NAME

mono – Mono's ECMA-CLI native code generator (Just-in-Time and Ahead-of-Time)

SYNOPSIS

mono [options] file [arguments...]

mono-sgen [options] file [arguments...]

DESCRIPTION

mono is a runtime implementation of the ECMA Common Language Infrastructure. This can be used to run ECMA and .NET applications.

The runtime loads the specified *file* and optionally passes the *arguments* to it. The *file* is an ECMA assembly. They typically have a .exe or .dll extension.

These executables can reference additional functionality in the form of assembly references. By default those assembly references are resolved as follows: the **mscorlib.dll** is resolved from the system profile that is configured by Mono, and other assemblies are loaded from the Global Assembly Cache (GAC).

The runtime contains a native code generator that transforms the Common Intermediate Language into native code.

The code generator can operate in two modes: Just-in-time compilation (JIT) or Ahead-of-time compilation (AOT). Since code can be dynamically loaded, the runtime environment and the JIT are always present, even if code is compiled ahead of time.

The runtime provides a number of configuration options for running applications, for developing and debugging, and for testing and debugging the runtime itself.

The *mono* command uses the moving and generational SGen garbage collector while the *mono-boehm* command uses the conservative Boehm garbage collector.

PORTABILITY

On Unix-based systems, Mono provides a mechanism to emulate the Windows-style file access, this includes providing a case insensitive view of the file system, directory separator mapping (from \ to /) and stripping the drive letters.

This functionality is enabled by setting the **MONO_IOMAP** environment variable to one of **all, drive** and **case.**

See the description for **MONO_IOMAP** in the environment variables section for more details.

METHOD DESCRIPTIONS

A number of diagnostic command line options take as argument a method description. A method description is a textual representation that can be used to uniquely identify a method. The syntax is as follows:
[namespace]classname:methodname[(arguments)]

The values in brackets are optional, like the namespace and the arguments. The arguments themselves are either empty, or a comma-separated list of arguments. Both the **classname** and **methodname** can be set to the special value '*' to match any values (Unix shell users should escape the argument to avoid the shell interpreting this).

The arguments, if present should be a comma separated list of types either a full typename, or for built-in types it should use the low-level ILAsm type names for the built-in types, like 'void', 'char', 'bool', 'byte', 'sbyte', 'uint16', 'int16', 'uint',

Pointer types should be the name of the type, followed by a '*', arrays should be the typename followed by '[' one or more commas (to indicate the rank of the array), and ']'.

Generic values should use '<', one or more type names, separated by both a comma and a space and '>'.

By-reference arguments should include a "&" after the typename.

Examples:

```
*:ctor(int)        // All constructors that take an int as an argument
*:Main             // Methods named Main in any class
```

*:Main(string[]) // Methods named Main that take a string array in any class

RUNTIME OPTIONS

The following options are available:

--aot, **--aot[=options]**

This option is used to precompile the CIL code in the specified assembly to native code. The generated code is stored in a file with the extension .so. This file will be automatically picked up by the runtime when the assembly is executed. Ahead-of-Time compilation is most useful if you use it in combination with the -O=all,-shared flag which enables all of the optimizations in the code generator to be performed. Some of those optimizations are not practical for Just-in-Time compilation since they might be very time consuming. Unlike the .NET Framework, Ahead-of-Time compilation will not generate domain independent code: it generates the same code that the Just-in-Time compiler would produce. Since most applications use a single domain, this is fine. If you want to optimize the generated code for use in multi-domain applications, consider using the -O=shared flag. This pre-compiles the methods, but the original assembly is still required to execute as this one contains the metadata and exception information which is not available on the generated file. When precompiling code, you might want to compile with all optimizations (-O=all). Pre-compiled code is position independent code. Precompilation is just a mechanism to reduce startup time, increase code sharing across multiple mono processes and avoid just-in-time compilation program startup costs. The original assembly must still be present, as the metadata is contained there. AOT code typically can not be moved from one computer to another (CPU-specific optimizations that are detected at runtime) so you should not try to move the pre-generated assemblies or package the pre-generated assemblies for deployment. A few options are available as a parameter to the **--aot** command line option. The options are separated by commas, and more than one can be specified:

asmonly

Instructs the AOT compiler to output assembly code instead of an object file.

bind-to-runtime-version

If specified, forces the generated AOT files to be bound to the runtime version of the compiling Mono. This will prevent the AOT files from being consumed by a different Mono runtime.

data-outfile=FILE.dll.aotdata

This instructs the AOT code generator to output certain data constructs into a separate file. This can reduce the executable images some five to twenty percent. Developers need to then ship the resulting aotdata as a resource and register a hook to load the data on demand by using the *mono_install_load_aot_data_hook* method.

direct-icalls

When this option is specified, icalls (internal calls made from the standard library into the mono runtime code) are invoked directly instead of going through the operating system symbol lookup operation. This requires use of the *static* option.

direct-pinvoke

When this option is specified, P/Invoke methods are invoked directly instead of going through the operating system symbol lookup operation. This requires use of the *static* option.

dwarfdebug

Instructs the AOT compiler to emit DWARF debugging information. When used together with the nodebug option, only DWARF debugging information is emitted, but not the information that can be used at runtime.

full This creates binaries which can be used with the --full-aot option.

hybrid This creates binaries which can be used with the --hybrid-aot option.

llvm AOT will be performed with the LLVM backend instead of the Mono backend where possible. This will be slower to compile but most likely result in a performance improvement.

llvmonly
AOT will be performed with the LLVM backend exclusively and the Mono backend will not be used. The only output in this mode will be the bitcode file normally specified with the *llvm-outfile* option. Use of *llvmonly* automatically enables the *full* and *llvm* options. This feature is experimental.

llvmopts=[options]
Use this option to add more flags to the built-in set of flags passed to the LLVM optimizer. When you invoke the *mono* command with the *--aot=llvm* it displays the current list of flags that are being passed to the *opt* command. *The list of possible flags that can be passed can be* obtained by calling the bundled *opt* program that comes with Mono, and calling it like this:

```
opt --help
```

llvmllc=[options]
Use this option to add more flags to the built-in set of flags passed to the LLVM static compiler (llc). The list of possible flags that can be passed can be obtained by calling the bundled *llc* program that comes with Mono, and calling it like this:

```
llc --help
```

llvm-outfile=[filename]
Gives the path for the temporary LLVM bitcode file created during AOT. *dedup* Each AOT module will typically contain the code for inflated methods and wrappers that are called by code in that module. In dedup mode, we identify and skip compiling all of those methods. When using this mode with fullaot, dedup-include is required or these methods will remain missing.

dedup-include=[filename]
In dedup-include mode, we are in the pass of compilation where we compile the methods that we had previously skipped. All of them are emitted into the assembly that is passed as this option. We consolidate the many duplicate skipped copies of the same method into one.

info Print the architecture the AOT in this copy of Mono targets and quit.

interp Generates all required wrappers, so that it is possible to run --interpreter without any code generation at runtime. This option only makes sense with **mscorlib.dll**. Embedders can set

depfile=[filename]
Outputs a gcc -M style dependency file.

```
mono_jit_set_aot_mode (MONO_AOT_MODE_INTERP);
```

ld-flags Additional flags to pass to the C linker (if the current AOT mode calls for invoking it).

llvm-path=<PREFIX>
Same for the llvm tools 'opt' and 'llc'.

msym-dir=<PATH>
Instructs the AOT compiler to generate offline sequence points .msym files. The generated .msym files will be stored into a subfolder of <PATH> named as the compilation

AOTID.

mtriple=<TRIPLE>
: Use the GNU style target triple <TRIPLE> to determine some code generation options, i.e. --mtriple=armv7-linux-gnueabi will generate code that targets ARMv7. This is currently only supported by the ARM backend. In LLVM mode, this triple is passed on to the LLVM llc compiler.

nimt-trampolines=[number]
: When compiling in full aot mode, the IMT trampolines must be precreated in the AOT image. You can add additional method trampolines with this argument. Defaults to 512.

ngsharedvt-trampolines=[number]
: When compiling in full aot mode, the value type generic sharing trampolines must be precreated in the AOT image. You can add additional method trampolines with this argument. Defaults to 512.

nodebug
: Instructs the AOT compiler to not output any debugging information.

no-direct-calls
: This prevents the AOT compiler from generating a direct calls to a method. The AOT compiler usually generates direct calls for certain methods that do not require going through the PLT (for example, methods that are known to not require a hook like a static constructor) or call into simple internal calls.

nrgctx-trampolines=[number]
: When compiling in full aot mode, the generic sharing trampolines must be precreated in the AOT image. You can add additional method trampolines with this argument. Defaults to 4096.

nrgctx-fetch-trampolines=[number]
: When compiling in full aot mode, the generic sharing fetch trampolines must be precreated in the AOT image. You can add additional method trampolines with this argument. Defaults to 128.

ntrampolines=[number]
: When compiling in full aot mode, the method trampolines must be precreated in the AOT image. You can add additional method trampolines with this argument. Defaults to 4096.

outfile=[filename]
: Instructs the AOT compiler to save the output to the specified file.

print-skipped-methods
: If the AOT compiler cannot compile a method for any reason, enabling this flag will output the skipped methods to the console.

profile=[file]
: Specify a file to use for profile-guided optimization. See the **AOT profiler** sub-section. To specify multiple files, include the *profile* option multiple times.

profile-only
: AOT *only* the methods described in the files specified with the *profile* option. See the **AOT profiler** sub-section.

readonly-value=namespace.typename.fieldname=type/value
: Override the value of a static readonly field. Usually, during JIT compilation, the static constructor is ran eagerly, so the value of a static readonly field is known at compilation time and the compiler can do a number of optimizations based on it. During AOT, instead, the static constructor can't be ran, so this option can be used to set the value of such a field and enable the same set of optimizations. Type can be any of i1, i2, i4 for integers of the respective sizes (in bytes). Note that signed/unsigned numbers do not matter

here, just the storage size. This option can be specified multiple times and it doesn't prevent the static constructor for the type defining the field to execute with the usual rules at runtime (hence possibly computing a different value for the field).

save-temps,keep-temps
Instructs the AOT compiler to keep temporary files.

soft-debug
This instructs the compiler to generate sequence point checks that allow Mono's soft debugger to debug applications even on systems where it is not possible to set breakpoints or to single step (certain hardware configurations like the cell phones and video gaming consoles).

static Create an ELF object file (.o) or .s file which can be statically linked into an executable when embedding the mono runtime. When this option is used, the object file needs to be registered with the embedded runtime using the mono_aot_register_module function which takes as its argument the mono_aot_module_<ASSEMBLY NAME>_info global symbol from the object file:

```
extern void *mono_aot_module_hello_info;

mono_aot_register_module (mono_aot_module_hello_info);
```

stats Print various stats collected during AOT compilation.

temp-path=[path]
Explicitly specify path to store temporary files created during AOT compilation.

threads=[number]
This is an experimental option for the AOT compiler to use multiple threads when compiling the methods.

tool-prefix=<PREFIX>
Prepends <PREFIX> to the name of tools ran by the AOT compiler, i.e. 'as'/'ld'. For example, --tool=prefix=arm-linux-gnueabi- will make the AOT compiler run

verbose
Prints additional information about type loading failures.

write-symbols,no-write-symbols
Instructs the AOT compiler to emit (or not emit) debug symbol information.

no-opt Instructs the AOT compiler tot no call opt when compiling with LLVM.

For more information about AOT, see: http://www.mono-project.com/docs/advanced/aot/

--aot-path=PATH
List of additional directories to search for AOT images.

--apply-bindings=FILE
Apply the assembly bindings from the specified configuration file when running the AOT compiler. This is useful when compiling an auxiliary assembly that is referenced by a main assembly that provides a configuration file. For example, if app.exe uses lib.dll then in order to make the assembly bindings from app.exe.config available when compiling lib.dll ahead of time, use:
mono --apply-bindings=app.exe.config --aot lib.dll

--assembly-loader=MODE
If mode is **strict**, Mono will check that the public key token, culture and version of a candidate assembly matches the requested strong name. If mode is **legacy**, as long as the name matches, the candidate will be allowed. **strict** is the behavior consistent with .NET Framework but may break some existing mono-based applications. The default is **legacy**.

--attach=[options]
Currently the only option supported by this command line argument is **disable** which disables the attach functionality.

--config filename
Load the specified configuration file instead of the default one(s). The default files are /etc/mono/config and ~/.mono/config or the file specified in the MONO_CONFIG environment variable, if set. See the mono-config(5) man page for details on the format of this file.

--debugger-agent=[options]
This instructs the Mono runtime to start a debugging agent inside the Mono runtime and connect it to a client user interface will control the Mono process. This option is typically used by IDEs, like the MonoDevelop or Visual Studio IDEs.
The configuration is specified using one of more of the following options:

address=host:port
Use this option to specify the IP address where your debugger client is listening to.

loglevel=LEVEL
Specifies the diagnostics log level for

logfile=filename
Used to specify the file where the log will be stored, it defaults to standard output.

server=[y/n]
Defaults to no, with the default option Mono will actively connect to the host/port configured with the **address** option. If you set it to 'y', it instructs the Mono runtime to start debugging in server mode, where Mono actively waits for the debugger front end to connect to the Mono process. Mono will print out to stdout the IP address and port where it is listening.

setpgid=[y/n]
If set to yes, Mono will call **setpgid(0, 0)** on startup, if that function is available on the system. This is useful for ensuring that signals delivered to a process that is executing the debuggee are not propagated to the debuggee, e.g. when Ctrl-C sends **SIGINT** to the **sdb** tool.

suspend=[y/n]
Defaults to yes, with the default option Mono will suspend the vm on startup until it connects successfully to a debugger front end. If you set it to 'n', in conjunction with **server=y**, it instructs the Mono runtime to run as normal, while caching metadata to send to the debugger front end on connection..

transport=transport_name
This is used to specify the transport that the debugger will use to communicate. It must be specified and currently requires this to be 'dt_socket'.

--desktop
Configures the virtual machine to be better suited for desktop applications. Currently this sets the GC system to avoid expanding the heap as much as possible at the expense of slowing down garbage collection a bit.

--full-aot
This flag instructs the Mono runtime to not generate any code at runtime and depend exclusively on the code generated from using mono --aot=full previously. This is useful for platforms that do not permit dynamic code generation, or if you need to run assemblies that have been stripped of IL (for example using mono-cil-strip). Notice that this feature will abort execution at runtime if a codepath in your program, or Mono's class libraries attempts to generate code dynamically. You should test your software upfront and make

sure that you do not use any dynamic features.

--full-aot-interp
Same as --full-aot with fallback to the interpreter.

--gc=boehm, **--gc=sgen**
Selects the Garbage Collector engine for Mono to use, Boehm or SGen. Currently this merely ensures that you are running either the *mono* or *mono-sgen* commands. This flag can be set in the **MONO_ENV_OPTIONS** environment variable to force all of your child processes to use one particular kind of GC with the Mono runtime.

--gc-debug=[options]
Command line equivalent of the **MONO_GC_DEBUG** environment variable.

--gc-params=[options]
Command line equivalent of the **MONO_GC_PARAMS** environment variable.

--arch=32, **--arch=64**
(Mac OS X only): Selects the bitness of the Mono binary used, if available. If the binary used is already for the selected bitness, nothing changes. If not, the execution switches to a binary with the selected bitness suffix installed side by side (for example, '/bin/mono --arch=64' will switch to '/bin/mono64' iff '/bin/mono' is a 32-bit build).

--help, **-h**
Displays usage instructions.

--interpreter
The Mono runtime will use its interpreter to execute a given assembly. The interpreter is usually slower than the JIT, but it can be useful on platforms where code generation at runtime is not allowed.

--hybrid-aot
This flag allows the Mono runtime to run assemblies that have been stripped of IL, for example using mono-cil-strip. For this to work, the assembly must have been AOT compiled with --aot=hybrid.

This flag is similar to --full-aot, but it does not disable the JIT. This means you can use dynamic features such as System.Reflection.Emit.

--llvm If the Mono runtime has been compiled with LLVM support (not available in all configurations), Mono will use the LLVM optimization and code generation engine to JIT or AOT compile. For more information, consult: http://www.mono-project.com/docs/advanced/mono-llvm/

--nollvm
When using a Mono that has been compiled with LLVM support, it forces Mono to fallback to its JIT engine and not use the LLVM backend.

--optimize=MODE, **-O=MODE**
MODE is a comma separated list of optimizations. They also allow optimizations to be turned off by prefixing the optimization name with a minus sign. In general, Mono has been tuned to use the default set of flags, before using these flags for a deployment setting, you might want to actually measure the benefits of using them. The following optimization flags are implemented in the core engine:

abcrem Array bound checks removal
all Turn on all optimizations
aot Usage of Ahead Of Time compiled code
branch Branch optimizations
cfold Constant folding
cmov Conditional moves [arch-dependency]
deadce Dead code elimination

consprop Constant propagation
copyprop Copy propagation
fcmov Fast x86 FP compares [arch-dependency]
float32 Perform 32-bit float arithmetic using 32-bit operations
gshared Enable generic code sharing.
inline Inline method calls
intrins Intrinsic method implementations
linears Linear scan global reg allocation
leaf Leaf procedures optimizations
loop Loop related optimizations
peephole Peephole postpass
precomp Precompile all methods before executing Main
sched Instruction scheduling
shared Emit per-domain code
sse2 SSE2 instructions on x86 [arch-dependency]
tailc Tail recursion and tail calls

For example, to enable all the optimization but dead code elimination and inlining, you can use:

-O=all,-deadce,-inline

The flags that are flagged with [arch-dependency] indicate that the given option if used in combination with Ahead of Time compilation (--aot flag) would produce pre-compiled code that will depend on the current CPU and might not be safely moved to another computer.

The following optimizations are supported

float32 Requests that the runtime performn 32-bit floating point operations using only 32-bits. By default the Mono runtime tries to use the highest precision available for floating point operations, but while this might render better results, the code might run slower. This options also affects the code generated by the LLVM backend.

inline Controls whether the runtime should attempt to inline (the default), or not inline methods invocations

--response=FILE Provides a response file, this instructs the Mono command to read other command line options from the specified file, as if the options had been specified on the command line. Useful when you have very long command lines.

--runtime=VERSION

Mono supports different runtime versions. The version used depends on the program that is being run or on its configuration file (named program.exe.config). This option can be used to override such autodetection, by forcing a different runtime version to be used. Note that this should only be used to select a later compatible runtime version than the one the program was compiled against. A typical usage is for running a 1.1 program on a 2.0 version:

mono --runtime=v2.0.50727 program.exe

--server

Configures the virtual machine to be better suited for server operations (currently, allows a heavier threadpool initialization).

--use-map-jit

Instructs Mono to generate code using MAP_JIT on MacOS. Necessary for bundled applications.

--verify-all

Verifies mscorlib and assemblies in the global assembly cache for valid IL, and all user code for IL verifiability.

This is different from **--security**'s verifiable or validil in that these options only check user code and skip mscorlib and assemblies located on the global assembly cache.

-V, **--version**
Prints JIT version information (system configuration, release number and branch names if available).

--version=number
Print version number only.

DEVELOPMENT OPTIONS

The following options are used to help when developing a JITed application.

--debug, **--debug=OPTIONS**
Turns on the debugging mode in the runtime. If an assembly was compiled with debugging information, it will produce line number information for stack traces.

The optional OPTIONS argument is a comma separated list of debugging options. These options are turned off by default since they generate much larger and slower code at runtime.

The following options are supported:

casts Produces a detailed error when throwing a InvalidCastException. This option needs to be enabled as this generates more verbose code at execution time.

mdb-optimizations
Disable some JIT optimizations which are usually only disabled when running inside the debugger. This can be helpful if you want to attach to the running process with mdb.

gdb Generate and register debugging information with gdb. This is only supported on some platforms, and only when using gdb 7.0 or later.

−−profile[=*profiler*[:*profiler_args*]]
Loads a profiler module with the given arguments. For more information, see the **PROFILING** section. This option can be used multiple times; each time will load an additional profiler module.

--trace[=expression]
Shows method names as they are invoked. By default all methods are traced. The trace can be customized to include or exclude methods, classes or assemblies. A trace expression is a comma separated list of targets, each target can be prefixed with a minus sign to turn off a particular target. The words 'program', 'all' and 'disabled' have special meaning. 'program' refers to the main program being executed, and 'all' means all the method calls. The 'disabled' option is used to start up with tracing disabled. It can be enabled at a later point in time in the program by sending the SIGUSR2 signal to the runtime. Assemblies are specified by their name, for example, to trace all calls in the System assembly, use:

mono --trace=System app.exe

Classes are specified with the T: prefix. For example, to trace all calls to the System.String class, use:

mono --trace=T:System.String app.exe

And individual methods are referenced with the M: prefix, and the standard method notation:

mono --trace=M:System.Console:WriteLine app.exe

Exceptions can also be traced, it will cause a stack trace to be printed every time an exception of the specified type is thrown. The exception type can be specified with or without the namespace,

and to trace all exceptions, specify 'all' as the type name.

mono --trace=E:System.Exception app.exe

As previously noted, various rules can be specified at once:

mono --trace=T:System.String,T:System.Random app.exe

You can exclude pieces, the next example traces calls to System.String except for the System.String:Concat method.

mono --trace=T:System.String,-M:System.String:Concat

You can trace managed to unmanaged transitions using the wrapper qualifier:

mono --trace=wrapper app.exe

Finally, namespaces can be specified using the N: prefix:

mono --trace=N:System.Xml

--no-x86-stack-align
Don't align stack frames on the x86 architecture. By default, Mono aligns stack frames to 16 bytes on x86, so that local floating point and SIMD variables can be properly aligned. This option turns off the alignment, which usually saves one instruction per call, but might result in significantly lower floating point and SIMD performance.

--jitmap
Generate a JIT method map in a /tmp/perf-PID.map file. This file is then used, for example, by the perf tool included in recent Linux kernels. Each line in the file has:

HEXADDR HEXSIZE methodname

Currently this option is only supported on Linux.

JIT MAINTAINER OPTIONS

The maintainer options are only used by those developing the runtime itself, and not typically of interest to runtime users or developers.

--bisect=optimization:filename
This flag is used by the automatic optimization bug bisector. It takes an optimization flag and a filename of a file containing a list of full method names, one per line. When it compiles one of the methods in the file it will use the optimization given, in addition to the optimizations that are otherwise enabled. Note that if the optimization is enabled by default, you should disable it with '-O', otherwise it will just apply to every method, whether it's in the file or not.

--break method
Inserts a breakpoint before the method whose name is 'method' (namespace.class:methodname). Use 'Main' as method name to insert a breakpoint on the application's main method. You can use it also with generics, for example "System.Collections.Generic.Queue`1:Peek"

--breakonex
Inserts a breakpoint on exceptions. This allows you to debug your application with a native debugger when an exception is thrown.

--compile name
This compiles a method (namespace.name:methodname), this is used for testing the compiler performance or to examine the output of the code generator.

--compile-all
Compiles all the methods in an assembly. This is used to test the compiler performance or to examine the output of the code generator

--graph=TYPE METHOD
This generates a postscript file with a graph with the details about the specified method (namespace.name:methodname). This requires 'dot' and ghostview to be installed (it expects Ghostview to be called "gv"). The following graphs are available:

cfg Control Flow Graph (CFG)
dtree Dominator Tree
code CFG showing code
ssa CFG showing code after SSA translation
optcode CFG showing code after IR optimizations

Some graphs will only be available if certain optimizations are turned on.

--ncompile
Instruct the runtime on the number of times that the method specified by --compile (or all the methods if --compile-all is used) to be compiled. This is used for testing the code generator performance.

--stats Displays information about the work done by the runtime during the execution of an application.

--wapi=hps|semdel
Perform maintenance of the process shared data. semdel will delete the global semaphore. hps will list the currently used handles.

-v, **--verbose**
Increases the verbosity level, each time it is listed, increases the verbosity level to include more information (including, for example, a disassembly of the native code produced, code selector info etc.).

ATTACH SUPPORT

The Mono runtime allows external processes to attach to a running process and load assemblies into the running program. To attach to the process, a special protocol is implemented in the Mono.Management assembly.

With this support it is possible to load assemblies that have an entry point (they are created with -target:exe or -target:winexe) to be loaded and executed in the Mono process.

The code is loaded into the root domain, and it starts execution on the special runtime attach thread. The attached program should create its own threads and return after invocation.

This support allows for example debugging applications by having the csharp shell attach to running processes.

PROFILING

The Mono runtime includes a profiler API that dynamically loaded profiler modules and embedders can use to collect performance-related data about an application. Profiler modules are loaded by passing the **−−profile** command line argument to the Mono runtime.

Mono ships with a few profiler modules, of which the **log** profiler is the most feature-rich. It is also the default profiler if the *profiler* argument is not given, or if **default** is given. It is possible to write your own profiler modules; see the **Custom profilers** sub-section.

Log profiler

The log profiler can be used to collect a lot of information about a program running in the Mono runtime. This data can be used (both while the process is running and later) to do analyses of the program behavior, determine resource usage, performance issues or even look for particular execution patterns.

This is accomplished by logging the events provided by the Mono runtime through the profiler API and periodically writing them to a file which can later be inspected with the **mprof−report**(1) tool.

More information about how to use the log profiler is available on the **mono−profilers**(1) page, under the

LOG PROFILER section, as well as the **mprof−report**(1) page.

Coverage profiler

The code coverage profiler can instrument a program to help determine which classes, methods, code paths, etc are actually executed. This is most useful when running a test suite to determine whether the tests actually cover the code they're expected to.

More information about how to use the coverage profiler is available on the **mono−profilers**(1) page, under the **COVERAGE PROFILER** section.

AOT profiler

The AOT profiler can help improve startup performance by logging which generic instantiations are used by a program, which the AOT compiler can then use to compile those instantiations ahead of time so that they won't have to be JIT compiled at startup.

More information about how to use the AOT profiler is available on the **mono−profilers**(1) page, under the **AOT PROFILER** section.

Custom profilers

Custom profiler modules can be loaded in exactly the same way as the standard modules that ship with Mono. They can also access the same profiler API to gather all kinds of information about the code being executed.

For example, to use a third-party profiler called **custom**, you would load it like this:

```
mono --profile=custom program.exe
```

You could also pass arguments to it:

```
mono --profile=custom:arg1,arg2=arg3 program.exe
```

In the above example, Mono will load the profiler from the shared library called *libmono−profiler−custom.so* (name varies based on platform, e.g., *libmono−profiler−custom.dylib* on OS X). This profiler module must be on your dynamic linker library path (**LD_LIBRARY_PATH** on most systems, **DYLD_LIBRARY_PATH** on OS X).

For a sample of how to write your own custom profiler, look at the *samples/profiler/sample.c* file in the Mono source tree.

DEBUGGING AIDS

To debug managed applications, you can use the **mdb** command, a command line debugger.

It is possible to obtain a stack trace of all the active threads in Mono by sending the QUIT signal to Mono, you can do this from the command line, like this:

```
kill -QUIT pid
```

Where pid is the Process ID of the Mono process you want to examine. The process will continue running afterwards, but its state is not guaranteed.

Important: this is a last-resort mechanism for debugging applications and should not be used to monitor or probe a production application. The integrity of the runtime after sending this signal is not guaranteed and the application might crash or terminate at any given point afterwards.

The **--debug=casts** option can be used to get more detailed information for Invalid Cast operations, it will provide information about the types involved.

You can use the MONO_LOG_LEVEL and MONO_LOG_MASK environment variables to get verbose debugging output about the execution of your application within Mono.

The *MONO_LOG_LEVEL* environment variable if set, the logging level is changed to the set value. Possible values are "error", "critical", "warning", "message", "info", "debug". The default value is "error". Messages with a logging level greater then or equal to the log level will be printed to stdout/stderr.

Use "info" to track the dynamic loading of assemblies.

Use the *MONO_LOG_MASK* environment variable to limit the extent of the messages you get: If set, the log mask is changed to the set value. Possible values are "asm" (assembly loader), "type", "dll" (native library loader), "gc" (garbage collector), "cfg" (config file loader), "aot" (precompiler), "security" (e.g. Moonlight CoreCLR support), "threadpool" (thread pool generic), "io-selector" (async socket operations), "io-layer" (I/O layer - processes, files, sockets, events, semaphores, mutexes and handles), "io-layer-process", "io-layer-file", "io-layer-socket", "io-layer-event", "io-layer-semaphore", "io-layer-mutex", "io-layer-handle" and "all". The default value is "all". Changing the mask value allows you to display only messages for a certain component. You can use multiple masks by comma separating them. For example to see config file messages and assembly loader messages set you mask to "asm,cfg".

The following is a common use to track down problems with P/Invoke:

```
$ MONO_LOG_LEVEL="debug" MONO_LOG_MASK="dll" mono glue.exe
```

DEBUGGING WITH LLDB

If you are using LLDB, you can use the **mono.py** script to print some internal data structures with it. To use this, add this to your **$HOME/.lldbinit** file:
command script import $PREFIX/lib/mono/lldb/mono.py

Where $PREFIX is the prefix value that you used when you configured Mono (typically /usr).

Once this is done, then you can inspect some Mono Runtime data structures, for example:
(lldb) p method

(MonoMethod *) $0 = 0x05026ac0 [mscorlib]System.OutOfMemoryException:.ctor()

SERIALIZATION

Mono's XML serialization engine by default will use a reflection-based approach to serialize which might be slow for continuous processing (web service applications). The serialization engine will determine when a class must use a hand-tuned serializer based on a few parameters and if needed it will produce a customized C# serializer for your types at runtime. This customized serializer then gets dynamically loaded into your application.

You can control this with the MONO_XMLSERIALIZER_THS environment variable.

The possible values are **'no'** to disable the use of a C# customized serializer, or an integer that is the minimum number of uses before the runtime will produce a custom serializer (0 will produce a custom serializer on the first access, 50 will produce a serializer on the 50th use). Mono will fallback to an interpreted serializer if the serializer generation somehow fails. This behavior can be disabled by setting the option **'nofallback'** (for example: MONO_XMLSERIALIZER_THS=0,nofallback).

ENVIRONMENT VARIABLES

GC_DONT_GC

Turns off the garbage collection in Mono. This should be only used for debugging purposes

HTTP_PROXY

(Also **http_proxy**) If set, web requests using the Mono Class Library will be automatically proxied through the given URL. Not supported on Windows, Mac OS, iOS or Android. See also **NO_PROXY**.

LLVM_COUNT

When Mono is compiled with LLVM support, this instructs the runtime to stop using LLVM after the specified number of methods are JITed. This is a tool used in diagnostics to help isolate problems in the code generation backend. For example **LLVM_COUNT=10** would only compile 10 methods with LLVM and then switch to the Mono JIT engine. **LLVM_COUNT=0** would disable the LLVM engine altogether.

MONO_ASPNET_INHIBIT_SETTINGSMAP

Mono contains a feature which allows modifying settings in the .config files shipped with Mono by using config section mappers. The mappers and the mapping rules are defined in the

$prefix/etc/mono/2.0/settings.map file and, optionally, in the settings.map file found in the top-level directory of your ASP.NET application. Both files are read by System.Web on application startup, if they are found at the above locations. If you don't want the mapping to be performed you can set this variable in your environment before starting the application and no action will be taken.

MONO_ASPNET_WEBCONFIG_CACHESIZE
Mono has a cache of ConfigSection objects for speeding up WebConfigurationManager queries. Its default size is 100 items, and when more items are needed, cache evictions start happening. If evictions are too frequent this could impose unnecessary overhead, which could be avoided by using this environment variable to set up a higher cache size (or to lower memory requirements by decreasing it).

MONO_CAIRO_DEBUG_DISPOSE
If set, causes Mono.Cairo to collect stack traces when objects are allocated, so that the finalization/Dispose warnings include information about the instance's origin.

MONO_CFG_DIR
If set, this variable overrides the default system configuration directory ($PREFIX/etc). It's used to locate machine.config file.

MONO_COM
Sets the style of COM interop. If the value of this variable is "MS" Mono will use string marhsalling routines from the liboleaut32 for the BSTR type library, any other values will use the mono-builtin BSTR string marshalling.

MONO_CONFIG
If set, this variable overrides the default runtime configuration file ($PREFIX/etc/mono/config). The --config command line options overrides the environment variable.

MONO_CPU_ARCH
Override the automatic cpu detection mechanism. Currently used only on arm. The format of the value is as follows:

```
"armvV [thumb[2]]"
```

where V is the architecture number 4, 5, 6, 7 and the options can be currently be "thumb" or "thumb2". Example:

```
MONO_CPU_ARCH="armv4 thumb" mono ...
```

MONO_ARM_FORCE_SOFT_FLOAT
When Mono is built with a soft float fallback on ARM and this variable is set to "1", Mono will always emit soft float code, even if a VFP unit is detected.

MONO_DARWIN_USE_KQUEUE_FSW
Fall back on the kqueue FileSystemWatcher implementation in Darwin. The default is the FSEvent implementation.

MONO_DARWIN_WATCHER_MAXFDS
This is a debugging aid used to force limits on the kqueue FileSystemWatcher implementation in Darwin. There is no limit by default.

MONO_DISABLE_MANAGED_COLLATION
If this environment variable is 'yes', the runtime uses unmanaged collation (which actually means no culture-sensitive collation). It internally disables managed collation functionality invoked via the members of System.Globalization.CompareInfo class. Collation is enabled by default.

MONO_DISABLE_SHARED_AREA
Unix only: If set, disable usage of shared memory for exposing performance counters. This means it will not be possible to both externally read performance counters from this processes or read those of external processes.

MONO_DNS
When set, enables the use of a fully managed DNS resolver instead of the regular libc functions. This resolver performs much better when multiple queries are run in parallel.

Note that /etc/nsswitch.conf will be ignored.

MONO_EGD_SOCKET
For platforms that do not otherwise have a way of obtaining random bytes this can be set to the name of a file system socket on which an egd or prngd daemon is listening.

MONO_ENABLE_AIO
If set, tells mono to attempt using native asynchronous I/O services. If not set, a default select/poll implementation is used. Currently epoll and kqueue are supported.

MONO_THREADS_SUSPEND Selects a mechanism that Mono will use to suspend
threads. May be set to "preemptive", "coop", or "hybrid". Threads may need to be suspended by the debugger, or using some .NET threading APIs, and most commonly when the SGen garbage collector needs to stop all threads during a critical phase of garbage collection. Preemptive mode is the mode that Mono has used historically, going back to the Boehm days, where the garbage collector would run at any point and suspend execution of all threads as required to perform a garbage collection. The cooperative mode on the other hand requires the cooperation of all threads to stop at a safe point. This makes for an easier to debug garbage collector and it improves the stability of the runtime because threads are not suspended when accessing critical resources. In scenarios where Mono is embedded in another application, cooperative suspend requires the embedder code to follow coding guidelines in order to cooperate with the garbage collector. Cooperative suspend in embedded Mono is currently experimental. Hybrid mode is a combination of the two that retains better compatability with scenarios where Mono is embedded in another application: threads that are running managed code or code that comprises the Mono runtime will be cooperatively suspended, while threads running embedder code will be preemptively suspended. Hybrid suspend is the default on some desktop platforms.

Alternatively, coop and hybrid mode can be enabled at compile time by using the --enable-cooperative-suspend or --enable-hybrid-suspend flags, respectively, when calling configure. The **MONO_THREADS_SUSPEND** environment variable takes priority over the compiled default.

MONO_ENABLE_COOP_SUSPEND
This environment variable is obsolete, but retained for backward compatibility. Use **MONO_THREADS_SUSPEND** set to "coop" instead. Note that if configure flags were provided to enable cooperative or hybrid suspend, this variable is ignored.

MONO_ENV_OPTIONS
This environment variable allows you to pass command line arguments to a Mono process through the environment. This is useful for example to force all of your Mono processes to use LLVM or SGEN without having to modify any launch scripts.

MONO_SDB_ENV_OPTIONS
Used to pass extra options to the debugger agent in the runtime, as they were passed using --debugger-agent=.

MONO_EVENTLOG_TYPE
Sets the type of event log provider to use (for System.Diagnostics.EventLog). Possible values are:

local[:path]
Persists event logs and entries to the local file system. The directory in which to persist the event logs, event sources and entries can be specified as part of the value. If the path

is not explicitly set, it defaults to "/var/lib/mono/eventlog" on unix and "%APPDATA%no\ventlog" on Windows.

win32 **Uses the native win32 API to write events and registers event logs and** event sources in the registry. This is only available on Windows. On Unix, the directory permission for individual event log and event source directories is set to 777 (with +t bit) allowing everyone to read and write event log entries while only allowing entries to be deleted by the user(s) that created them.

null Silently discards any events.

The default is "null" on Unix (and versions of Windows before NT), and "win32" on Windows NT (and higher).

MONO_EXTERNAL_ENCODINGS

If set, contains a colon-separated list of text encodings to try when turning externally-generated text (e.g. command-line arguments or filenames) into Unicode. The encoding names come from the list provided by iconv, and the special case "default_locale" which refers to the current locale's default encoding.

When reading externally-generated text strings UTF-8 is tried first, and then this list is tried in order with the first successful conversion ending the search. When writing external text (e.g. new filenames or arguments to new processes) the first item in this list is used, or UTF-8 if the environment variable is not set.

The problem with using MONO_EXTERNAL_ENCODINGS to process your files is that it results in a problem: although its possible to get the right file name it is not necessarily possible to open the file. In general if you have problems with encodings in your filenames you should use the "convmv" program.

MONO_GC_PARAMS

When using Mono with the SGen garbage collector this variable controls several parameters of the collector. The variable's value is a comma separated list of words.

max-heap-size=*size*

Sets the maximum size of the heap. The size is specified in bytes and must be a power of two. The suffixes 'k', 'm' and 'g' can be used to specify kilo-, mega- and gigabytes, respectively. The limit is the sum of the nursery, major heap and large object heap. Once the limit is reached the application will receive OutOfMemoryExceptions when trying to allocate. Not the full extent of memory set in max-heap-size could be available to satisfy a single allocation due to internal fragmentation. By default heap limits is disabled and the GC will try to use all available memory.

nursery-size=*size*

Sets the size of the nursery. The size is specified in bytes and must be a power of two. The suffixes 'k', 'm' and 'g' can be used to specify kilo-, mega- and gigabytes, respectively. The nursery is the first generation (of two). A larger nursery will usually speed up the program but will obviously use more memory. The default nursery size 4 MB.

major=*collector*

Specifies which major collector to use. Options are 'marksweep' for the Mark&Sweep collector, 'marksweep-conc' for concurrent Mark&Sweep and 'marksweep-conc-par' for parallel and concurrent Mark&Sweep. The concurrent Mark&Sweep collector is the default.

mode=balanced|throughput|pause[:*max-pause*]

Specifies what should be the garbage collector's target. The 'throughput' mode aims to reduce time spent in the garbage collector and improve application speed, the 'pause' mode aims to keep pause times to a minimum and it receives the argument *max-pause* which specifies the maximum pause time in milliseconds that is acceptable and the 'balanced' mode which is a general purpose optimal mode.

soft-heap-limit=*size*

Once the heap size gets larger than this size, ignore what the default major collection trigger metric says and only allow four nursery size's of major heap growth between major collections.

evacuation-threshold=*threshold*

Sets the evacuation threshold in percent. This option is only available on the Mark&Sweep major collectors. The value must be an integer in the range 0 to 100. The default is 66. If the sweep phase of the collection finds that the occupancy of a specific heap block type is less than this percentage, it will do a copying collection for that block type in the next major collection, thereby restoring occupancy to close to 100 percent. A value of 0 turns evacuation off.

(no-)lazy-sweep

Enables or disables lazy sweep for the Mark&Sweep collector. If enabled, the sweeping of individual major heap blocks is done piecemeal whenever the need arises, typically during nursery collections. Lazy sweeping is enabled by default.

(no-)concurrent-sweep

Enables or disables concurrent sweep for the Mark&Sweep collector. If enabled, the iteration of all major blocks to determine which ones can be freed and which ones have to be kept and swept, is done concurrently with the running program. Concurrent sweeping is enabled by default.

stack-mark=*mark-mode*

Specifies how application threads should be scanned. Options are 'precise' and 'conservative'. Precise marking allow the collector to know what values on stack are references and what are not. Conservative marking threats all values as potentially references and leave them untouched. Precise marking reduces floating garbage and can speed up nursery collection and allocation rate, it has the downside of requiring a significant extra memory per compiled method. The right option, unfortunately, requires experimentation.

save-target-ratio=*ratio*

Specifies the target save ratio for the major collector. The collector lets a given amount of memory to be promoted from the nursery due to minor collections before it triggers a major collection. This amount is based on how much memory it expects to free. It is represented as a ratio of the size of the heap after a major collection. Valid values are between 0.1 and 2.0. The default is 0.5. Smaller values will keep the major heap size smaller but will trigger more major collections. Likewise, bigger values will use more memory and result in less frequent major collections. This option is EXPERIMENTAL, so it might disappear in later versions of mono.

default-allowance-ratio=*ratio*

Specifies the default allocation allowance when the calculated size is too small. The allocation allowance is how much memory the collector let be promoted before triggered a major collection. It is a ratio of the nursery size. Valid values are between 1.0 and 10.0. The default is 4.0. Smaller values lead to smaller heaps and more frequent major collections. Likewise, bigger values will allow the heap to grow faster but use more memory when it reaches a stable size. This option is EXPERIMENTAL, so it might disappear in later versions of mono.

minor=*minor-collector*

Specifies which minor collector to use. Options are 'simple' which promotes all objects from the nursery directly to the old generation, 'simple-par' which has same promotion behavior as 'simple' but using multiple workers and 'split' which lets objects stay longer on the nursery before promoting.

alloc-ratio=*ratio*
Specifies the ratio of memory from the nursery to be use by the alloc space. This only can only be used with the split minor collector. Valid values are integers between 1 and 100. Default is 60.

promotion-age=*age*
Specifies the required age of an object must reach inside the nursery before been promoted to the old generation. This only can only be used with the split minor collector. Valid values are integers between 1 and 14. Default is 2.

(no-)cementing
Enables or disables cementing. This can dramatically shorten nursery collection times on some benchmarks where pinned objects are referred to from the major heap.

allow-synchronous-major
This forbids the major collector from performing synchronous major collections. The major collector might want to do a synchronous collection due to excessive fragmentation. Disabling this might trigger OutOfMemory error in situations that would otherwise not happen.

MONO_GC_DEBUG
When using Mono with the SGen garbage collector this environment variable can be used to turn on various debugging features of the collector. The value of this variable is a comma separated list of words. Do not use these options in production.

number Sets the debug level to the specified number.

print-allowance
After each major collection prints memory consumption for before and after the collection and the allowance for the minor collector, i.e. how much the heap is allowed to grow from minor collections before the next major collection is triggered.

print-pinning
Gathers statistics on the classes whose objects are pinned in the nursery and for which global remset entries are added. Prints those statistics when shutting down.

collect-before-allocs

check-remset-consistency
This performs a remset consistency check at various opportunities, and also clears the nursery at collection time, instead of the default, when buffers are allocated (clear-at-gc). The consistency check ensures that there are no major to minor references that are not on the remembered sets.

mod-union-consistency-check
Checks that the mod-union cardtable is consistent before each finishing major collection pause. This check is only applicable to concurrent major collectors.

check-mark-bits
Checks that mark bits in the major heap are consistent at the end of each major collection. Consistent mark bits mean that if an object is marked, all objects that it had references to must also be marked.

check-nursery-pinned
After nursery collections, and before starting concurrent collections, check whether all nursery objects are pinned, or not pinned - depending on context. Does nothing when the split nursery collector is used.

xdomain-checks
Performs a check to make sure that no references are left to an unloaded AppDomain.

clear-at-tlab-creation
Clears the nursery incrementally when the thread local allocation buffers (TLAB) are created. The default setting clears the whole nursery at GC time.

debug-clear-at-tlab-creation
Clears the nursery incrementally when the thread local allocation buffers (TLAB) are created, but at GC time fills it with the byte '0xff', which should result in a crash more quickly if 'clear-at-tlab-creation' doesn't work properly.

clear-at-gc
This clears the nursery at GC time instead of doing it when the thread local allocation buffer (TLAB) is created. The default is to clear the nursery at TLAB creation time.

disable-minor
Don't do minor collections. If the nursery is full, a major collection is triggered instead, unless it, too, is disabled.

disable-major
Don't do major collections.

conservative-stack-mark
Forces the GC to scan the stack conservatively, even if precise scanning is available.

no-managed-allocator
Disables the managed allocator.

check-scan-starts
If set, does a plausibility check on the scan_starts before and after each collection

verify-nursery-at-minor-gc
If set, does a complete object walk of the nursery at the start of each minor collection.

dump-nursery-at-minor-gc
If set, dumps the contents of the nursery at the start of each minor collection. Requires verify-nursery-at-minor-gc to be set.

heap-dump=*file*
Dumps the heap contents to the specified file. To visualize the information, use the mono-heapviz tool.

binary-protocol=*file*
Outputs the debugging output to the specified file. For this to work, Mono needs to be compiled with the BINARY_PROTOCOL define on sgen-gc.c. You can then use this command to explore the output

```
sgen-grep-binprot 0x1234 0x5678 < file
```

nursery-canaries
If set, objects allocated in the nursery are suffixed with a canary (guard) word, which is checked on each minor collection. Can be used to detect/debug heap corruption issues.

do-not-finalize(=*classes***)**
If enabled, finalizers will not be run. Everything else will be unaffected: finalizable objects will still be put into the finalization queue where they survive until they're scheduled to finalize. Once they're not in the queue anymore they will be collected regularly. If a list of comma-separated class names is given, only objects from those classes will not be finalized.

log-finalizers
Log verbosely around the finalization process to aid debugging.

MONO_GAC_PREFIX
Provides a prefix the runtime uses to look for Global Assembly Caches. Directories are separated by the platform path separator (colons on unix). MONO_GAC_PREFIX should point to the top directory of a prefixed install. Or to the directory provided in the gacutil /gacdir command. Example: **/home/username/.mono:/usr/local/mono/**

MONO_IOMAP
Enables some filename rewriting support to assist badly-written applications that hard-code Windows paths. Set to a colon-separated list of "drive" to strip drive letters, or "case" to do case-insensitive file matching in every directory in a path. "all" enables all rewriting methods. (Backslashes are always mapped to slashes if this variable is set to a valid option).
For example, this would work from the shell:

```
MONO_IOMAP=drive:case
export MONO_IOMAP
```

If you are using mod_mono to host your web applications, you can use the **MonoIOMAP** directive instead, like this:

```
MonoIOMAP <appalias> all
```

See mod_mono(8) for more details.

MONO_LLVM
When Mono is using the LLVM code generation backend you can use this environment variable to pass code generation options to the LLVM compiler.

MONO_MANAGED_WATCHER
If set to "disabled", System.IO.FileSystemWatcher will use a file watcher implementation which silently ignores all the watching requests. If set to any other value, System.IO.FileSystemWatcher will use the default managed implementation (slow). If unset, mono will try to use inotify, FAM, Gamin, kevent under Unix systems and native API calls on Windows, falling back to the managed implementation on error.

MONO_MESSAGING_PROVIDER
Mono supports a plugin model for its implementation of System.Messaging making it possible to support a variety of messaging implementations (e.g. AMQP, ActiveMQ). To specify which messaging implementation is to be used the evironement variable needs to be set to the full class name for the provider. E.g. to use the RabbitMQ based AMQP implementation the variable should be set to:

```
Mono.Messaging.RabbitMQ.RabbitMQMessagingProvider,Mono.Messaging.RabbitMQ
```

MONO_NO_SMP
If set causes the mono process to be bound to a single processor. This may be useful when debugging or working around race conditions.

MONO_NO_TLS
Disable inlining of thread local accesses. Try setting this if you get a segfault early on in the execution of mono.

MONO_PATH
Provides a search path to the runtime where to look for library files. This is a tool convenient for debugging applications, but should not be used by deployed applications as it breaks the assembly loader in subtle ways.
Directories are separated by the platform path separator (colons on unix). Example:
/home/username/lib:/usr/local/mono/lib
Relative paths are resolved based on the launch-time current directory.

Alternative solutions to MONO_PATH include: installing libraries into the Global Assembly Cache (see gacutil(1)) or having the dependent libraries side-by-side with the main executable.
For a complete description of recommended practices for application deployment, see
http://www.mono-project.com/docs/getting-started/application-deployment/

MONO_SHARED_DIR
If set its the directory where the ".wapi" handle state is stored.
This is the directory where the Windows I/O Emulation layer stores its shared state data (files, events, mutexes, pipes). By default Mono will store the ".wapi" directory in the users's home directory.

MONO_SHARED_HOSTNAME
Uses the string value of this variable as a replacement for the host name when creating file names in the ".wapi" directory. This helps if the host name of your machine is likely to be changed when a mono application is running or if you have a .wapi directory shared among several different computers. Mono typically uses the hostname to create the files that are used to share state across multiple Mono processes. This is done to support home directories that might be shared over the network.

MONO_STRICT_IO_EMULATION
If set, extra checks are made during IO operations. Currently, this includes only advisory locks around file writes.

MONO_TLS_PROVIDER
This environment variable controls which TLS/SSL provider Mono will use. The options are usually determined by the operating system where Mono was compiled and the configuration options that were used for it.

default Uses the default TLS stack that the Mono runtime was configured with. Usually this is configured to use Apple's SSL stack on Apple platforms, and Boring SSL on other platforms.

apple Forces the use of the Apple SSL stack, only works on Apple platforms.

btls Forces the use of the BoringSSL stack. See https://opensource.google.com/projects/boringssl for more information about this stack.

legacy This is the old Mono stack, which only supports SSL and TLS up to version 1.0. It is deprecated and will be removed in the future.

MONO_TLS_SESSION_CACHE_TIMEOUT
The time, in seconds, that the SSL/TLS session cache will keep it's entry to avoid a new negotiation between the client and a server. Negotiation are very CPU intensive so an application-specific custom value may prove useful for small embedded systems.
The default is 180 seconds.

MONO_THREADS_PER_CPU
The minimum number of threads in the general threadpool will be MONO_THREADS_PER_CPU * number of CPUs. The default value for this variable is 1.

MONO_XMLSERIALIZER_THS
Controls the threshold for the XmlSerializer to produce a custom serializer for a given class instead of using the Reflection-based interpreter. The possible values are 'no' to disable the use of a

custom serializer or a number to indicate when the XmlSerializer should start serializing. The default value is 50, which means that the a custom serializer will be produced on the 50th use.

MONO_X509_REVOCATION_MODE
Sets the revocation mode used when validating a X509 certificate chain (https, ftps, smtps...). The default is 'nocheck', which performs no revocation check at all. The other possible values are 'offline', which performs CRL check (not implemented yet) and 'online' which uses OCSP and CRL to verify the revocation status (not implemented yet).

NO_PROXY
(Also **no_proxy**) If both **HTTP_PROXY** and **NO_PROXY** are set, **NO_PROXY** will be treated as a comma-separated list of "bypass" domains which will not be sent through the proxy. Domains in **NO_PROXY** may contain wildcards, as in "*.mono-project.com" or "build????.local". Not supported on Windows, Mac OS, iOS or Android.

ENVIRONMENT VARIABLES FOR DEBUGGING

MONO_ASPNET_NODELETE
If set to any value, temporary source files generated by ASP.NET support classes will not be removed. They will be kept in the user's temporary directory.

MONO_DEBUG
If set, enables some features of the runtime useful for debugging. This variable should contain a comma separated list of debugging options. Currently, the following options are supported:

align-small-structs
Enables small structs alignment to 4/8 bytes.

arm-use-fallback-tls
When this option is set on ARM, a fallback thread local store will be used instead of the default fast thread local storage primitives.

break-on-unverified
If this variable is set, when the Mono VM runs into a verification problem, instead of throwing an exception it will break into the debugger. This is useful when debugging verifier problems

casts This option can be used to get more detailed information from InvalidCast exceptions, it will provide information about the types involved.

check-pinvoke-callconv
This option causes the runtime to check for calling convention mismatches when using pinvoke, i.e. mixing cdecl/stdcall. It only works on windows. If a mismatch is detected, an ExecutionEngineException is thrown.

collect-pagefault-stats
Collects information about pagefaults. This is used internally to track the number of page faults produced to load metadata. To display this information you must use this option with "--stats" command line option.

debug-domain-unload
When this option is set, the runtime will invalidate the domain memory pool instead of destroying it.

disable_omit_fp
Disables a compiler optimization where the frame pointer is omitted from the stack. This optimization can interact badly with debuggers.

dont-free-domains
This is an Optimization for multi-AppDomain applications (most commonly ASP.NET applications). Due to internal limitations Mono, Mono by default does not use typed allocations on multi-appDomain applications as they could leak memory when a domain is unloaded. Although this is a fine default, for applications that use more than on AppDomain heavily (for example, ASP.NET applications) it is worth trading off the small leaks for the increased performance (additionally, since ASP.NET applications are not likely going to unload the application domains on production systems, it is worth using this feature).

dyn-runtime-invoke
Instructs the runtime to try to use a generic runtime-invoke wrapper instead of creating one invoke wrapper.

explicit-null-checks
Makes the JIT generate an explicit NULL check on variable dereferences instead of depending on the operating system to raise a SIGSEGV or another form of trap event when an invalid memory location is accessed.

gdb Equivalent to setting the **MONO_XDEBUG** variable, this emits symbols into a shared library as the code is JITed that can be loaded into GDB to inspect symbols.

gen-seq-points
Automatically generates sequence points where the IL stack is empty. These are places where the debugger can set a breakpoint.

no-compact-seq-points
Unless the option is used, the runtime generates sequence points data that maps native offsets to IL offsets. Sequence point data is used to display IL offset in stacktraces. Stacktraces with IL offsets can be symbolicated using mono-symbolicate tool.

handle-sigint
Captures the interrupt signal (Control-C) and displays a stack trace when pressed. Useful to find out where the program is executing at a given point. This only displays the stack trace of a single thread.

init-stacks
Instructs the runtime to initialize the stack with some known values (0x2a on x86-64) at the start of a method to assist in debuggin the JIT engine.

keep-delegates
This option will leak delegate trampolines that are no longer referenced as to present the user with more information about a delegate misuse. Basically a delegate instance might be created, passed to unmanaged code, and no references kept in managed code, which will garbage collect the code. With this option it is possible to track down the source of the problems.

no-gdb-backtrace
This option will disable the GDB backtrace emitted by the runtime after a SIGSEGV or SIGABRT in unmanaged code.

partial-sharing
When this option is set, the runtime can share generated code between generic types effectively reducing the amount of code generated.

reverse-pinvoke-exceptions
This option will cause mono to abort with a descriptive message when during stack unwinding after an exception it reaches a native stack frame. This happens when a managed delegate is passed to native code, and the managed delegate throws an exception. Mono will normally try to unwind the stack to the first (managed) exception handler, and it will skip any native stack frames in the process. This leads to undefined behaviour (since

mono doesn't know how to process native frames), leaks, and possibly crashes too.

single-imm-size
This guarantees that each time managed code is compiled the same instructions and registers are used, regardless of the size of used values.

soft-breakpoints
This option allows using single-steps and breakpoints in hardware where we cannot do it with signals.

suspend-on-native-crash
This option will suspend the program when a native crash occurs (SIGSEGV, SIGILL, ...). This is useful for debugging crashes which do not happen under gdb, since a live process contains more information than a core file.

suspend-on-sigsegv
Same as **suspend-on-native-crash**.

suspend-on-exception
This option will suspend the program when an exception occurs.

suspend-on-unhandled
This option will suspend the program when an unhandled exception occurs.

thread-dump-dir=DIR
Use DIR for storage thread dumps created by SIGQUIT.

verbose-gdb
Make gdb output on native crashes more verbose.

MONO_LOG_LEVEL
The logging level, possible values are 'error', 'critical', 'warning', 'message', 'info' and 'debug'. See the DEBUGGING section for more details.

MONO_LOG_MASK
Controls the domain of the Mono runtime that logging will apply to. If set, the log mask is changed to the set value. Possible values are "asm" (assembly loader), "type", "dll" (native library loader), "gc" (garbage collector), "cfg" (config file loader), "aot" (precompiler), "security" (e.g. Moonlight CoreCLR support) and "all". The default value is "all". Changing the mask value allows you to display only messages for a certain component. You can use multiple masks by comma separating them. For example to see config file messages and assembly loader messages set you mask to "asm,cfg".

MONO_LOG_DEST
Controls where trace log messages are written. If not set then the messages go to stdout. If set, the string either specifies a path to a file that will have messages appended to it, or the string "syslog" in which case the messages will be written to the system log. Under Windows, this is simulated by writing to a file called "mono.log". **MONO_LOG_HEADER** Controls whether trace log messages not directed to syslog have the id, timestamp, and pid as the prefix to the log message. To enable a header this environment variable need just be non-null.

MONO_TRACE
Used for runtime tracing of method calls. The format of the comma separated trace options is:

[-]M:method name
[-]N:namespace
[-]T:class name
[-]all
[-]program
disabled Trace output off upon start.

You can toggle trace output on/off sending a SIGUSR2 signal to the program.

MONO_TRACE_LISTENER
If set, enables the System.Diagnostics.DefaultTraceListener, which will print the output of the System.Diagnostics Trace and Debug classes. It can be set to a filename, and to Console.Out or Console.Error to display output to standard output or standard error, respectively. If it's set to Console.Out or Console.Error you can append an optional prefix that will be used when writing messages like this: Console.Error:MyProgramName. See the System.Diagnostics.DefaultTraceListener documentation for more information.

MONO_WCF_TRACE
This eases WCF diagnostics functionality by simply outputs all log messages from WCF engine to "stdout", "stderr" or any file passed to this environment variable. The log format is the same as usual diagnostic output.

MONO_XEXCEPTIONS
This throws an exception when a X11 error is encountered; by default a message is displayed but execution continues

MONO_XMLSERIALIZER_DEBUG
Set this value to 1 to prevent the serializer from removing the temporary files that are created for fast serialization; This might be useful when debugging.

MONO_XSYNC
This is used in the System.Windows.Forms implementation when running with the X11 backend. This is used to debug problems in Windows.Forms as it forces all of the commands send to X11 server to be done synchronously. The default mode of operation is asynchronous which makes it hard to isolate the root of certain problems.

MONO_XDEBUG
When the the MONO_XDEBUG env var is set, debugging info for JITted code is emitted into a shared library, loadable into gdb. This enables, for example, to see managed frame names on gdb backtraces.

MONO_VERBOSE_METHOD
Enables the maximum JIT verbosity for the specified method. This is very helpfull to diagnose a miscompilation problems of a specific method. This can be a semicolon-separated list of method names to match. If the name is simple, this applies to any method with that name, otherwise you can use a mono method description (see the section METHOD DESCRIPTIONS).

MONO_JIT_DUMP_METHOD
Enables sending of the JITs intermediate representation for a specified method to the IdealGraphVisualizer tool.

MONO_VERBOSE_HWCAP
If set, makes the JIT output information about detected CPU features (such as SSE, CMOV, FCMOV, etc) to stdout.

MONO_CONSERVATIVE_HWCAP
If set, the JIT will not perform any hardware capability detection. This may be useful to pinpoint the cause of JIT issues. This is the default when Mono is built as an AOT cross compiler, so that the generated code will run on most hardware.

VALGRIND

If you want to use Valgrind, you will find the file 'mono.supp' useful, it contains the suppressions for the GC which trigger incorrect warnings. Use it like this:

```
valgrind --suppressions=mono.supp mono ...
```

DTRACE

On some platforms, Mono can expose a set of DTrace probes (also known as user-land statically defined, USDT Probes).

They are defined in the file 'mono.d'.

ves-init-begin, ves-init-end
Begin and end of runtime initialization.

method-compile-begin, method-compile-end
Begin and end of method compilation. The probe arguments are class name, method name and signature, and in case of method-compile-end success or failure of compilation.

gc-begin, gc-end
Begin and end of Garbage Collection.

To verify the availability of the probes, run:
dtrace -P mono'$target' -l -c mono

PERMISSIONS

Mono's Ping implementation for detecting network reachability can create the ICMP packets itself without requiring the system ping command to do the work. If you want to enable this on Linux for non-root users, you need to give the Mono binary special permissions.

As root, run this command:
setcap cap_net_raw=+ep /usr/bin/mono

FILES

On Unix assemblies are loaded from the installation lib directory. If you set 'prefix' to /usr, the assemblies will be located in /usr/lib. On Windows, the assemblies are loaded from the directory where mono and mint live.

~/.mono/aot-cache
The directory for the ahead-of-time compiler demand creation assemblies are located.

/etc/mono/config, ~/.mono/config
Mono runtime configuration file. See the mono-config(5) manual page for more information.

~/.config/.mono/certs, /usr/share/.mono/certs
Contains Mono certificate stores for users / machine. See the certmgr(1) manual page for more information on managing certificate stores and the mozroots(1) page for information on how to import the Mozilla root certificates into the Mono certificate store.

~/.mono/assemblies/ASSEMBLY/ASSEMBLY.config
Files in this directory allow a user to customize the configuration for a given system assembly, the format is the one described in the mono-config(5) page.

~/.config/.mono/keypairs, /usr/share/.mono/keypairs
Contains Mono cryptographic keypairs for users / machine. They can be accessed by using a CspParameters object with DSACryptoServiceProvider and RSACryptoServiceProvider classes.

~/.config/.isolatedstorage, ~/.local/share/.isolatedstorage, /usr/share/.isolatedstorage
Contains Mono isolated storage for non-roaming users, roaming users and local machine. Isolated storage can be accessed using the classes from the System.IO.IsolatedStorage namespace.

<assembly>.config
Configuration information for individual assemblies is loaded by the runtime from side-by-side files with the .config files, see the http://www.mono-project.com/Config for more information.

Web.config, web.config
ASP.NET applications are configured through these files, the configuration is done on a per-directory basis. For more information on this subject see the http://www.mono-project.com/Config_system.web page.

MAILING LISTS

Mailing lists are listed at the http://www.mono-project.com/community/help/mailing-lists/

WEB SITE

http://www.mono-project.com

SEE ALSO

certmgr(1), **cert-sync**(1), **csharp**(1), **gacutil**(1), **mcs**(1), **monodis**(1), **mono-config**(5), **mono−profilers**(1), **mprof−report**(1), **pdb2mdb**(1), **xsp**(1), **mod_mono**(8)

For more information on AOT: http://www.mono-project.com/docs/advanced/aot/

For ASP.NET-related documentation, see the xsp(1) manual page