reproducible-builds.org

Home Documentation Talks & Resources Tools Tests Contribute Blog Who is involved? News

## Introduction

Definitions
Buy-in
Making plans
Test bench

## Achieve deterministic builds

**SOURCE_DATE_EPOCH**
Deterministic build systems
Volatile inputs can disappear
Stable order for inputs
Value initialization
Version information
Timestamps
Timezones
Locales
Archive metadata
Stable order for outputs
Randomness
Build path
System images

## Define a build environment

What's in a build environment?
Recording the build environment
Definition strategies
Proprietary operating systems

## Distribute the environment

Building from source
Virtual machine drivers
Formal definition

## Comparison protocol

Cryptographic checksums
Embedded signatures
Sharing certifications

# SOURCE_DATE_EPOCH

S_D_E (for short) is a standard that defines an environment variable `SOURCE_DATE_EPOCH` that distributions can set centrally, and have build tools consume this in order to produce reproducible output.

Before implementing this, you should scan through our checklist to see if you can avoid implementing it.

If you find that it's ideal for your use-case, please feel free to jump straight to our **published specification**.

## Proposal

Please read our SOURCE_DATE_EPOCH specification for details.

See Standard Environment Variables for more detailed discussion of the rationales behind this mechanism.

Below we also have [[#More_detailed_discussion more detailed discussion]] about this specific variable, as well as documentation on [[#history-and-alternatives history and alternative proposals]].

## Setting the variable

In Debian, this is automatically set to the same time as the latest entry in `debian/changelog`, i.e. the same as the output of `dpkg-parsechangelog -SDate`.

1. For packages using debhelper, versions >= 9.20151004 (Bug:791823) exports this variable during builds, so you probably don't need to change anything. One exception is if your `debian/rules` needs this variable in non-debhelper parts, in which case you can try (3) or (4).

2. For packages using CDBS, versions >= 0.4.131 (Bug:794241) exports this variable during builds, so no changes are needed.

3. With dpkg >= 1.18.8 (Bug:824572) you can either `include /usr/share/dpkg/pkg-info.mk` or `include /usr/share/dpkg/default.mk`.

4. If none of the above options are good (you should have a "very good reason") then package maintainers may set and export this variable manually in `debian/rules`:

```
export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog -STimestamp)
```

If you need/want to support dpkg versions earlier than 1.18.8:

```
export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog -SDate | date -f- +%s)
```

If you need/want to support dpkg versions earlier than 1.17.0:

```
export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog | grep -Po '^Date: \K.*' | date -f- +%s)
```

This snippet is believed to work on dpkg versions as far back as 2003.

## Reading the variable

We are persuading upstream tools to support this directly. You may help by writing patches for these tools; please add links to the bug reports here so we know, and to act as an example resource for future patch writers.

Pending:: gettext, qt4-x11 Complete::

- cmake (>= 3.8.0)
- debhelper (>= 9.20151004)
- docbook-utils (Debian >= 0.6.14-3.1, upstream TODO)
- doxygen (>= 1.8.12, Debian pending)
- epydoc (>= 3.0.1+dfsg-8, upstream pending)
- gcc (>= 7, Debian >= 5.3.1-17, Debian >= 6.1.1-1)
- ghostscript (Debian >= 9.16~dfsg-1, upstream unfortunately rejected due to additional constraints they have)
- groff (Debian >= 1.22.3-2, upstream pending)
- help2man (>= 1.47.1)
- libxslt (>= 1.1.29, Debian >= 1.1.28-3)
- man2html (Debian >= 1.6g-8, needs forwarding)
- mkdocs (>= 0.16, Debian pending)
- ocamldoc (>= 4.03.0, Debian >= 4.02.3-1)
- pydoctor (>= 0.5+git20151204)
- rpm upstream (>4.13 other relevant patches linked in there)
- sphinx (>= 1.4, Debian >= 1.3.1-3)
- texi2html (Debian >= 1.82+dfsg1-4, needs forwarding)
- texlive-bin (>= 2016.20160512.41045)
- txt2man (>= 1.5.7, Debian >= 1.5.6-4)
- rcc (Qt5 >= 5.13.0, Debian TODO)

Or search in all Debian sources: https://codesearch.debian.net/search?perpkg=1&q=SOURCE_DATE_EPOCH

## Python

```
import os
import time
import datetime

build_date = datetime.datetime.utcfromtimestamp(int(os.environ.get('SOURCE_DATE_EPOCH',
```

## Bash / POSIX shell

For GNU systems:

```
BUILD_DATE="$(date --utc --date="@${SOURCE_DATE_EPOCH:-$(date +%s)}" +%Y-%m-%d)"
```

If you need to also support BSD date as well:

```
DATE_FMT="%Y-%m-%d"
SOURCE_DATE_EPOCH="${SOURCE_DATE_EPOCH:-$(date +%s)}"
BUILD_DATE=$(date -u -d "@$SOURCE_DATE_EPOCH" "+$DATE_FMT" 2>/dev/null || date -u -r "$
```

## Perl

```
use POSIX qw(strftime);
my $date = strftime("%Y-%m-%d", gmtime($ENV{SOURCE_DATE_EPOCH} || time));
```

## Makefile

```
DATE_FMT = %Y-%m-%d
ifdef SOURCE_DATE_EPOCH
    BUILD_DATE ?= $(shell date -u -d "@$(SOURCE_DATE_EPOCH)" "+$(DATE_FMT)" 2>/dev/null
else
    BUILD_DATE ?= $(shell date "+$(DATE_FMT)")
endif
```

"The above will work with either GNU or BSD date, and fallback to ignore SOURCE_DATE_EPOCH if both fails."

## CMake

```
STRING(TIMESTAMP BUILD_DATE "%Y-%m-%d" UTC)
```

works with cmake >= 2.8.11 , but is only reproducible with cmake >= 3.8.0. If you do not have a modern cmake, but need reproducibility, you can use the less preferred variant:

```
if (DEFINED ENV{SOURCE_DATE_EPOCH})
  execute_process(
    COMMAND "date" "-u" "-d" "@$ENV{SOURCE_DATE_EPOCH}" "+%Y-%m-%d"
    OUTPUT_VARIABLE BUILD_DATE
    OUTPUT_STRIP_TRAILING_WHITESPACE)
else ()
  execute_process(
    COMMAND "date" "+%Y-%m-%d"
    OUTPUT_VARIABLE BUILD_DATE
    OUTPUT_STRIP_TRAILING_WHITESPACE)
endif ()
```

"The above will work only with GNU date. See POSIX shell example on how to support BSD date."

## C

```
#include <errno.h>
#include <limits.h>

struct tm *build_time;
time_t now;
char *source_date_epoch;
unsigned long long epoch;
char *endptr;

source_date_epoch = getenv("SOURCE_DATE_EPOCH");
if (source_date_epoch) {
        errno = 0;
        epoch = strtoull(source_date_epoch, &endptr, 10);
        if ((errno == ERANGE && (epoch == ULLONG_MAX || epoch == 0))
                        || (errno != 0 && epoch == 0)) {
                fprintf(stderr, "Environment variable $SOURCE_DATE_EPOCH: strtoull: %s\
                exit(EXIT_FAILURE);
        }
        if (endptr == source_date_epoch) {
                fprintf(stderr, "Environment variable $SOURCE_DATE_EPOCH: No digits wer
                exit(EXIT_FAILURE);
        }
        if (*endptr != '\0') {
                fprintf(stderr, "Environment variable $SOURCE_DATE_EPOCH: Trailing garb
                exit(EXIT_FAILURE);
        }
        if (epoch > ULONG_MAX) {
                fprintf(stderr, "Environment variable $SOURCE_DATE_EPOCH: value must be
                exit(EXIT_FAILURE);
        }
        now = epoch;
} else {
        now = time(NULL);
}
build_time = gmtime(&now);
```

## C++

```
#include <sstream>
#include <iostream>
#include <cstdlib>
#include <ctime>

  time_t now;
  char *source_date_epoch = std::getenv("SOURCE_DATE_EPOCH");
  if (source_date_epoch) {
    std::istringstream iss(source_date_epoch);
    iss >> now;
    if (iss.fail() || !iss.eof()) {
      std::cerr << "Error: Cannot parse SOURCE_DATE_EPOCH as integer\n";
      exit(27);
    }
  } else {
    now = std::time(NULL);
  }
```

## Go

```
import (
        "fmt"
        "os"
        "strconv"
        "time"
)

[...]

source_date_epoch := os.Getenv("SOURCE_DATE_EPOCH")
var build_date string
if source_date_epoch == "" {
        build_date = time.Now().UTC().Format(http.TimeFormat)
} else {
        sde, err := strconv.ParseInt(source_date_epoch, 10, 64)
        if err != nil {
                panic(fmt.Sprintf("Invalid SOURCE_DATE_EPOCH: %s", err))
        }
        build_date = time.Unix(sde, 0).UTC().Format(http.TimeFormat)
}
```

## git repository

to set SOURCE_DATE_EPOCH to the last modification of a git repository, in shell:

```
SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct)
```

## PHP

```
\date('Y', (int)\getenv('SOURCE_DATE_EPOCH') ?: \time())
```

## Emacs-Lisp

```
(current-time-string
  (when (getenv "SOURCE_DATE_EPOCH")
    (seconds-to-time
      (string-to-number
        (getenv "SOURCE_DATE_EPOCH")))))
```

## Javascript / node.js

```
var timestamp = new Date(process.env.SOURCE_DATE_EPOCH ? (process.env.SOURCE_DATE_EPOCH

// Alternatively, to ensure a fixed timezone:

var now = new Date();
if (process.env.SOURCE_DATE_EPOCH) {
  now = new Date((process.env.SOURCE_DATE_EPOCH * 1000) + (now.getTimezoneOffset() * 60
}
```

## Ruby

```
if ENV['SOURCE_DATE_EPOCH'].nil?
  now = Time.now
else
  now = Time.at(ENV['SOURCE_DATE_EPOCH'].to_i).gmtime
end
```

Note that Ruby's Datetime.strftime is locale-independent by default.

## Last-resort using faketime

"As a last resort to be avoided where possible" (e.g. if the upstream tool is too hard to patch, or too time-consuming for you right now to patch, or if they are being uncooperative or unresponsive), package maintainers may try something like the following:

`debian/strip-nondeterminism/a2x`:

```
#!/bin/sh
# Depends: faketime
# Eventually the upstream tool should support SOURCE_DATE_EPOCH internally.
test -n "$SOURCE_DATE_EPOCH" || { echo >&2 "$0: SOURCE_DATE_EPOCH not set"; exit 255; }
exec env NO_FAKE_STAT=1 faketime -f "$(TZ=UTC date -d "@$SOURCE_DATE_EPOCH" +'%Y-%m-%d
```

`debian/rules`:

```
export PATH := $(CURDIR)/debian/strip-nondeterminism:$(PATH)
```

`debian/control`:

```
Build-Depends: faketime
```

But please be aware that this does not work out of the box with pbuilder on Debian 7 Wheezy, see #778462 against faketime and #700591 against pbuilder (fixed in Jessie, but not Wheezy). Adding an according hook to `/etc/pbuilder/hook.d` which mounts `/run/shm` inside the chroot should suffice as local workaround, though.

TODO: document some other nicer options. Generally, all invocations of `date(1)` need to have a fixed `TZ` environment set.

NOTE: faketime BREAKS builds on some archs, for example hurd. See #778462 for details.

## More detailed discussion

Sometimes developers of build tools do not want to support `SOURCE_DATE_EPOCH`, or they will tweak the suggestion to something related but different. We really do think the best approach is to use `SOURCE_DATE_EPOCH` exactly as-is described above in our proposal, without any variation. Here we explain our reasoning versus the arguments we have encountered.

(See Standard Environment Variables]] for

| | |
|---|---|
| [[ReproducibleBuilds/StandardEnvironmentVariables#more-detailed-discussion | general arguments.) |

## "Lying about the time" / "violates language spec"

This argument arises when the tool processes some input which contains a static instruction to the effect of "get_current_time()". The input has a specification that defines what this means. The tool executes this instruction, then embeds the result in the output. It is argued that `SOURCE_DATE_EPOCH` would break these semantics and violate the specification.

In most cases, this argument places too much weight on the absoluteness of time. Regardless of what any specification says, the user can set their own system clock and achieve an effect similar to `SOURCE_DATE_EPOCH`. Note: Setting the system clock is not enough for ''reliable'' reproducible builds - we need `SOURCE_DATE_EPOCH` for long-running build processes that take varying amounts of time. If the tool was run near the end of the process, then merely setting the system clock would not make timestamps here reproducible. It is not up to the tool to judge whether the user is lying with their system clock, and likewise, it is not up to the tool to judge whether `SOURCE_DATE_EPOCH` is a "lie" or not.

For all intents and purposes, if the user has set `SOURCE_DATE_EPOCH` then they are taking a position that "this **is** the current time; please use this instead of whatever clock you normally use". Yes, the project developer wrote "get_current_time()" but I as the user, by setting `SOURCE_DATE_EPOCH`, am choosing to override this with my own idea of what time it is. Please execute the build as if the current time was `SOURCE_DATE_EPOCH`. FOSS software should generally prefer to respect end-users' wishes rather than developers' wishes. (And in practise, we haven't seen ''any'' instance where a project developer really really prefers "time of build" over "modtime of source".)

In conclusion, the tool may choose to ignore `SOURCE_DATE_EPOCH` for other reasons, but to judge that this is a ''lie'' is to disrespect the user's wishes. Furthermore, choosing to support this is unlikely to ''actually'' violate any specifications, since they generally don't define "current". This does not take into account, if the specification needs to interoperate consistently with other programs in a strong cryptographic ledger protocol where time values ''must'' be consistent across multiple entities. However this scenario is unlikely to apply, in the context of build tools where `SOURCE_DATE_EPOCH` would be set.)

Many tools allow the user to override the "current" date - e.g. `-D__TIME__=xxx`, `\\year=yyy`, etc. In these cases, it makes even less sense to ignore `SOURCE_DATE_EPOCH` for data integrity reasons - you ''already'' have a mechanism where the user can "lie" or "break the spec"; `SOURCE_DATE_EPOCH` would just be adding an extra mechanism that makes it easier to do this globally across all tools.

If for some reason you're still conflicted on suddenly changing the meaning of your "now()" function and desire another switch other than `SOURCE_DATE_EPOCH` being set or not, the `texlive` project came up with the variable `FORCE_SOURCE_DATE`; when that environment variable is set to `1` cases that wouldn't normally obey `SOURCE_DATE_EPOCH` will do. We **strongly discourage** the usage of such variable; `SOURCE_DATE_EPOCH` is meant to be already a flag forcing a particular timestamp to be used.

OTOH, one alternative we can agree with, that also avoids `SOURCE_DATE_EPOCH`, would be to translate the static instruction "get_current_time()" from the input format to ''an equivalent instruction'' in the output format, if the output format supports that.

## History and alternative proposals

1 and the surrounding messages describe the initial motivation behind this, including an evaluation of how different programming languages handle date formats.

At present, we do not have a proposal that includes anything resembling a "time zone". Developing such a standard requires consideration of various issues:

Intuitive and naive ways of handling human-readable dates, such as the POSIX date functions, are highly flawed and freely mix implicit not-well-defined calendars with absolute time. For example, they don't specify they mean the Gregorian calendar, and/or don't specify what to do with dates before when the Gregorian calendar was introduced, or use named time zones that require an up-to-date timezone database (e.g. with historical DST definitions) to parse properly.

Since this is meant to be a universal standard that all tools and distributions can support, we need to keep things simple and precise, so that different groups of people cannot accidentally interpret it in different ways. So it is probably unwise to try to standardise anything that resembles a named time zone, since that is very very complex.

One likely candidate would be something similar to the git internal timestamp format, see `man git-commit`:

It is , where is the number of seconds since the UNIX epoch. is a positive or negative offset from UTC. For example CET (which is 2 hours ahead UTC) is +0200.

We already have `SOURCE_DATE_EPOCH` so the time zone offset could be placed in `SOURCE_DATE_TZOFFSET` or something like that. But all of this needs further discussion.

Other non-standard variables that we haven't yet agreed upon, use at your own risk:

```
export SOURCE_DATE_TZOFFSET = $(shell dpkg-parsechangelog -SDate | tail -c6)
export SOURCE_DATE_RFC2822 = $(shell dpkg-parsechangelog -SDate)
export SOURCE_DATE_ISO8601 = $(shell python -c 'import time,email.utils,sys;t=email.uti
print(time.strftime("%Y-%m-%dT%H:%M:%S",t[:-1])+"{0:+03d}{1:02d}".format(t[-1]/3600,t[-
```

The ISO8601 code snippet is complex, in order to preserve the same timezone offset as in the RFC2822 form. If one is OK with stripping out this offset, i.e. forcing to UTC, then one can just use `date -u` instead. However, this then contains the same information as the unix timestamp, but the latter is generally easier to work with in nearly all programming languages.

BACK NEXT

“Reproducible builds” aim to provide a verifiable path from software source code to its compiled binary form.

@ReproBuilds

Content licensed under CC BY-SA 4.0.

Logos and trademarks belong to their respective owners.

Projects working on reproducible builds:

Arch Linux, Baserock, Bitcoin, coreboot, Debian, ElectroBSD, F-Droid, FreeBSD, Fedora, GNU Guix, Monero, NetBSD, NixOS, OpenEmbedded, openSUSE, OpenWrt, Qubes OS, Symfony, Tails, Tor Browser, Webconverger, Yocto Project.

Please tell us about yours!

Patches highly welcome through our Git repository (more info) or via our mailing list.