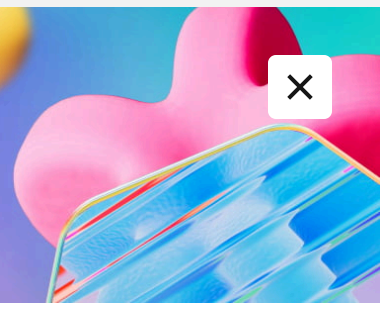

Dump collection and analysis utility (dotnet-dump)

Article • 03/14/2023 • 19 contributors

Feedback

In this article

This article applies to: ✔ `dotnet-dump` version 3.0.47001 and later versions

> **Note**
>
> `dotnet-dump` for macOS is only supported with .NET 5 and later versions.

Install

There are two ways to download and install `dotnet-dump`:

- **dotnet global tool:**

  To install the latest release version of the `dotnet-dump` NuGet package, use the dotnet tool install command:

  .NET CLI Copy

  ```
  dotnet tool install --global dotnet-dump
  ```

- **Direct download:**

  Download the tool executable that matches your platform:

Expand table

OS	Platform

Windows	x86	x64	Arm	Arm-x64
Linux	x64	Arm	Arm64	musl-x64	musl-Arm64

> **Note**
>
> To use `dotnet-dump` on an x86 app, you need a corresponding x86 version of the tool.

Synopsis

Console Copy

```
dotnet-dump [-h|--help] [--version] <command>
```

Description

The `dotnet-dump` global tool is a way to collect and analyze dumps on Windows, Linux, and macOS without any native debugger involved. This tool is important on platforms like Alpine Linux where a fully working `lldb` isn't available. The `dotnet-dump` tool allows you to run SOS commands to analyze crashes and the garbage collector (GC), but it isn't a native debugger so things like displaying native stack frames aren't supported.

Options

- **`--version`**

  Displays the version of the dotnet-dump utility.

- **`-h|--help`**

  Shows command-line help.

Commands

Expand table

Command
dotnet-dump collect
dotnet-dump analyze
dotnet-dump ps

dotnet-dump collect

Captures a dump from a process.

Synopsis

Console Copy

```
dotnet-dump collect [-h|--help] [-p|--process-id] [-n|--name] [--type] [-o|--out
```

Options

- `-h|--help`

  Shows command-line help.

- `-p|--process-id <PID>`

  Specifies the process ID number to collect a dump from.

- `-n|--name <name>`

  Specifies the name of the process to collect a dump from.

- `--type <Full|Heap|Mini>`

  Specifies the dump type, which determines the kinds of information that are collected from the process. There are three types:
  - `Full` - The largest dump containing all memory including the module images.
  - `Heap` - A large and relatively comprehensive dump containing module lists, thread lists, all stacks, exception information, handle information, and all memory except for mapped images.
  - `Mini` - A small dump containing module lists, thread lists, exception information, and all stacks.

  If not specified, `Full` is the default.

- `-o|--output <output_dump_path>`

  The full path and file name where the collected dump should be written. Ensure that the user under which the dotnet process is running has write permissions to the specified directory.

  If not specified:
  - Defaults to .*dump_YYYYMMDD_HHMMSS.dmp* on Windows.
  - Defaults to *./core_YYYYMMDD_HHMMSS* on Linux and macOS.

  YYYYMMDD is Year/Month/Day and HHMMSS is Hour/Minute/Second.

- `--diag`

  Enables dump collection diagnostic logging.

- `--crashreport`

  Enables crash report generation.

> **Note**
>
> On Linux and macOS, this command expects the target application and `dotnet-dump` to share the same `TMPDIR` environment variable. Otherwise, the command will time out.

> **Note**
>
> To collect a dump using `dotnet-dump`, it needs to be run as the same user as the user running target process or as root. Otherwise, the tool will fail to establish a connection with the target process.

dotnet-dump analyze

Starts an interactive shell to explore a dump. The shell accepts various SOS commands.

Synopsis

Console Copy

```
dotnet-dump analyze <dump_path> [-h|--help] [-c|--command]
```

Arguments

- `<dump_path>`

  Specifies the path to the dump file to analyze.

Options

- `-c|--command <debug_command>`

  Runs the command on start. Multiple instances of this parameter can be used in an invocation to chain commands. Commands will get run in the order that they are provided on the command line. If you want dotnet dump to exit after the commands, your last command should be 'exit'.

Analyze SOS commands

Expand table

Command	Function
`analyzeoom`	Displays the info of the last OOM that occurred on an allocation request to the GC heap.
`clrmodules`	Lists the managed modules in the process.
`clrstack`	Provides a stack trace of managed code only.
`clrthreads`	Lists the managed threads that are running.
`clru`	Displays an annotated disassembly of a managed method.
`d` or `readmemory`	Dumps memory contents.
`dbgout`	Enables/disables (`-off`) internal SOS logging.
`dso`	Displays all managed objects found within the bounds of the current stack.
`dumpalc`	Displays details about a collectible AssemblyLoadContext to which the specified object is loaded.
`dumparray`	Displays details about a managed array.
`dumpasync`	Displays info about async state machines on the garbage-collected heap.
`dumpassembly`	Displays details about an assembly.
`dumpclass`	Displays information about the `EEClass` structure at the specified address.
`dumpconcurrentdictionary`	Displays concurrent dictionary content.
`dumpconcurrentqueue`	Displays concurrent queue content.
`dumpdelegate`	Displays information about a delegate.
`dumpdomain`	Displays information about the all assemblies within all the AppDomains or the specified one.

`dumpgcdata`	Displays information about the GC data.
`dumpgen`	Displays heap content for the specified generation.
`dumpheap`	Displays info about the garbage-collected heap and collection statistics about objects.
`dumpil`	Displays the common intermediate language (CIL) that's associated with a managed method.
`dumplog`	Writes the contents of an in-memory stress log to the specified file.
`dumpmd`	Displays information about the `MethodDesc` structure at the specified address.
`dumpmodule`	Displays information about the module at the specified address.
`dumpmt`	Displays information about the method table at the specified address.
`dumpobj`	Displays info the object at the specified address.
`dumpruntimetypes`	Finds all System.RuntimeType objects in the GC heap and prints the type name and MethodTable they refer too.
`dumpsig`	Dumps the signature of a method or field specified by `<sigaddr> <moduleaddr>`.
`dumpsigelem`	Dumps a single element of a signature object.
`dumpstackobjects`	Displays all managed objects found within the bounds of the current stack.
`dumpvc`	Displays info about the fields of a value class.
`eeheap`	Displays info about process memory consumed by internal runtime data structures.
`eestack`	Runs `dumpstack` on all threads in the process.
`eeversion`	Displays information about the runtime and SOS versions.
`ehinfo`	Displays the exception handling blocks in a JIT-ed method.
`exit` or `quit`	Exits interactive mode.
`finalizequeue`	Displays all objects registered for finalization.
`findappdomain`	Attempts to resolve the AppDomain of a GC object.
`gchandles`	Displays statistics about garbage collector handles in the process.
`gcheapstat`	Displays statistics about garbage collector.
`gcinfo`	Displays the JIT GC encoding for a method.
`gcroot`	Displays info about references (or roots) to the object at the specified address.
`gcwhere`	Displays the location in the GC heap of the specified address.
`histclear`	Releases any resources used by the family of Hist commands.
`histinit`	Initializes the SOS structures from the stress log saved in the debuggee.
`histobj`	Examines all stress log relocation records and displays the chain of garbage collection relocations that may have led to the address passed in as an argument.
`histobjfind`	Displays all the log entries that reference the object at the specified address.

`histroot`	Displays information related to both promotions and relocations of the specified root.
`histstats`	Displays stress log stats.
`ip2md`	Displays the `MethodDesc` structure at the specified address in code that has been JIT-compiled.
`listnearobj`	Displays the object preceding and succeeding the specified address.
`logopen`	Enables console file logging.
`logclose`	Disables console file logging.
`logging`	Enables/disables internal SOS logging.
`lm` or `modules`	Displays the native modules in the process.
`name2ee`	Displays the `MethodTable` and `EEClass` structures for the specified type or method in the specified module.
`objsize`	Displays the size of the specified object.
`parallelstacks`	Displays the merged threads stack similarly to the Visual Studio 'Parallel Stacks' panel.
`pathto`	Displays the GC path from `<root>` to `<target>`.
`pe` or `printexception`	Displays and formats fields of any object derived from the Exception class at the specified address.
`r` or `registers`	Displays the thread's registers.
`runtimes`	Lists the runtimes in the target or changes the default runtime.
`setclrpath`	Sets the path to load coreclr dac/dbi files using `setclrpath <path>`.
`setsymbolserver`	Enables the symbol server support.
`sos`	Executes various coreclr debugging commands. Use the syntax `sos <command-name> <args>`. For more information, see 'soshelp'.
`soshelp` or `help`	Displays all available commands.
`soshelp <command>` or `help <command>`	Displays the specified command.
`syncblk`	Displays the SyncBlock holder info.
`taskstate`	Displays a Task state in a human readable format.
`threadpool`	Displays info about the runtime thread pool.
`threadpoolqueue`	Displays queued thread pool work items.
`threadstate`	Pretty prints the meaning of a threads state.
`threads <threadid>` or `setthread <threadid>`	Sets or displays the current thread ID for the SOS commands.
`timerinfo`	Displays information about running timers.
`token2ee`	Displays the MethodTable structure and MethodDesc structure for the specified token and module.
`traverseheap`	Writes out heap information to a file in a format understood by the CLR Profiler.
`verifyheap`	Checks the GC heap for signs of corruption.
`verifyobj`	Checks the object that is passed as an argument for signs of corruption.

> **Note**
>
> Additional details can be found in SOS Debugging Extension for .NET.

dotnet-dump ps

Lists the dotnet processes that dumps can be collected from. `dotnet-dump` version 6.0.320703 and later versions also display the command-line arguments that each process was started with, if available.

Synopsis

Console Copy

```
dotnet-dump ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-dump ps` command. The output you'll see is as follows. The command-line arguments, if any, are shown in `dotnet-dump` version 6.0.320703 and later.

Console Copy

```
> dotnet-dump ps

  21932 dotnet     C:\Program Files\dotnet\dotnet.exe   run --configuration Rele
  36656 dotnet     C:\Program Files\dotnet\dotnet.exe
```

Using `dotnet-dump`

The first step is to collect a dump. This step can be skipped if a core dump has already been generated. The operating system or the .NET Core runtime's built-in dump generation feature can each create core dumps.

Console Copy

```
$ dotnet-dump collect --process-id 1902
Writing minidump to file ./core_20190226_135837
Written 98983936 bytes (24166 pages) to core file
Complete
```

Now analyze the core dump with the `analyze` command:

Console Copy

```
$ dotnet-dump analyze ./core_20190226_135850
Loading core dump: ./core_20190226_135850
Ready to process analysis commands. Type 'help' to list available commands or 'h
Type 'quit' or 'exit' to exit the session.
>
```

This action brings up an interactive session that accepts commands like:

Console Copy

```
> clrstack
OS Thread Id: 0x573d (0)
```

```
    Child SP               IP Call Site
00007FFD28B42C58 00007fb22c1a8ed9 [HelperMethodFrame_PROTECTOBJ: 00007ffd28b42c5
00007FFD28B42DD0 00007FB1B1334F67 System.Reflection.RuntimeMethodInfo.Invoke(Sys
00007FFD28B42E20 00007FB1B18D33ED SymbolTestApp.Program.Foo4(System.String) [/hc
00007FFD28B42ED0 00007FB1B18D2FC4 SymbolTestApp.Program.Foo2(Int32, System.Strir
00007FFD28B42F00 00007FB1B18D2F5A SymbolTestApp.Program.Foo1(Int32, System.Strir
00007FFD28B42F30 00007FB1B18D168E SymbolTestApp.Program.Main(System.String[]) [/
00007FFD28B43210 00007fb22aa9cedf [GCFrame: 00007ffd28b43210]
00007FFD28B43610 00007fb22aa9cedf [GCFrame: 00007ffd28b43610]
```

To see an unhandled exception that killed your app:

Console Copy

```
> pe -lines
Exception object: 00007fb18c038590
Exception type:   System.Reflection.TargetInvocationException
Message:          Exception has been thrown by the target of an invocation.
InnerException:   System.Exception, Use !PrintException 00007FB18C038368 to see
StackTrace (generated):
SP               IP               Function
00007FFD28B42DD0 0000000000000000 System.Private.CoreLib.dll!System.RuntimeMetho
00007FFD28B42E20 00007FB1B18D33ED SymbolTestApp.dll!SymbolTestApp.Program.Foo4(S
00007FFD28B42ED0 00007FB1B18D2FC4 SymbolTestApp.dll!SymbolTestApp.Program.Foo2(I
00007FFD28B42F00 00007FB1B18D2F5A SymbolTestApp.dll!SymbolTestApp.Program.Foo1(I
00007FFD28B42F30 00007FB1B18D168E SymbolTestApp.dll!SymbolTestApp.Program.Main(S

StackTraceString: <none>
HResult: 80131604
```

Troubleshooting dump collection issues

Dump collection requires the process to be able to call `ptrace`. If you are facing issues collecting dumps, the environment you are running on may be configured to restrict such calls. See our Dumps: FAQ for troubleshooting tips and potential solutions to common issues.

See also

- Collecting and analyzing memory dumps blog
- Heap analysis tool (dotnet-gcdump)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback