        ((ADDRESS (__cdecl*)(int,void*)RTLinker_m_152_L_153)(0,
              RTLinker_m_150_L_151) /* Call with static link. */
29        :
30        ((ADDRESS (__cdecl* )(int)RTLinker_m_152_L_153)(0)); /* Call
              without static link. */
31      static_link_initialized = false;
32      return result;
33    }) ())));
34    ...
35  }
```

The solution to fix this problem in the Modula-3 code generator is provisional and a general solution for all targets should be developed, which will likely require changes in the front end (in example in UserProc.m3) of the Modula-3 compiler. The static link should be initialized and then used to select the procedure call with the matching signature.

4 List of files changed

The first 5 changes were required to remove a compile error (with target AMD64_NT and Visual Studio C++) caused by the parameter of the function **cabs()**. These changes are unrelated to the changes required to compile and run Modula-3 programs with Emscripten.

1. **cm3/m3-libs/libm3/src/arith/WIN32/m3makefile:**
   Fix compile error with Math.i3 on AMD64_NT with Visual Studio C++
2. **cm3/m3-libs/libm3/src/arith/WIN32/Math.i3:**
   <*EXTERNAL "_cabs" *> implemented in Modula-3
3. **cm3/m3-libs/libm3/src/arith/WIN32/Math.m3:**
   PROCEDURE cabs (z: Complex): LONGREAL calls MathWin32C.cabs(...)
4. **cm3/m3-libs/libm3/src/arith/WIN32/MathWin32C.c:**
   Implementation of double MathWin32C__cabs(double x, double y)
5. **cm3/m3-libs/libm3/src/arith/WIN32/MathWin32C.i3:**
   Interface with <*EXTERNAL "MathWin32C__cabs"*> PROCEDURE cabs (x, y: LONGREAL): LONGREAL;
6. **cm3/m3-libs/libm3/src/uid/POSIX/MachineIDPosixC.c:**
   #if defined (__EMSCRIPTEN__) added to #define HAS_SIOCGIFCONF
7. **cm3/m3-libs/m3core/src/runtime/POSIX/RTSignalC.c:**
   #if defined (__EMSCRIPTEN__) added to #define GetPC(x) 0
8. **cm3/m3-libs/m3core/src/thread/m3makefile:**
   if equal (TARGET_OS, "EMSCRIPTEN") then include_dir ("Emscripten")
9. **cm3/m3-libs/m3core/src/thread/Emscripten/getdtablesize.c:**
   Implementation of int getdtablesize(void)
10. **cm3/m3-libs/m3core/src/thread/Emscripten/m3makefile:**
    New m3makefile
11. **cm3/m3-libs/m3core/src/thread/Emscripten/sigsuspend.c:**
    Empty implementation of int sigsuspend(const sigset_t *mask)
12. **cm3/m3-libs/m3core/src/thread/Emscripten/em_ucontext.c:**
    Emscripten fiber ucontext and em_proc_enter()
13. **cm3/m3-libs/m3core/src/thread/POSIX/ThreadPosixC.c:**
    Changes for Emscripten user threads

14. **cm3/m3-libs/m3core/src/thread/PTHREAD/ThreadPThreadC.c:**
    int SIG_SUSPEND = SIGRTMAX;

15. **cm3/m3-libs/m3core/src/thread/Common/ThreadInternal.c**
    Parameters of select system call adapted for Emscripten

16. **cm3/m3-libs/m3core/src/unix/Common/UnixC.c:**
    Use emscripten_force_exit(i);

17. **cm3/m3-sys/cminstall/src/config/Emscripten.common:**
    New common configuration file for Emscripten

18. **cm3/m3-sys/cminstall/src/config/Wasm32_EM:**
    New configuration file Wasm32_EM for Emscripten

19. **cm3/m3-sys/cminstall/src/config/Wasm64_EM:**
    New configuration file Wasm64_EM for Emscripten

20. **cm3/m3-sys/m3back/src/M3C.m3:**
    Fix signature mismatch and other things

21. **cm3/scripts/version:**
    New CM3VERSION d5.12.0

22. **cm3/scripts/version.quake:**
    New CM3VERSION d5.12.0

23. **cm3/scripts/python3/upgrade-min.py:**
    New script for upgrading without installing for creating cm3.wasm

5 Experimental Setup

Modula-3 release:	d5.12.0 (with CM3 compiler version d5.11.9)
Modula-3 targets used:	AMD64_NT, Wasm32_EM, Wasm64_EM
Emscripten version:	6.0.6
Node.js version:	24.19.0
Python version:	3.14.7
Visual Studio:	Community Edition 2022
Browsers:	Chrome, Edge, Firefox
OS:	Windows

1. Download cm3-boot-AMD64_NT-d5.12.0.7z from github[4] and build cm3.exe.

[4]Location: https://github.com/modula3/cm3/releases/tag/d5.12.0

To avoid ,,Run-Time Check Failure #3 - The variable 'RTLinker_m_151_L_152' is being used without being initialized." because of uninitialized variables, disable these checks. These checks are usually enabled for DEBUG builds, but disabled for RELEASE builds with Visual Studio C++.

cm3 -version of this compiler is ,,Critical Mass Modula-3 version d5.11.9".

2. Create a /m3[5] directory (not necessarily at the top level).

3. Create directory /m3/bin and copy cm3.exe (built with step 1.) to it.

   Make sure /m3/bin is in your PATH (check with cm3 -version).

4. Download **cm3-d5.12.0.zip** (source code) from github and unzip it in /m3.

5. Create two symbolic links ,,cm3" and ,,pkg" to cm3-d5.12.0.

   /m3>mklink /D cm3 cm3-d5.12.0
   /m3>mklink /D pkg cm3-d5.12.0

6. Unzip M3_ForTheWeb.zip in /m3.

   This does patch the cm3 distribution with the changes required for Emscripten.

7. Create symbolic links with /m3/cm3/make_pkg_links.bat.

   Run this batch file in an Administrative Window in directory /m3/cm3 and close the command window afterwards.

   /m3/cm3>make_pkg_links.bat

8. Open an ,,x64 Native Tools Command Prompt for VS 2022".

   cd /m3/cm3/scripts/python3

   Make sure python is in your PATH (check with python –version). The target used is AMD64_NT with Visual C++.

   /m3/cm3/scripts/python3>python do-cm3-min.py noclean

   This command may fail with the following error:

```
1  Traceback (most recent call last):
2    File "\m3\cm3\scripts\python3\do-cm3-min.py", line 8, in <module>
3      SetupEnvironment()
4      ~~~~~~~~~~~~~~~~~^^
5    File "\m3\cm3\scripts\python3\pylib.py", line 1629, in
       SetupEnvironment
6      SetVisualCPlusPlus2015OrNewer()
7      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^
8    File "\m3\cm3\scripts\python3\pylib.py", line 1596, in
       SetVisualCPlusPlus2015OrNewer
9      cver = GetVisualCPlusPlusVersion()
```

[5] '/' is used as path separator instead of '\' for convenience

```
10    File "\m3\cm3\scripts\python3\pylib.py", line 1557, in
          GetVisualCPlusPlusVersion
11      a = os.popen("cl 2>&1 >nul").read().lower()
12          ~~~~~~~~~~~~~~~~~~~~~~~~~~~^^
13    File "...\Python\Python314\Lib\encodings\cp1252.py", line 23, in
          decode
14      return codecs.charmap_decode(input,self.errors,decoding_table)
            [0]
15             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
16  UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in
      position 62: character maps to <undefined>
```

You can fix the error above by changing line 1557 in pylib.py:

```
1  def GetVisualCPlusPlusVersion():
2      #a = os.popen("cl 2>&1 >nul").read().lower()
3      return "2022"
```

Afterwards run ,,python do-cm3-min.py noclean" again.

If this fails with error LNK2001: Unresolved symbol "m3_jmpbuf_size" then start the python script again (without changing something). This should then finally display ,,do-cm3-min.py: Success.".

9. Rebuild the Modula-3 compiler.

   /m3/cm3/scripts/python3>python upgrade.py noclean

   If this fails with error LNK2001 run ,,python upgrade.py noclean" again. The script should finally end with ,,upgrade.py: Success.".

10. Edit /m3/bin/cm3.cfg and add ,,**ROOT = INSTALL_ROOT**" as last line.

/m3/cm3/scripts/python3>cm3 –version

This should display ,,Critical Mass Modula-3 version d5.12.0" (this is currently an unofficial CM3 version with the WebAssembly upgrade). Congratulations - you have built the new Modula-3 compiler which does generate C/C++ code, suitable to be translated to WebAssembly with Emscripten and executed with Node.js.

6 Building the library

We are building below for Wasm32_EM. It does work in the same way for Wasm64_EM by setting the environment variable CM3_TARGET appropriately.

Open a normal command prompt window and setup the Emscripten EMSDK environment. The ,,x64 Native Tools Command Prompt" is not required. Run ,,emsdk/emsdk_env.bat" and make sure **emcc** (the Emscripten compiler) is found (check

with emcc –version).

cd /m3/cm3/scripts/python3
/m3/cm3/scripts/python3>set CM3_TARGET=Wasm32_EM
/m3/cm3/scripts/python3>python do-cm3-min.py noclean

This should end with ,,do-cm3-min.py: Success.".

7 Compiling and executing programs

Lets build ,,/m3/cm3/libm3/tests/random/src/RandomTest.m3".

cd /m3/cm3/libm3/tests/random/src
/m3/cm3/libm3/tests/random/src>cm3 -DTARGET=Wasm32_EM -commands
/m3/cm3/libm3/tests/random/src>cd ../Wasm32_EM
/m3/cm3/libm3/tests/random/Wasm32_EM>node RandomTest.js -all

Lets try the thread test ,,/m3/cm3/m3core/tests/thread/src/Main.m3". Because it does depend on ,,parseparams" we must build this first.

cd /m3/cm3/m3-libs/parseparams/src
/m3/cm3/m3-libs/parseparams/src>cm3 -DTARGET=Wasm32_EM -commands
/m3/cm3/m3-libs/parseparams/src>cd /m3/cm3/m3core/tests/thread/src
/m3/cm3/m3core/tests/thread/src>cm3 -DTARGET=Wasm32_EM -commands
/m3/cm3/m3core/tests/thread/src>cd ../Wasm32_EM
/m3/cm3/m3core/tests/thread/src>node threadtest.js -tests creat,lock

```
1  Writing file...done
2  Creating creat threads...done
3  Creating lock threads...done
4  running...printing oldest/median age/newest
5  .........laziest thread is 1/0/0 (tests: creat 0/0/0 lock 1/0/0)
6  .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
7  .........laziest thread is 1/0/0 (tests: creat 0/0/0 lock 1/0/0)
8  .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
9  .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
10 .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
11 .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
12 .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
13 .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
14 .........laziest thread is 0/0/0 (tests: creat 0/0/0 lock 0/0/0)
15 All tests complete.  Congratulations.
```

If you want compile for Wasm64_EM (after building the library for Wasm64_EM by setting CM3_TARGET=Wasm64_EM) use ,,**cm3 -DTARGET=Wasm64_EM -commands**" to compile single programs. Note that garbage collection does currently not work with Wasm64_EM. It is possible to run the thread test with ,,**node threadtest.js -tests**

tryexcept,lock @M3nogc" (no garbage collection) with Wasm64_EM, because these two tests do not allocate a lot of memory.

8 Conclusion

The first steps have been made to create a Modula-3 compiler toolchain for the web platform. The changes made are experimental to get it working. They should be checked and improved, to make Modula-3 to one of the languages supporting WebAssembly. The Modula-3 compiler and runtime system have again proven to be very portable and with WebAssembly code generation, programs can be compiled once and executed everywhere.

Currently only user threads have garbage collection and have the advantage of not requiring the SharedArrayBuffer. The support of garbage collection for Pthreads (WebAssembly threads) is an open problem and may require the implementation of the sigsuspend system call or direct suspension of threads in Emscripten, with the current Modula-3 runtime system. The solution which is used to keep the virtual timer running for the user threads is provisional and a more sophisticated self adjusting solution is required to get a stable timer signal.

The examples compiled with Wasm32_EM can be executed in current web browsers (Chrome, Edge, Firefox) on the Modula-3 Testpage.