NAME

cmake-configure-log − CMake Configure Log

Added in version 3.26.

INTRODUCTION

CMake writes a running log, known as the *configure log*, of certain events that occur during the Configure step. The configure log does*not* contain a log of all output, errors, or messages printed while configuring a project. It is a log of detailed information about specific events, such as toolchain inspection by *try_compile()*, meant for use in debugging the configuration of a build tree.

For human use, this version of CMake writes the configure log to the file:

```
${CMAKE_BINARY_DIR}/CMakeFiles/CMakeConfigureLog.yaml
```

However, the *location and name of the log file may change* in future versions of CMake. Tools that read the configure log should get its location using a *configureLog* query to the *cmake−file−api(7)*. See the*Lo g Versioning* section below for details.

LOG STRUCTURE

The configure log is designed to be both machine− and human−readable.

The log file is a YAML document stream containing zero or more YAML documents separated by document markers. Each document begins with a `---` document marker line, contains a single YAML mapping that logs events from one CMake "configure" step, and, if the configure step finished normally, ends with a **...** document marker line:

```
---
events:
  -
    kind: "try_compile-v1"
    # (other fields omitted)
  -
    kind: "try_compile-v1"
    # (other fields omitted)
...
```

A new document is appended to the log every time CMake configures the build tree and logs new events.

The keys of each document root mapping are:

events A YAML block sequence of nodes corresponding to events logged during one CMake "configure" step. Each event is a YAML node containing one of the *Event Kinds* documented below.

Log Versioning

Each of the *Event Kinds* is versioned independently. The set of keys an event's log entry provides is specific to its major version. When an event is logged, the latest version of its event kind that is known to the running version of CMake is always written to the log.

Tools reading the configure log must ignore event kinds and versions they do not understand:

- A future version of CMake may introduce a new event kind or version.
- If an existing build tree is re−configured with a different version of CMake, the log may contain different versions of the same event kind.

- If *cmake−file−api(7)* queries request one or more *configureLog* object versions, the log may contain multiple entries for the same event, each with a different version of its event kind.

IDEs should write a *cmake−file−api(7)* query requesting a specific *configureLog* object version, before running CMake, and then read the configure log only as described by the file−api reply.

Text Block Encoding

In order to make the log human−readable, text blocks are always represented using YAML literal block scalars (|). Since literal block scalars do not support escaping, backslashes and non−printable characters are encoded at the application layer:

- \\ encodes a backslash.
- **\xXX** encodes a byte using two hexadecimal digits, **XX**.

EVENT KINDS

Every event kind is represented by a YAML mapping of the form:

```
kind: "<kind>−v<major>"
backtrace:
  − "<file>:<line> (<function>)"
checks:
  − "Checking for something"
#...event−specific keys...
```

The keys common to all events are:

kind A string identifying the event kind and major version.

backtrace
A YAML block sequence reporting the call stack of CMake source locations at which the event occurred, from most−recent to least−recent. Each node is a string specifying one location formatted as **<file>:<line> (<function>)**.

checks An optional key that is present when the event occurred with at least one pending *message(CHECK_START)*. Its value is a YAML block sequence reporting the stack of pending checks, from most−recent to least−recent. Each node is a string containing a pending check message.

Additional mapping keys are specific to each (versioned) event kind, described below.

Event Kind message

The *message(CONFIGURE_LOG)* command logs **message** events.

There is only one **message** event major version, version 1.

message−v1 Event

A **message−v1** event is a YAML mapping:

```
kind: "message−v1"
backtrace:
  − "CMakeLists.txt:123 (message)"
checks:
  − "Checking for something"
message: |
  # ...
```

The keys specific to **message−v1** mappings are:

message
A YAML literal block scalar containing the message text, represented using our *Text Block Encoding*.

Event Kind try_compile
The *try_compile()* command logs **try_compile** events.

There is only one **try_compile** event major version, version 1.

try_compile−v1 Event
A **try_compile−v1** event is a YAML mapping:

```
kind: "try_compile-v1"
backtrace:
  - "CMakeLists.txt:123 (try_compile)"
checks:
  - "Checking for something"
description: "Explicit LOG_DESCRIPTION"
directories:
  source: "/path/to/.../TryCompile-01234"
  binary: "/path/to/.../TryCompile-01234"
cmakeVariables:
  SOME_VARIABLE: "Some Value"
buildResult:
  variable: "COMPILE_RESULT"
  cached: true
  stdout: |
    # ...
  exitCode: 0
```

The keys specific to **try_compile−v1** mappings are:

description
An optional key that is present when the **LOG_DESCRIPTION <text>** option was used. Its value is a string containing the description **<text>**.

directories
A mapping describing the directories associated with the compilation attempt. It has the following keys:

source String specifying the source directory of the *try_compile()* project.

binary String specifying the binary directory of the *try_compile()* project. For non−project invocations, this is often the same as the source directory.

cmakeVariables
An optional key that is present when CMake propagates variables into the test project, either automatically or due to the *CMAKE_TRY_COMPILE_PLATFORM_VARIABLES* variable. Its value is a mapping from variable names to their values.

buildResult
A mapping describing the result of compiling the test code. It has the following keys:

variable
A string specifying the name of the CMake variable storing the result of trying to build the test project.

cached A boolean indicating whether the above result **variable** is stored in the CMake cache.

stdout A YAML literal block scalar containing the output from building the test project, represented using our *Text Block Encoding*. This contains build output from both stdout and stderr.

exitCode
An integer specifying the build tool exit code from trying to build the test project.

Event Kind try_run
The *try_run()* command logs **try_run** events.

There is only one **try_run** event major version, version 1.

try_run−v1 Event
A **try_run−v1** event is a YAML mapping:

```
kind: "try_run-v1"
backtrace:
  - "CMakeLists.txt:456 (try_run)"
checks:
  - "Checking for something"
description: "Explicit LOG_DESCRIPTION"
directories:
  source: "/path/to/.../TryCompile-56789"
  binary: "/path/to/.../TryCompile-56789"
buildResult:
  variable: "COMPILE_RESULT"
  cached: true
  stdout: |
    # ...
  exitCode: 0
runResult:
  variable: "RUN_RESULT"
  cached: true
  stdout: |
    # ...
  stderr: |
    # ...
  exitCode: 0
```

The keys specific to **try_run−v1** mappings include those documented by the *try_compile−v1 event*, plus:

runResult
A mapping describing the result of running the test code. It has the following keys:

variable
A string specifying the name of the CMake variable storing the result of trying to run the test executable.

cached A boolean indicating whether the above result **variable** is stored in the CMake cache.

stdout An optional key that is present when the test project built successfully. Its value is a YAML literal block scalar containing output from running the test executable, represented using our *Text Block Encoding*.

If **RUN_OUTPUT_VARIABLE** was used, stdout and stderr are captured together, so this will contain both. Otherwise, this will contain only the stdout output.

stderr An optional key that is present when the test project built successfully and the **RUN_OUTPUT_VARIABLE** option was not used. Its value is a YAML literal block scalar containing output from running the test executable, represented using our *Text Block Encoding*.

If **RUN_OUTPUT_VARIABLE** was used, stdout and stderr are captured together in the **stdout** key, and this key will not be present. Otherwise, this will contain the stderr output.

exitCode
An optional key that is present when the test project built successfully. Its value is an integer specifying the exit code, or a string containing an error message, from trying to run the test executable.

COPYRIGHT