

# Package ‘pgenlib’

January 16, 2025

**Type** Package

**Title** PLINK 2 Binary (.pgen) Reader

**Version** 0.4.0

**Date** 2025-01-15

**Description** A thin wrapper over PLINK 2's core libraries which provides an R interface for reading .pgen files. A minimal .pvar loader is also included. Chang et al. (2015) \doi{10.1186/s13742-015-0047-8}.

**Encoding** UTF-8

**License** LGPL (>= 3)

**Copyright** This package includes sources of the libdeflate library owned by Eric Biggers, sources of the SIMDc library owned by Evan Nemerson, sources of the Zstd library owned by Meta Platforms, Inc., and sources of the pgenlib library owned by Christopher Chang.

**BugReports** <https://github.com/chrchang/plink-ng/issues>

**Imports** Rcpp (>= 1.0.1)

**LinkingTo** Rcpp

**RoxygenNote** 7.3.1

**NeedsCompilation** yes

**Author** Christopher Chang [aut, cre],  
Eric Biggers [ctb, cph] (Author of included libdeflate library),  
Yann Collet [ctb] (Author of included Zstd library),  
Meta Platforms, Inc. [cph] (Zstd library),  
Evan Nemerson [ctb, cph] (Author of included SIMDc library),  
Przemyslaw Skibinski [ctb] (Author of included Zstd library),  
Nick Terrell [ctb] (Author of included Zstd library)

**Maintainer** Christopher Chang <chrchang@alumni.caltech.edu>

**Repository** CRAN

**Date/Publication** 2025-01-15 23:30:03 UTC

## Contents

pgenlibr-package	2
AlleleCodeBuf	4
BoolBuf	4
Buf	5
ClosePgen	5
ClosePvar	6
GetAlleleCode	6
GetAlleleCt	7
GetMaxAlleleCt	7
GetRawSampleCt	8
GetVariantChrom	8
GetVariantCt	9
GetVariantId	9
GetVariantPos	10
GetVariantsById	10
HardcallPhasePresent	11
IntAlleleCodeBuf	11
IntBuf	12
NewPgen	12
NewPvar	13
Read	13
ReadAlleles	14
ReadHardcalls	14
ReadIntList	15
ReadList	16
VariantScores	16
<b>Index</b>	<b>17</b>

---

pgenlibr-package      *PLINK 2 Binary (.pgen) Reader*

---

### Description

A thin wrapper over PLINK 2's core libraries which provides an R interface for reading .pgen files. A minimal .pvar loader is also included.

### Details

NewPvar and NewPgen initialize the respective readers. Then, you can either iterate through one variant at a time (Read, ReadAlleles) or perform a multi-variant matrix load (ReadIntList, ReadList). When you're done, ClosePgen and ClosePvar free resources.

### Author(s)

Christopher Chang <chrchang@alumni.caltech.edu>

## References

Chang, C.C. and Chow, C.C. and Tellier, L.C.A.M. and Vattikuti, S. and Purcell, S.M. and Lee J.J. (2015) Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience* 4:7. doi:10.1186/s1374201500478.

## Examples

```
# This is modified from https://yosuketanigawa.com/posts/2020/09/PLINK2 .
library(pgenlibr)

# These files are subsetted from downloads available at
# https://www.cog-genomics.org/plink/2.0/resources#phase3_1kg .
# Note that, after downloading the original files, the .pgen file must be
# decompressed before use; but both pgenlibr and the PLINK 2 program can
# handle compressed .pvar files.
pvar_path <- system.file("extdata", "chr21_phase3_start.pvar.zst", package="pgenlibr")
pgen_path <- system.file("extdata", "chr21_phase3_start.pgen", package="pgenlibr")

pvar <- pgenlibr::NewPvar(pvar_path)
pgen <- pgenlibr::NewPgen(pgen_path, pvar=pvar)

# Check the number of variants and samples.
pgenlibr::GetVariantCt(pgen)
pgenlibr::GetRawSampleCt(pgen)

# Get the chromosome, position, and ID of the first variant.
GetVariantChrom(pvar, 1)
GetVariantPos(pvar, 1)
GetVariantId(pvar, 1)

# Read the 14th variant.
buf <- pgenlibr::Buf(pgen)
pgenlibr::Read(pgen, buf, 14)

# Get the index of the variant with ID "rs569225703".
var_id <- pgenlibr::GetVariantsById(pvar, "rs569225703")

# Get allele count.
pgenlibr::GetAlleleCt(pvar, var_id)

# It has three alleles, i.e. two ALT alleles.
# Read first-ALT-allele dosages for that variant.
pgenlibr::Read(pgen, buf, var_id)

# Read second-ALT-allele dosages.
pgenlibr::Read(pgen, buf, var_id, allele_num=3)

# Read a matrix with both variants. Note that, for the multiallelic variant,
# the dosages of both ALT alleles are summed here.
geno_mat <- pgenlibr::ReadList(pgen, c(14, var_id))

pgenlibr::ClosePgen(pgen)
```

```
pgenlibr::ClosePvar(pvar)
```

---

AlleleCodeBuf	<i>Returns an empty two-row numeric matrix that ReadAlleles() can load to.</i>
---------------	--

---

### Description

Returns an empty two-row numeric matrix that ReadAlleles() can load to.

### Usage

```
AlleleCodeBuf(pgen)
```

### Arguments

pgen            Object returned by NewPgen().

### Value

Numeric matrix with two rows, and appropriate number of columns for ReadAlleles().

---

BoolBuf	<i>Returns a bool buffer that ReadAlleles() can load phasing information to.</i>
---------	--

---

### Description

Returns a bool buffer that ReadAlleles() can load phasing information to.

### Usage

```
BoolBuf(pgen)
```

### Arguments

pgen            Object returned by NewPgen().

### Value

Logical vector with appropriate length for ReadAlleles().

---

Buf	<i>Returns a numeric buffer that Read() or ReadHardcalls() can load to.</i>
-----	---

---

**Description**

Returns a numeric buffer that Read() or ReadHardcalls() can load to.

**Usage**

Buf(pgen)

**Arguments**

pgen            Object returned by NewPgen().

**Value**

Numeric vector with appropriate length for Read() and ReadHardcalls().

---

ClosePgen	<i>Closes a pgen object, releasing resources.</i>
-----------	---

---

**Description**

Closes a pgen object, releasing resources.

**Usage**

ClosePgen(pgen)

**Arguments**

pgen            Object returned by NewPgen().

**Value**

No return value, called for side-effect.

---

ClosePvar	<i>Closes a pvar object, releasing memory.</i>
-----------	--

---

**Description**

Closes a pvar object, releasing memory.

**Usage**

```
ClosePvar(pvar)
```

**Arguments**

pvar	Object returned by NewPvar().
------	-------------------------------

**Value**

No return value, called for side-effect.

---

GetAlleleCode	<i>Look up an allele code.</i>
---------------	--------------------------------

---

**Description**

Look up an allele code.

**Usage**

```
GetAlleleCode(pvar, variant_num, allele_num)
```

**Arguments**

pvar	Object returned by NewPvar().
variant_num	Variant index (1-based).
allele_num	Allele index (1-based).

**Value**

The allele\_numth allele code for the variant\_numth variant. allele\_num=1 corresponds to the REF allele, allele\_num=2 corresponds to the first ALT allele, allele\_num=3 corresponds to the second ALT allele if it exists and errors out otherwise, etc.

---

GetAlleleCt	<i>Returns the effective number of alleles for a variant. Note that if no pvar was provided to the NewPgen() call, this function may return 2 even at multiallelic variants, since the .pgen may not store allele-count information.</i>
-------------	--

---

**Description**

Returns the effective number of alleles for a variant. Note that if no pvar was provided to the NewPgen() call, this function may return 2 even at multiallelic variants, since the .pgen may not store allele-count information.

**Usage**

```
GetAlleleCt(pvar_or_pgen, variant_num)
```

**Arguments**

pvar_or_pgen	Object returned by NewPvar() or NewPgen().
variant_num	Variant index (1-based).

**Value**

max(2, <number of alleles the variant\_numth variant is known to have>). Note that if no

---

GetMaxAlleleCt	<i>Returns the maximum GetAlleleCt() value across all variants in the file.</i>
----------------	---

---

**Description**

Returns the maximum GetAlleleCt() value across all variants in the file.

**Usage**

```
GetMaxAlleleCt(pvar_or_pgen)
```

**Arguments**

pvar_or_pgen	Object returned by NewPvar() or NewPgen().
--------------	--

**Value**

Maximum GetAlleleCt() value across all variants.

---

GetRawSampleCt      *Returns the number of samples in the file.*

---

**Description**

Returns the number of samples in the file.

**Usage**

GetRawSampleCt(pgen)

**Arguments**

pgen      Object returned by NewPgen().

**Value**

Number of samples.

---

GetVariantChrom      *Retrieve chromosome ID for given variant index.*

---

**Description**

Retrieve chromosome ID for given variant index.

**Usage**

GetVariantChrom(pvar, variant\_num)

**Arguments**

pvar      Object returned by NewPvar().

variant\_num      Variant index (1-based).

**Value**

Chromosome ID for the variant\_numth variant.



---

GetVariantCt	<i>Returns the number of variants in the file.</i>
--------------	--

---

**Description**

Returns the number of variants in the file.

**Usage**

```
GetVariantCt(pvar_or_pgen)
```

**Arguments**

pvar\_or\_pgen    Object returned by NewPvar() or NewPgen().

**Value**

Number of variants.

---

GetVariantId	<i>Convert variant index to variant ID string.</i>
--------------	--

---

**Description**

Convert variant index to variant ID string.

**Usage**

```
GetVariantId(pvar, variant_num)
```

**Arguments**

pvar            Object returned by NewPvar().  
variant\_num    Variant index (1-based).

**Value**

The variant\_numth variant ID string.

GetVariantPos      *Retrieve POS (base-pair coordinate on a chromosome) for given variant index.*

---

**Description**

Retrieve POS (base-pair coordinate on a chromosome) for given variant index.

**Usage**

```
GetVariantPos(pvar, variant_num)
```

**Arguments**

pvar                Object returned by NewPvar().  
variant\_num        Variant index (1-based).

**Value**

POS for the variant\_numth variant.

---

GetVariantsById      *Convert variant ID string to variant index(es).*

---

**Description**

Convert variant ID string to variant index(es).

**Usage**

```
GetVariantsById(pvar, id)
```

**Arguments**

pvar                Object returned by NewPvar().  
id                   Variant ID to look up.

**Value**

A list of all (1-based) variant indices with the given variant ID.

---

HardcallPhasePresent    *Returns whether explicitly phased hardcalls are present.*

---

**Description**

Returns whether explicitly phased hardcalls are present.

**Usage**

HardcallPhasePresent (pgen)

**Arguments**

pgen                    Object returned by NewPgen().

**Value**

TRUE if the file contains at least one phased heterozygous hardcall, FALSE otherwise.

---

IntAlleleCodeBuf        *Returns an empty two-row integer matrix that ReadAlleles() can load to.*

---

**Description**

Returns an empty two-row integer matrix that ReadAlleles() can load to.

**Usage**

IntAlleleCodeBuf (pgen)

**Arguments**

pgen                    Object returned by NewPgen().

**Value**

Integer matrix with two rows, and appropriate number of columns for ReadAlleles().

---

IntBuf	<i>Returns an integer buffer that ReadHardcalls() can load to.</i>
--------	--

---

**Description**

Returns an integer buffer that ReadHardcalls() can load to.

**Usage**

IntBuf(pgen)

**Arguments**

pgen	Object returned by NewPgen().
------	-------------------------------

**Value**

Integer vector with appropriate length for ReadHardcalls().

---

NewPgen	<i>Opens a .pgen or PLINK 1 .bed file.</i>
---------	--

---

**Description**

Opens a .pgen or PLINK 1 .bed file.

**Usage**

NewPgen(filename, pvar = NULL, raw\_sample\_ct = NULL, sample\_subset = NULL)

**Arguments**

filename	.pgen/.bed file path.
pvar	Object (see NewPvar()) corresponding to the .pgen's companion .pvar; technically optional, but necessary for some functionality. In particular, at multiallelic variants, all ALT alleles may be collapsed together when .pvar information is not available.
raw_sample_ct	Number of samples in file; required if it's a PLINK 1 .bed file, otherwise optional.
sample_subset	List of 1-based positions of samples to load; optional, all samples are loaded if this is not specified.

**Value**

A pgen object, which can be queried for genotype/dosage data.

---

NewPvar	<i>Loads variant positions, IDs, and allele codes from a .pvar or .bim file (which can be compressed with gzip or Zstd).</i>
---------	--

---

### Description

Loads variant positions, IDs, and allele codes from a .pvar or .bim file (which can be compressed with gzip or Zstd).

### Usage

```
NewPvar(filename, omit_chrom = FALSE, omit_pos = FALSE)
```

### Arguments

filename	.pvar/.bim file path.
omit_chrom	Whether to skip CHROM column.
omit_pos	Whether to skip POS column.

### Value

A pvar object, which can be queried for variant IDs and allele codes.

---

Read	<i>Loads the variant_numth variant, and then fills buf with numeric dosages in [0, 2] indicating the dosages of the first ALT (or user-specified) allele for each sample, with missing values represented by NA.</i>
------	--

---

### Description

This function treats the data as diploid; divide by 2 to obtain haploid dosages.

### Usage

```
Read(pgen, buf, variant_num, allele_num = 2L)
```

### Arguments

pgen	Object returned by NewPgen().
buf	Buffer returned by Buf().
variant_num	Variant index (1-based).
allele_num	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults 2.

**Value**

No return value, called for buf-filling side-effect.

---

ReadAlleles	<i>Loads the variant_numth variant, and then fills acbuf with integer allele codes, where each column of the buffer corresponds to a sample. An allele code of 0 corresponds to the REF allele, 1 to the first ALT, 2 to the second ALT, etc. Missing hardcalls are represented by a pair of NA codes.</i>
-------------	--

---

**Description**

This function treats the data as diploid. If it's really haploid, you may want to compare the two rows, and then treat samples where the allele codes differ as missing values.

**Usage**

```
ReadAlleles(pgen, acbuf, variant_num, phasepresent_buf = NULL)
```

**Arguments**

pgen	Object returned by NewPgen().
acbuf	Buffer returned by AlleleCodeBuf() or IntAlleleCodeBuf().
variant_num	Variant index (1-based).
phasepresent_buf	Buffer returned by BoolBuf(). Optional; if provided, elements are set to true when the sample has known phase. Most of these values will be TRUE even when the raw data is unphased, because homozygous genotypes always have known phase. (Missing genotypes are considered to have unknown phase.)

**Value**

No return value, called for acbuf-filling side-effect.

---

ReadHardcalls	<i>Loads the variant_numth variant, and then fills buf with {0, 1, 2, NA} values indicating the number of copies of the first ALT (or user-specified) allele each sample has.</i>
---------------	---

---

**Description**

This function treats the data as diploid; you can divide by 2, and then treat 0.5 as NA, if it's actually haploid.

**Usage**

```
ReadHardcalls(pgen, buf, variant_num, allele_num = 2L)
```

**Arguments**

`pgen` Object returned by `NewPgen()`.

`buf` Buffer returned by `Buf()` or `IntBuf()`.

`variant_num` Variant index (1-based).

`allele_num` Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults 2.

**Value**

No return value, called for buf-filling side-effect.

---

ReadIntList	<i>Load hardcalls for multiple variants as an integer matrix.</i>
-------------	---

---

**Description**

This function treats the data as diploid; you can divide by 2, and then treat 0.5 as NA, if it's actually haploid.

**Usage**

```
ReadIntList(pgen, variant_subset)
```

**Arguments**

`pgen` Object returned by `NewPgen()`.

`variant_subset` Integer vector containing 1-based indexes of variants to load.

**Value**

Integer matrix, where rows correspond to samples, columns correspond to `variant_subset`, and values are in `{0, 1, 2, NA}` indicating the number of hardcall ALT allele copies. For multiallelic variants, all ALT alleles are combined.

---

ReadList	<i>Load dosages for multiple variants as a numeric matrix.</i>
----------	--

---

**Description**

This function treats the data as diploid; divide by 2 to obtain haploid dosages.

**Usage**

```
ReadList(pgen, variant_subset, meanimpute = FALSE)
```

**Arguments**

pgen	Object returned by NewPgen().
variant_subset	Integer vector containing 1-based indexes of variants to load.
meanimpute	Optional; if true, missing values are mean-imputed instead of being represented by NA.

**Value**

Numeric matrix, where rows correspond to samples, and columns correspond to variant\_subset. Values are in [0, 2] indicating ALT allele dosages, or NA for missing dosages. For multiallelic variants, all ALT alleles are combined.

---

VariantScores	<i>Compute variant scores.</i>
---------------	--------------------------------

---

**Description**

This function treats the data as diploid; divide by 2 to obtain scores based on a haploid dosage matrix.

**Usage**

```
VariantScores(pgen, weights, variant_subset = NULL)
```

**Arguments**

pgen	Object returned by NewPgen().
weights	Sample weights.
variant_subset	Integer vector containing 1-based indexes of variants to include in the dosage matrix. Optional; by default, all variants are included.

**Value**

Numeric vector, containing product of sample-weight vector and the specified subset of the dosage matrix.



# Index

## \* package

[pgenlibr-package](#), 2

[AlleleCodeBuf](#), 4

[BoolBuf](#), 4

[Buf](#), 5

[ClosePgen](#), 5

[ClosePvar](#), 6

[GetAlleleCode](#), 6

[GetAlleleCt](#), 7

[GetMaxAlleleCt](#), 7

[GetRawSampleCt](#), 8

[GetVariantChrom](#), 8

[GetVariantCt](#), 9

[GetVariantId](#), 9

[GetVariantPos](#), 10

[GetVariantsById](#), 10

[HardcallPhasePresent](#), 11

[IntAlleleCodeBuf](#), 11

[IntBuf](#), 12

[NewPgen](#), 12

[NewPvar](#), 13

[pgenlibr \(pgenlibr-package\)](#), 2

[pgenlibr-package](#), 2

[Read](#), 13

[ReadAlleles](#), 14

[ReadHardcalls](#), 14

[ReadIntList](#), 15

[ReadList](#), 16

[VariantScores](#), 16