

# Package ‘rjuliabugs’

September 9, 2025

**Type** Package

**Title** Interface to 'JuliaBUGS.jl' from R

**Version** 0.1.0

**Description** Provides an R interface to the 'JuliaBUGS.jl' package (<https://github.com/TuringLang/JuliaBUGS.jl>) for Bayesian inference using the BUGS modeling language. Allows R users to run models in Julia and return results as familiar R objects. Visualization and posterior analysis are supported via the 'bayesplot' and 'posterior' packages.

**License** MIT + file LICENSE

**Encoding** UTF-8

**SystemRequirements** Julia ( $\geq 1.10.8$ ) with the following packages installed: 'JuliaBUGS.jl', 'LogDensityProblemsAD.jl', 'ReverseDiff.jl', 'AdvancedHMC.jl', 'AbstractMCMC.jl', 'LogDensityProblems.jl', 'MCMCChains.jl', 'RCall.jl', 'Suppressor.jl'.

**Imports** coda, JuliaCall, posterior

**Suggests** bayesplot, knitr, rmarkdown, gridExtra, rstan, nimble

**URL** <https://mateusmaiaads.github.io/rjuliabugs/>

**BugReports** <https://github.com/MateusMaiaDS/rjuliabugs/issues>

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Mateus Maia [aut, cre],  
Xianda Sun [aut],  
Robert Goudie [aut]

**Maintainer** Mateus Maia <[mateus.maiamarques@glasgow.ac.uk](mailto:mateus.maiamarques@glasgow.ac.uk)>

**Repository** CRAN

**Date/Publication** 2025-09-09 14:40:02 UTC

## Contents

as_draws	2
as_mcmc	3
as_rvar	4
bugs2juliaBUGS	5
convert_numeric_types	6
delete_julia_obj	7
extract	7
juliaBUGS	8
julia_assign_int	11
load_rjuliabugs	12
save_rjuliabugs	13
setup_juliaBUGS	13
summary.rjuliabugs	15
wrap_model_to_juliaBUGS	16
<b>Index</b>	<b>17</b>

---

as_draws	<i>Convert to posterior::draws format</i>
----------	---

---

### Description

Generic function to convert an object to a `posterior::draws` representation (e.g., `draws_array`, `draws_matrix`, or other formats).

### Usage

```
as_draws(x, ...)

## S3 method for class 'rjuliabugs'
as_draws(x, ...)

## S3 method for class 'array'
as_draws(x, ...)
```

### Arguments

`x` An object to convert (e.g., a `rjuliabugs` object or a 3D array).  
`...` Further arguments passed to specific methods.

### Value

A draws object (e.g., `draws_array`) or a modified `rjuliabugs` object.

An object of class "rjuliabugs" (a named list) with the following elements:

**params** Posterior samples converted to `posterior::draws_array`.

- name** The name of the Julia sampler object (unchanged).
- sampler** The sampler object returned by `AbstractMCMC.sample` in Julia (unchanged).
- n\_threads** Number of Julia threads detected (unchanged).
- mcmc** MCMC configuration parameters; `posterior_type` is updated to "draws".
- control** Control options passed to and used by the sampler (unchanged).

An object of class `draws_array` (from the **posterior** package). The 3D array (iterations × chains × parameters) is converted to a `draws_array`, preserving the chain structure and parameter names.

### See Also

[posterior::as\\_draws\(\)](#)

---

as\_mcmc

*Convert to `coda::mcmc` or `mcmc.list` format*

---

### Description

Generic function to convert Markov Chain Monte Carlo (MCMC) output to `coda::mcmc` or `coda::mcmc.list` format.

### Usage

```
as_mcmc(x, ...)
```

```
## S3 method for class 'rjuliabugs'
```

```
as_mcmc(x, ...)
```

```
## S3 method for class 'array'
```

```
as_mcmc(x, ...)
```

### Arguments

- `x` An object to convert (e.g., a `rjuliabugs` object or a 3D array).
- `...` Further arguments passed to specific methods.

### Value

An object of class `mcmc`, `mcmc.list`, or a `rjuliabugs` object with updated params.

An object of class "rjuliabugs" (a named list) with the following elements:

- params** Posterior samples converted to `coda::mcmc` if a single chain, or `coda::mcmc.list` if multiple chains.
- name** The name of the Julia sampler object (unchanged).
- sampler** The sampler object returned by `AbstractMCMC.sample` in Julia (unchanged).
- n\_threads** Number of Julia threads detected (unchanged).

**mcmc** MCMC configuration parameters; `posterior_type` is updated to "mcmc".

**control** Control options passed to and used by the sampler (unchanged).

Returns posterior samples converted to `coda::mcmc` if the array has one chain, or `coda::mcmc.list` if multiple chains. Input must be a 3D array (iterations  $\times$  chains  $\times$  parameters). Each column corresponds to a parameter, and each row corresponds to an iteration.

### See Also

[coda::as.mcmc\(\)](#), [coda::mcmc\(\)](#)

---

as_rvar	<i>Convert to posterior::rvar format</i>
---------	--

---

### Description

Generic function to convert an object to a `posterior::rvar` representation. This is typically used to convert Markov Chain Monte Carlo (MCMC) output into a more flexible and vectorized format.

### Usage

```
as_rvar(x, ...)

## S3 method for class 'rjuliabugs'
as_rvar(x, ...)

## S3 method for class 'array'
as_rvar(x, n_mcmc = NULL, ...)
```

### Arguments

**x** An object to convert (e.g., a `rjuliabugs` object or a 3D numeric array).

**...** Further arguments passed to specific methods.

**n\_mcmc** (For arrays only) Number of Markov Chain Monte Carlo (MCMC) chains. Required if `x` is an array.

### Value

An object of class `rvar`, or an updated `rjuliabugs` object with `params` converted to `rvar`.

An object of class "rjuliabugs" (a named list) with the following elements:

**params** Posterior samples, converted to the requested format: `rvar` for `as_rvar`, `mcmc/mcmc.list` for `as_mcmc`, `draws_array` for `as_draws`.

**name** The name of the Julia sampler object.

**sampler** The sampler object returned by `AbstractMCMC.sample` in Julia.

**n\_threads** Number of Julia threads detected.

**mcmc** A list of MCMC configuration parameters, now including `posterior_type` indicating the format of params.

**control** Control options passed to and used by the sampler.

An object of class `rvar` (from the **posterior** package). The input 3D array (iterations  $\times$  chains  $\times$  parameters) is converted into a posterior `rvar` object, where each parameter is represented as a random variable across iterations and chains.

### See Also

[posterior::rvar\(\)](#)

---

bugs2juliaBUGS	<i>Convert Bayesian Updating for Gibbs Sampling (BUGS) Model to Julia's '@bugs' Macro Format</i>
----------------	--

---

### Description

This function formats a Bayesian Updating for Gibbs Sampling (BUGS) model string into Julia's '@bugs("..."...)", `convert_var_name, true`' macro syntax, used to run BUGS models in Julia. By default, R-style variable names (e.g., 'a.b.c') are converted to Julia-style ('a\_b\_c'). You can disable this behavior by setting '`convert_var_name = FALSE`'.

### Usage

```
bugs2juliaBUGS(model_code, convert_var_name = TRUE)
```

### Arguments

`model_code` A character string containing the BUGS model code.

`convert_var_name` Logical; if TRUE (default), R-style variable names (e.g., a.b.c) are converted to Julia-style (a\_b\_c). Set to FALSE to preserve the original names.

### Value

A character string representing a valid Julia '@bugs' macro call. This string can be evaluated in Julia to define the BUGS model with optional variable name conversion.

### Examples

```
## Not run:
model <- "
  for i in 1:N
    y[i] ~ dnorm(mu, tau)
  end
  mu ~ dnorm(0.0, 1.0E-6)
  tau ~ dgamma(0.001, 0.001)
```

```
"  
bugs2juliaBUGS(model)  
bugs2juliaBUGS(model, convert_var_name = FALSE)  
  
## End(Not run)
```

---

convert\_numeric\_types *Convert Numeric Elements in a List to Integer or Float*

---

## Description

This function takes a list of numeric vectors and returns a new list where each numeric element is automatically converted to either an integer (if it is a whole number) or kept as a float (numeric). It also preserves the original names of vector elements, if any.

## Usage

```
convert_numeric_types(data)
```

## Arguments

data            A list of numeric vectors.

## Value

A list of the same structure where each numeric element is coerced to the appropriate type: integers for whole numbers, and floats otherwise. Names are preserved.

## Examples

```
## Not run:  
input_list <- list(  
  a = c(x = 1.0, y = 2.5, z = 3.0),  
  b = c(foo = 4.0, bar = 5.1),  
  c = c(6, 7, 8)  
)  
convert_numeric_types(input_list)  
  
## End(Not run)
```

---

delete\_julia\_obj      *Delete an object from the Julia Main environment*

---

### Description

This function removes a variable or object from the Julia Main module using JuliaCall. It is useful for cleaning up or resetting objects defined in the Julia environment from R.

### Usage

```
delete_julia_obj(obj_name)
```

### Arguments

obj_name	A character string specifying the name of the Julia object to be deleted. This should correspond to a variable or symbol previously defined in the Julia Main module.
----------	---

### Value

No return value, called for side effects.

### Examples

```
## Not run:  
JuliaCall::julia_command("x = 10") # Define a Julia variable  
delete_julia_obj("x")             # Delete it  
  
## End(Not run)
```

---

extract      *Extract Posterior Samples from an 'rjuliabugs' S3 Object*

---

### Description

Extracts posterior samples for specified parameters from a fitted 'rjuliabugs' object. Output can be returned in several formats depending on downstream analysis requirements.

### Usage

```
extract(rjuliabugs, pars = NULL, type = "array", include = TRUE)
```

**Arguments**

<code>rjuliabugs</code>	An S3 object of class <code>'rjuliabugs'</code> , typically returned by a call to the <code>'juliaBUGS()'</code> function. Must contain a <code>'params'</code> 3D array and <code>'mcmc'</code> list with fields <code>'params_to_save'</code> and <code>'n_chain'</code> .
<code>pars</code>	Character vector of parameter names to extract. If <code>'NULL'</code> , defaults to <code>'rjuliabugs\$mcmc\$params_to_save'</code> .
<code>type</code>	Character string indicating output type: one of <code>"array"</code> (default), <code>"rvar"</code> , <code>"mcmc"</code> , or <code>"draws"</code> .
<code>include</code>	Logical; if <code>'TRUE'</code> , extract only <code>'pars'</code> ; if <code>'FALSE'</code> , exclude <code>'pars'</code> .

**Value**

The posterior samples in the specified format:

- a 3D `'array'` [iterations  $\times$  chains  $\times$  parameters],
- a `'posterior::rvar'` object,
- a `'coda::mcmc'` or `'coda::mcmc.list'`,
- or a `'posterior::draws_array'/'draws_list'`.

---

juliaBUGS

---

*Run a Julia HMC Sampler for a BUGS-like Probabilistic Model*


---

**Description**

Executes a Hamiltonian Monte Carlo (HMC) sampler in Julia from R, using a model specified in Julia or in BUGS syntax. It compiles the model, converts data, sets sampler parameters, and returns posterior samples in various formats. The setup for the HMC sampler uses Not-U-Turn Sampler (NUTS) with the target acceptance probability  $\delta=0.8$  for step size adaptation.

**Usage**

```
juliaBUGS(
  data,
  model_def,
  params_to_save,
  initializations = NULL,
  name = "sampler_juliaBUGS",
  n_iter = 2000,
  n_warmup = floor(n_iter/2),
  n_discard = n_warmup,
  n_thin = 1,
  n_chain = 1,
  use_parallel = TRUE,
  posterior_type = "array",
  force_setup_juliaBUGS = FALSE,
```



```

    control = NULL,
    progress = TRUE,
    verbose = TRUE,
    ...
)

```

## Arguments

<code>data</code>	A named list of numeric values (integer or double). All elements must be named.
<code>model_def</code>	A character string with the model definition, either in Julia-compatible format or BUGS syntax.
<code>params_to_save</code>	Character vector with the names of model parameters to extract from the sampler output.
<code>initializations</code>	A named list of parameter names for which you may wish to set corresponding initial values for the sampler. The default is <code>NULL</code> , which means no default initial values are used.
<code>name</code>	Character. Name for the sampler object created in Julia (must be a valid Julia variable name).
<code>n_iter</code>	Integer. Total number of MCMC iterations. Default is 2000.
<code>n_warmup</code>	Integer. Number of iterations used warm-up or tuning (e.g., adaption steps in NUTS). Default is <code>floor(n_iter / 2)</code> .
<code>n_discard</code>	Integer. Number of initial samples to be completely discarded. Default is <code>n_warmup</code> , i.e: discard all the iterations used as adaptation steps.
<code>n_thin</code>	Integer. Thinning interval. Default is 1 (no thinning).
<code>n_chain</code>	Integer. Number of MCMC chains. Default is 1.
<code>use_parallel</code>	Logical. Whether to use <code>AbstractMCMC.MCMCThreads()</code> for parallel sampling. Default is <code>TRUE</code> .
<code>posterior_type</code>	Character. Format of the posterior samples. One of "array", "rvar", "mcmc", or "draws". Default is "array".
<code>force_setup_juliaBUGS</code>	Logical. If <code>TRUE</code> , forces reinitialization of the Julia environment via <code>setup_juliaBUGS()</code> . Default is <code>FALSE</code> .
<code>control</code>	Optional list of control parameters. Supported entries: <ul style="list-style-type: none"> <li><code>data_convert_int</code> Logical. If <code>TRUE</code>, coerces numeric values to integers when possible. Default is <code>TRUE</code>.</li> <li><code>convert_var_name</code> Logical. If <code>TRUE</code>, automatically renames variables in the Bayesian Updating for Gibbs Sampling (BUGS) model. Default is <code>FALSE</code>.</li> <li><code>julia_model</code> Logical. If <code>TRUE</code>, assumes the model is already in Julia format used by the models under the <code>Turing.jl</code> approach, not a Bayesian Updating for Gibbs Sampling (BUGS) model. Default is <code>FALSE</code>.</li> </ul>
<code>progress</code>	Logical. If <code>TRUE</code> , a progress bar for the sampler is displayed; if <code>FALSE</code> , no progress bar is shown. The default is <code>TRUE</code> . However, when the function is run interactively inside an RStudio session, progress is automatically overridden

	to FALSE, which suppresses the progress output from <code>AbstractMCMC.sample</code> . For a complete progress bar display with <code>rjuliabugs</code> , as the one showed when running Julia code, we recommend running the code from a terminal outside of RStudio.
<code>verbose</code>	Logical. If FALSE will omit any message from the function to indicate the sampler/setup progress
<code>...</code>	Additional arguments passed to <code>setup_juliaBUGS()</code> .

### Details

This function relies on Julia packages `LogDensityProblems`, `AdvancedHMC`, and `AbstractMCMC`. Gradients are computed via `ReverseDiff`. The model is compiled before sampling.

The `posterior_type` argument determines the return format:

- `"array"`: 3D numeric array (iterations  $\times$  chains  $\times$  parameters).
- `"rvar"`: `posterior::rvar` object.
- `"mcmc"`: `coda::mcmc` (single chain) or `mcmc.list` (multiple chains).
- `"draws"`: `posterior::draws_array`.

### Value

An object of class `"rjuliabugs"` (a named list) with the following elements:

**params** Posterior samples, in the format specified by `posterior_type` (`"array"`, `"rvar"`, `"mcmc"`, or `"draws"`).

**name** A character string giving the name of the Julia sampler object.

**sampler** The sampler object returned by `AbstractMCMC.sample` in Julia.

**n\_threads** An integer giving the number of Julia threads detected.

**mcmc** A list of MCMC configuration parameters used in the run.

**control** The list of control options passed to and used by the sampler.

### Note

You must call `setup_juliaBUGS()` at least once before using this function. If parallel sampling is requested but only one Julia thread is available, a warning is issued and sampling will run serially.

### Examples

```
## Not run:
model_def <- "model = @model ... end"
data <- list(N = 10, x = rnorm(10))
result <- juliaBUGS(
  data = data,
  model_def = model_def,
  params_to_save = c("mu"),
  name = "my_sampler"
)
```

```
## End(Not run)
```

---

julia\_assign\_int      *Assign an Integer Value to a Julia Variable*

---

### Description

This function wraps ‘JuliaCall::julia\_assign’ to assign a value from R to a Julia variable, and then explicitly casts the variable to the ‘Integer’ type in Julia.

### Usage

```
julia_assign_int(x, value)
```

### Arguments

x	A character string. The name of the Julia variable to assign the value to.
value	The R object to assign to the Julia variable. This will be passed to Julia.

### Details

The function first assigns the value from R to a variable named ‘x’ in the Julia session using ‘JuliaCall::julia\_assign’. It then forces Julia to cast the variable to the ‘Integer’ type using Julia’s ‘Integer()’ constructor.

Note: This function assumes that the ‘value’ provided is compatible with Julia’s ‘Integer’ type. If it is not, an error will be thrown by Julia.

### Value

Invisibly returns ‘NULL’. The main effect is side-effects in the Julia session.

### Examples

```
## Not run:  
julia_assign_int("x", 3.5) # Will be cast to 3L in Julia  
JuliaCall::julia_eval("x") # Returns 3 (as Integer in Julia)  
  
## End(Not run)
```

---

`load_rjuliabugs`*Load an rjuliabugs Object and Restore the Julia State*

---

### Description

Loads an object of class `rjuliabugs` from an `.rds` file and restores the corresponding Julia sampler object using Julia's `Serialization.deserialize`. The path linking the `Chains` object from Julia is defined in the when the function `save_rjuliabugs()` is called.

### Usage

```
load_rjuliabugs(file)
```

### Arguments

`file`                    A character string giving the path to the `.rds` file.

### Details

If the original sampler name (`name`) already exists in the active Julia session, a new unique name is generated to avoid overwriting it. A warning will be issued to indicate that the name has changed.

The `.rds` file must contain a valid `rjuliabugs` object with both the `name` and `chains_file` fields defined. The function checks if the sampler name is already defined in Julia. If so, a unique name is generated using [check\\_sampler\\_is\\_defined](#), and the Julia object is loaded under that name.

### Value

An object of class `rjuliabugs`, with the Julia sampler object loaded into the current session. If the name was changed to avoid conflict, the returned object reflects the updated name.

### See Also

[check\\_sampler\\_is\\_defined](#)

### Examples

```
## Not run:
model <- load_rjuliabugs("my_model.rds")
# model$name now contains the (possibly updated) name used in Julia

## End(Not run)
```

---

save_rjuliabugs	<i>Save an rjuliabugs Object and Its Julia State</i>
-----------------	--

---

**Description**

Serializes the Julia object contained in an `rjuliabugs` object and saves the entire object as an `.rds` file. The Julia object is saved separately using Julia's `Serialization.serialize`. The file path can be passed manually, or retrieved from the `chains_file` slot in the object.

**Usage**

```
save_rjuliabugs(rjuliabugs_model, file, chains_file = NULL)
```

**Arguments**

<code>rjuliabugs_model</code>	An object of class <code>rjuliabugs</code> , containing at least the fields <code>name</code> (Julia object name as a string) and <code>chains_file</code> .
<code>file</code>	A character string giving the base name or path for saving the <code>.rds</code> file. If the extension <code>.rds</code> is missing, it will be appended automatically.
<code>chains_file</code>	Optional character string giving the path where the Julia object should be serialized. The file name should have the <code>.jls</code> extension. If <code>NULL</code> , uses the <code>chains_file</code> field from <code>rjuliabugs_model</code> .

**Value**

No return value, called for saving both the Julia object and the R `rjuliabugs` object to disk.

**Examples**

```
## Not run:
save_rjuliabugs(my_model, file = "my_model", chains_file = "chains.jls")

## End(Not run)
```

---

setup_juliaBUGS	<i>Setup Julia Environment for JuliaBUGS</i>
-----------------	--

---

**Description**

Installs and loads the required Julia packages to use JuliaBUGS via JuliaCall in R.

**Usage**

```

setup_juliaBUGS(
  extra_packages = NULL,
  verify_package = TRUE,
  install_from_dev = FALSE,
  verbose = TRUE,
  ...
)

```

**Arguments**

`extra_packages` Character vector of additional Julia packages to install and load. Defaults to `NULL`, meaning only the core packages are handled.

`verify_package` Logical; if `TRUE`, verifies and installs missing core packages. Default is `TRUE`.

`install_from_dev` Logical; if `TRUE`, installs JuliaBUGS from its development repository. Default is `FALSE`.

`verbose` Logical. If `FALSE` will omit any message from the function to indicate the setup progress.

`...` Additional arguments passed to `JuliaCall::julia_setup()`, such as `installJulia = TRUE`.

**Details**

This function checks whether the core Julia packages needed for running JuliaBUGS are installed, installs any missing ones, and loads them into the current Julia session. Optionally, additional Julia packages can be installed and loaded by specifying them via `extra_packages`.

The core Julia packages installed (if needed) are:

- Serialization
- LogDensityProblemsAD
- ReverseDiff
- AdvancedHMC
- AbstractMCMC
- LogDensityProblems
- MCMCChains
- DataFrames
- JuliaBUGS

After installation, all these packages are loaded in the Julia session using `using`. Any additional packages provided via `extra_packages` are also installed and loaded.

**Value**

Invisibly returns `NULL`. The function is called for its side effects.

**See Also**

[julia\\_install\\_package\\_if\\_needed](#), [julia\\_eval](#)

**Examples**

```
## Not run:
# Setup Julia with core packages only
setup_juliaBUGS()

# Setup Julia with additional packages
setup_juliaBUGS(extra_packages = c("Distributions", "Turing"))

## End(Not run)
```

---

summary.rjuliabugs      *Summary Method for JuliaBUGS Sampler Output*

---

**Description**

Provides a summary of the results from a JuliaBUGS sampler object, including Markov Chain Monte Carlo (MCMC) settings, summary statistics, and optionally quantiles.

**Usage**

```
## S3 method for class 'rjuliabugs'
summary(object, ...)
```

**Arguments**

object	An object of class <code>rjuliabugs</code> containing a reference to a Markov Chain Monte Carlo (MCMC) sampler from Julia.
...	Additional optional arguments. Supported options: <ul style="list-style-type: none"> <li>• <code>digits</code>: Integer. Number of significant digits to display. Default: 4.</li> <li>• <code>n_display</code>: Integer. Number of rows of summary statistics to show. Default: 10.</li> <li>• <code>get_summary</code>: Logical. If TRUE, returns summary statistics in the output list. Default: FALSE.</li> <li>• <code>get_quantiles</code>: Logical. If TRUE, returns quantiles in the output list. Default: FALSE.</li> <li>• <code>julia_summary_only</code>: Logical. If TRUE, displays the Julia summary directly and returns NULL invisibly. Default: FALSE.</li> </ul>

**Details**

This method wraps Julia's `MCMCchains.summarystats` and `MCMCchains.quantile` to extract and display results in R using the `JuliaCall` interface. It also extracts key MCMC settings like number of chains, iterations, and samples per chain. The printed summary is truncated to `n_display` rows.

**Value**

If `julia_summary_only = TRUE`, no value is called for the print message. Otherwise, returns a list possibly containing:

- `summary`: Data frame of summary statistics (if `get_summary = TRUE`).
- `quantiles`: Data frame of quantiles (if `get_quantiles = TRUE`).

---

wrap\_model\_to\_juliaBUGS

*Wrap Bayesian Updating for Gibbs Sampling (BUGS) Model for Julia*

---

**Description**

Wraps a Bayesian Updating for Gibbs Sampling (BUGS) model string with ‘`model = @bugs begin`’ and ‘`end`’, if it is not already wrapped. This is useful for preparing BUGS models for use with Julia packages that expect this specific block structure.

**Usage**

```
wrap_model_to_juliaBUGS(model_code)
```

**Arguments**

`model_code` A character string containing the body of a Bayesian Updating for Gibbs Sampling (BUGS) model. If the model already starts with ‘`model = @bugs begin`’ and ends with ‘`end`’, the function returns it unchanged.

**Value**

A character string with the BUGS model wrapped properly in Julia-compatible syntax.

**Examples**

```
## Not run:
model_body <- "
  for i in 1:N
    r[i] ~ dbin(p[i], n[i])
    b[i] ~ dnorm(0.0, tau)
    p[i] = logistic(alpha0 + alpha1 * x1[i] + alpha2 * x2[i] + alpha12 * x1[i] * x2[i] + b[i])
  end
  alpha0 ~ dnorm(0.0, 1.0E-6)
  tau ~ dgamma(0.001, 0.001)
  sigma = 1 / sqrt(tau)
"
wrap_model_to_juliaBUGS(model_body)

## End(Not run)
```



# Index

as\_draws, [2](#)  
as\_mcmc, [3](#)  
as\_rvar, [4](#)

bugs2juliaBUGS, [5](#)

check\_sampler\_is\_defined, [12](#)  
coda::as.mcmc(), [4](#)  
coda::mcmc, [3](#)  
coda::mcmc(), [4](#)  
coda::mcmc.list, [3](#)  
convert\_numeric\_types, [6](#)

delete\_julia\_obj, [7](#)

extract, [7](#)

julia\_assign\_int, [11](#)  
julia\_eval, [15](#)  
julia\_install\_package\_if\_needed, [15](#)  
juliaBUGS, [8](#)

load\_rjuliabugs, [12](#)

posterior::as\_draws(), [3](#)  
posterior::draws, [2](#)  
posterior::rvar, [4](#)  
posterior::rvar(), [5](#)

save\_rjuliabugs, [13](#)  
setup\_juliaBUGS, [13](#)  
summary.rjuliabugs, [15](#)

wrap\_model\_to\_juliaBUGS, [16](#)