Package: PEcAn.settings (via r-universe)

June 27, 2024

Title PEcAn Settings package

Version 1.7.2

Date 2021-10-04

License BSD_3_clause + file LICENSE

Copyright Authors

LazyLoad yes

LazyData FALSE

Require hdf5

Description Contains functions to read PEcAn settings files.

Depends methods

Imports PEcAn.DB, PEcAn.logger, PEcAn.remote, PEcAn.utils, lubridate (>= 1.6.0), purrr, XML (>= 3.98-1.3), optparse

Suggests mockery, testthat (>= 2.0.0), withr

Encoding UTF-8

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Repository https://pecanproject.r-universe.dev

RemoteUrl https://github.com/PecanProject/pecan

RemoteRef HEAD

RemoteSha d5c7bffdf233077968945a182c11240b5d76e42d

Contents

`addSecrets` *Add Users secrets*

Description

Add secret information from ~/.pecan.xml

Usage

```
addSecrets(settings, force = FALSE)
```

Arguments

`settings`	settings file
`force`	Logical: add secrets even if they have been added previously?

Details

Copies certains sections from ~/.pecan.xml to the settings. This allows a user to have their own unique parameters also when sharing the pecan.xml file we don't expose these secrets. Currently this will copy the database and browndog sections

Value

will return the updated settings values

Author(s)

Rob Kooper

---

check.bety.version *Check BETY Version*

Description

check to make sure BETY is up to date

Usage

```
check.bety.version(dbcon)
```

Arguments

dbcon	database connection object

---

check.database *Check Database*

Description

Check Database

Usage

```
check.database(database)
```

Arguments

database	settings list to check. You'll probably use `settings$database`

check.database.settings *Check Database Settings*

Description

Check Database Settings

Usage

```
check.database.settings(settings)
```

Arguments

settings	settings file

check.ensemble.settings *Check ensemble Settings*

Description

Check ensemble Settings

Usage

```
check.ensemble.settings(settings)
```

Arguments

settings	settings file

check.inputs *Check Inputs*

Description

check to see if inputs are specified - this should be part of the model code

Usage

```
check.inputs(settings)
```

Arguments

settings	settings file

check.model.settings *Check Model Settings*

Description

Check Model Settings

Usage

```
check.model.settings(settings, dbcon = NULL)
```

Arguments

settings	settings file
dbcon	database connection.

check.run.settings *Check Run Settings*

Description

Check Run Settings

Usage

```
check.run.settings(settings, dbcon = NULL)
```

Arguments

settings	settings file
dbcon	database connection.

check.settings *Check Settings*

Description

Sanity checks. Checks the settings file to make sure expected fields exist. It will try to use default values for any missing values, or stop the exection if no defaults are possible.

Usage

```
check.settings(settings, force = FALSE)
```

Arguments

settings	settings file
force	Logical: Rerun check even if these settings have been checked previously?

Details

Expected fields in settings file are:

- pfts with at least one pft defined

Value

will return the updated settings values with defaults set.

Author(s)

Rob Kooper, David LeBauer

check.workflow.settings *Check Workflow Settings*

Description

Check Workflow Settings

Usage

```
check.workflow.settings(settings, dbcon = NULL)
```

Arguments

settings	settings file

clean.settings *Cleans PEcAn settings file*

Description

This will try and clean the settings file so it is ready for a new run. This will remove all run specific information and set the outdir to be 'pecan' for the next run.

Usage

```
clean.settings(inputfile = "pecan.xml", outputfile = "pecan.xml", write = TRUE)
```

Arguments

inputfile	the PEcAn settings file to be used.
outputfile	the name of file to which the settings will be written inside the outputdir.
write	Indicates whether to write the modified settings to a file.

Value

list of all settings as saved to the XML file(s)

Author(s)

Rob Kooper

Examples

```
## Not run:
clean.settings('output/PEcAn_1/pecan.xml', 'pecan.xml')

## End(Not run)
```

createMultiSiteSettings *Transform Settings into multi-site MultiSettings*

Description

Create a MultiSettings object containing (identical) run blocks for multiple different sites

Usage

```
createMultiSiteSettings(templateSettings, siteIds)
```

Arguments

templateSettings	A Settings object that will be the template for the resulting MultiSettings.
siteIds	The site IDs to be used in the resulting MultiSettings

Details

Starts with a template settings object, and duplicates the run$site block once for each specified site ID. The resulting MultiSettings is thus identical to the input, except ready to run for each site in the vector of site IDs.

Value

A MultiSettings object with the same settings as templateSettings but replicated run$site blocks, one for each specified site ID.

Author(s)

Ryan Kelly

Examples

```
dontrun <- function() { # Added by Alexey Shiklomanov
                        # so this doesn't run and break the build

# This isn't necessarily a fully working settings object.
# Enough to get the idea though.
# Note it has a $run block with settings that will be shared across all sites

template <- Settings(list(
  info = structure(list(
      notes = NULL, userid = "1000000005", username = "Ryan Kelly",
      date = "2016/07/13 13:23:46 -0400"),
    .Names = c("notes", "userid", "username", "date")),
  database = structure(list(
      bety = structure(
        list(user = "bety", password = "bety", host = "psql-pecan.bu.edu",
          dbname = "bety", driver = "PostgreSQL", write = "TRUE"),
        .Names = c("user", "password", "host", "dbname", "driver", "write")),
      fia = structure(
        list(user = "bety", password = "bety", host = "psql-pecan.bu.edu",
          dbname = "fia5", driver = "PostgreSQL", write = "true"),
        .Names = c("user", "password", "host", "dbname", "driver", "write"))),
    .Names = c("bety", "fia")),
  pfts = structure(list(
      pft = structure(
        list(comment = NULL, name = "temperate.Evergreen_Hardwood",
          constants = structure(list(num = "1"), .Names = "num")),
        .Names = c("comment", "name", "constants")),
      pft = structure(
        list(name = "temperate.Hydric",
```

```
        constants = structure(list(num = "2"), .Names = "num")),
      .Names = c("name", "constants"))),
  .Names = c("pft", "pft")),
meta.analysis = structure(list(
    iter = "3000", random.effects = list(on = FALSE, use_ghs = TRUE),
    update = "AUTO", threshold = "1.2"),
  .Names = c("iter", "random.effects", "update", "threshold")),
ensemble = structure(list(size = "1", variable = "NPP"),
  .Names = c("size", "variable")),
model = structure(list(id = "2000000005",
    edin = "/home/rykelly/pecan/RK_files/ED2IN/ED2IN.rgit.mandifore_04",
    config.header = structure(list(
        radiation = structure(list(lai_min = "0.01"), .Names = "lai_min"),
        ed_misc = structure(list(output_month = "12"),
          .Names = "output_month")),
      .Names = c("radiation", "ed_misc")),
    phenol.scheme = "0", prerun = "module load hdf5/1.8.11",
    binary = "/usr2/postdoc/rykelly/ED2/ED/build/ed_2.1-opt"),
  .Names = c("id", "edin", "config.header", "phenol.scheme", "prerun",
    "binary")),
host = structure(list(name = "geo.bu.edu", user = "rykelly",
    folder = "/projectnb/dietzelab/pecan.data/output/rykelly",
    qsub = "qsub -V -N @NAME@ -o @STDOUT@ -e @STDERR@ -S /bin/bash",
    qsub.jobid = "Your job ([0-9]+) .*",
    qstat = "qstat -j @JOBID@ || echo DONE",
    prerun = "module load udunits R/R-3.0.0_gnu-4.4.6",
    dbfiles = "/projectnb/dietzelab/pecan.data/input",
    modellauncher = structure(list(
        binary = "/usr2/postdoc/rykelly/pecan/utils/modellauncher/modellauncher",
        qsub.extra = "-pe omp 20"),
      .Names = c("binary", "qsub.extra"))),
  .Names = c("name", "user", "folder", "qsub", "qsub.jobid", "qstat",
    "prerun", "dbfiles", "modellauncher")),
run = structure(list(
    inputs = structure(list(
        met = structure(list(source = "NARR", output = "ED2"),
          .Names = c("source", "output")),
        lu = structure(list(id = "294",
            path = "/projectnb/dietzelab/EDI/ed_inputs/glu/"),
          .Names = c("id", "path")),
        soil = structure(list(id = "297",
            path = "/projectnb/dietzelab/EDI/faoOLD/FAO_"),
          .Names = c("id", "path")),
        thsum = structure(list(id = "295",
            path = "/projectnb/dietzelab/EDI/ed_inputs/"),
          .Names = c("id", "path")),
        veg = structure(list(id = "296",
            path = "/projectnb/dietzelab/EDI/oge2OLD/OGE2_"),
          .Names = c("id", "path")),
        pss = structure(list(source = "FIA"), .Names = "source")),
      .Names = c("met", "lu", "soil", "thsum", "veg", "pss")),
    start.date = "2004/01/01",
    end.date = "2004/01/31"),
```

```
    .Names = c("inputs", "start.date", "end.date"))
))


sitegroupId <- 1000000002
startDate <- "2000/01/01"
endDate <- "2015/12/31"
nSite <- 10
outDir <- "~/multisite_setup_test"


template <- setDates(template, startDate = startDate, endDate = endDate)
template <- setOutDir(template, outDir)

multiRunSettings <- createSitegroupMultiSettings(
    template,
    sitegroupId = sitegroupId,
    nSite = nSite)

dir.create(outDir, showWarnings = FALSE)
write.settings(multiRunSettings, outputfile = "pecan.xml")

} # dontrun
```

`createSitegroupMultiSettings` *Create Sitegroup MultiSettings*

Description

Helps to create a MultiSettings object to run some or all sites in a Sitegroup.

Usage

```
createSitegroupMultiSettings(
  templateSettings,
  sitegroupId,
  nSite,
  con = NULL,
  params = templateSettings$database$bety
)
```

Arguments

`templateSettings`	A `Settings` object that will be the template for the resulting MultiSettings.
`sitegroupId`	The Bety ID of the sitegroup to draw from
`nSite`	The number of sites to randomly select (without replacement) from the siteGroup. Omit to use all sites in the group.
`con, params`	Bety DB connection or parameters. Passed directly to `db.query`

Details

Starts with a template settings object, and fills in the run block with site info sampled from the sitegroup. The template could be fully set up except for the site info, or more or less empty if you plan to fill in the other settings later. A MultiSettings is created from templateSettings, nSite sites (or all of them, if nSite is unset) are selected from Bety, and their info is dropped into the MultiSettings.

Value

A MultiSettings object with the same settings as templateSettings but site information for the selected sites

Author(s)

Ryan Kelly

Examples

```
dontrun <- function() { # Added by Alexey Shiklomanov
                        # so this doesn't run and break the build

# This isn't necessarily a fully working settings object.
# Enough to get the idea though.
# Note it has a $run block with settings that will be shared across all sites

template <- Settings(list(
  info = structure(list(
      notes = NULL, userid = "1000000005", username = "Ryan Kelly",
      date = "2016/07/13 13:23:46 -0400"),
    .Names = c("notes", "userid", "username", "date")),
  database = structure(list(
      bety = structure(
        list(user = "bety", password = "bety", host = "psql-pecan.bu.edu",
          dbname = "bety", driver = "PostgreSQL", write = "TRUE"),
        .Names = c("user", "password", "host", "dbname", "driver", "write")),
      fia = structure(
        list(user = "bety", password = "bety", host = "psql-pecan.bu.edu",
          dbname = "fia5", driver = "PostgreSQL", write = "true"),
        .Names = c("user", "password", "host", "dbname", "driver", "write"))),
    .Names = c("bety", "fia")),
  pfts = structure(list(
      pft = structure(
        list(comment = NULL, name = "temperate.Evergreen_Hardwood",
          constants = structure(list(num = "1"), .Names = "num")),
        .Names = c("comment", "name", "constants")),
      pft = structure(
        list(name = "temperate.Hydric",
          constants = structure(list(num = "2"), .Names = "num")),
        .Names = c("name", "constants"))),
    .Names = c("pft", "pft")),
  meta.analysis = structure(list(
```

```
      iter = "3000", random.effects = list(on = FALSE, use_ghs = TRUE),
      update = "AUTO", threshold = "1.2"),
    .Names = c("iter", "random.effects", "update", "threshold")),
  ensemble = structure(list(size = "1", variable = "NPP"),
    .Names = c("size", "variable")),
  model = structure(list(id = "2000000005",
      edin = "/home/rykelly/pecan/RK_files/ED2IN/ED2IN.rgit.mandifore_04",
      config.header = structure(list(
          radiation = structure(list(lai_min = "0.01"), .Names = "lai_min"),
          ed_misc = structure(list(output_month = "12"),
            .Names = "output_month")),
        .Names = c("radiation", "ed_misc")),
      phenol.scheme = "0", prerun = "module load hdf5/1.8.11",
      binary = "/usr2/postdoc/rykelly/ED2/ED/build/ed_2.1-opt"),
    .Names = c("id", "edin", "config.header", "phenol.scheme", "prerun",
      "binary")),
  host = structure(list(name = "geo.bu.edu", user = "rykelly",
      folder = "/projectnb/dietzelab/pecan.data/output/rykelly",
      qsub = "qsub -V -N @NAME@ -o @STDOUT@ -e @STDERR@ -S /bin/bash",
      qsub.jobid = "Your job ([0-9]+) .*",
      qstat = "qstat -j @JOBID@ || echo DONE",
      prerun = "module load udunits R/R-3.0.0_gnu-4.4.6",
      dbfiles = "/projectnb/dietzelab/pecan.data/input",
      modellauncher = structure(list(
          binary = "/usr2/postdoc/rykelly/pecan/utils/modellauncher/modellauncher",
          qsub.extra = "-pe omp 20"),
        .Names = c("binary", "qsub.extra"))),
    .Names = c("name", "user", "folder", "qsub", "qsub.jobid", "qstat",
      "prerun", "dbfiles", "modellauncher")),
  run = structure(list(
      inputs = structure(list(
          met = structure(list(source = "NARR", output = "ED2"),
            .Names = c("source", "output")),
          lu = structure(list(id = "294",
              path = "/projectnb/dietzelab/EDI/ed_inputs/glu/"),
            .Names = c("id", "path")),
          soil = structure(list(id = "297",
              path = "/projectnb/dietzelab/EDI/faoOLD/FAO_"),
            .Names = c("id", "path")),
          thsum = structure(list(id = "295",
              path = "/projectnb/dietzelab/EDI/ed_inputs/"),
            .Names = c("id", "path")),
          veg = structure(list(id = "296",
              path = "/projectnb/dietzelab/EDI/oge2OLD/OGE2_"),
            .Names = c("id", "path")),
          pss = structure(list(source = "FIA"), .Names = "source")),
        .Names = c("met", "lu", "soil", "thsum", "veg", "pss")),
      start.date = "2004/01/01",
      end.date = "2004/01/31"),
    .Names = c("inputs", "start.date", "end.date"))
))
```

```
sitegroupId <- 1000000002
startDate <- "2000/01/01"
endDate <- "2015/12/31"
nSite <- 10
outDir <- "~/multisite_setup_test"


template <- setDates(template, startDate = startDate, endDate = endDate)
template <- setOutDir(template, outDir)

multiRunSettings <- createSitegroupMultiSettings(
    template,
    sitegroupId = sitegroupId,
    nSite = nSite)

dir.create(outDir, showWarnings = FALSE)
write.settings(multiRunSettings, outputfile = "pecan.xml")

} # dontrun
```

`expandMultiSettings` *generic function for expanding multi-settings.*

Description

generic function for expanding multi-settings.

Usage

```
expandMultiSettings(x)
```

Arguments

`x`	object to be expanded.

`fix.deprecated.settings` *Fix Deprecated Settings*

Description

Checks for and attempts to fix deprecated settings structure

Usage

```
fix.deprecated.settings(settings, force = FALSE)
```

Arguments

settings	settings list
force	Logical: re-run fixing of deprecated settings even if it has been done previously?

Value

updated settings list

Author(s)

Ryan Kelly

---

getRunSettings *Build run MultiSettings for a single site id*

Description

Processes one site from the siteIds argument of createMultiSiteSettings. You probably don't need to call it directly.

Usage

```
getRunSettings(templateSettings, siteId)
```

Arguments

templateSettings	A Settings object that will be the template for the resulting MultiSettings.
siteId	site to process. See createMultiSiteSettings

---

get_args *Get Args*

Description

Used in web/workflow.R to parse command line arguments. See also https://github.com/PecanProject/pecan/pull/2626.

Usage

```
get_args()
```

Value

list generated by parse_args; see there for details.

Examples

```
## Not run: ./web/workflow.R -h
```

listToXml *A generic function to convert list to XML*

Description

A generic function to convert list to XML

Usage

```
listToXml(x, ...)
```

Arguments

x	list to be converted
...	arguments passed to methods

listToXml.default *List to XML*

Description

Convert List to XML

Usage

```
## Default S3 method:
listToXml(x, ...)
```

Arguments

x	object to be converted. Despite the function name, need not actually be a list
...	further arguments. Used to set the element name of the created XML object, which is taken from an argument named tag if present, or otherwise from the first element of ...

Details

Can convert list or other object to an xml object using xmlNode

Value

xmlNode

Author(s)

David LeBauer, Carl Davidson, Rob Kooper

loadPath.sitePFT *Title loadPath.sitePFT*

Description

The csv or the text file needs to have a header and be separated using comma. Under the first column in the text file, one needs to specify the site id and in the second column there has to be the name of the PFT.

Usage

```
loadPath.sitePFT(settings, Path)
```

Arguments

settings	pecan setting list.
Path	Character of file name with extension. The path will be generated using the outdir tag in pecan settings.

Value

a dataframe of two columns of site and pft

MultiSettings *Create a PEcAn MultiSettings object*

Description

Create a PEcAn MultiSettings object

Usage

```
MultiSettings(...)

as.MultiSettings(x)

is.MultiSettings(x)
```

Arguments

...	Settings objects to concatenate
x	object to test or coerce

Value

list with class "Multisettings"

Functions

- `as.MultiSettings()`: coerce an existing object to MultiSettings
- `is.MultiSettings()`: test if an object is a MultiSettings

Author(s)

Ryan Kelly

`papply` *Apply functions to PEcAn MultiSettings*

Description

Works like lapply(), but for PEcAn Settings and MultiSettings objects

Usage

```
papply(settings, fn, ..., stop.on.error = FALSE)
```

Arguments

`settings`	A `MultiSettings`, `Settings`, or `list` to operate on
`fn`	The function to apply to `settings`
`...`	additional arguments to `fn`
`stop.on.error`	Whether to halt execution if a single element in `settings` results in error. See Details.

Details

`papply` is mainly used to call a function on each `Settings` object in a `MultiSettings` object, and returning the results in a list. It has some additional features, however:

- If the result of `fn` is a `Settings` object, then `papply` will coerce the returned list into a new `MultiSettings`.
- If `settings` is a `Settings` object, then `papply` knows to call `fn` on it directly.
- If `settings` is a generic `list`, then `papply` coerces it to a `Settings` object and then calls `fn` on it directly. This is meant for backwards compatibility with old-fashioned PEcAn settings lists, but could have unintended consequences
- By default, `papply` will proceed even if `fn` throws an error for one or more of the elements in `settings`. Note that if this option is used, the returned results list will have entries for *only* those elements that did not result in an error.

Value

A single `fn` return value, or a list of such values (coerced to `MultiSettings` if appropriate; *see Details*)

Author(s)

Ryan Kelly

Examples

```
f = function(settings, ...) {
  # Here's how I envisioned a typical use case within a standard PEcAn function
  if(is.MultiSettings(settings)) {
    return(papply(settings, f, ...))
  }

 # Don't worry about the beolow, it's just some guts to make the function do something we can see
  l <- list(...)
  for(i in seq_along(l)) {
    ind <- length(settings) + 1
    settings[[ind]] <- l[[i]]
    if(!is.null(names(l))) {
      names(settings)[ind] <- names(l)[i]
    }
  }
  return(settings)
}

# Example
settings1 <- Settings(list(a="aa", b=1:3, c="NA"))
settings2 <- Settings(list(a="A", b=4:5, c=paste))
multiSettings <- MultiSettings(settings1, settings2)

# The fucntion should add element $d = D to either a Settings, or each entry in a MultiSettings
f(settings1, d="D")
print(f(multiSettings, d="D"), TRUE)
```

`prepare.settings` *Prepare Settings*

Description

Update, set defaults for, and otherwise prepare a PEcAn Settings object

Usage

```
prepare.settings(settings, force = FALSE)
```

Arguments

`settings`	settings list
`force`	Whether to force the function to run even if it determines it has been run on these settings already.

Details

Performs various checks, fixes deprecated contructs, and assigns missing values where possible.

Author(s)

Ryan Kelly

Betsy Cowdery

---

printAll *generic function for printing contents of objects.*

Description

generic function for printing contents of objects.

Usage

```
printAll(x)
```

Arguments

x	object to be printed.

---

read.settings *Loads PEcAn settings file*

Description

This will try and find the PEcAn settings file in the following order:

1. `--settings <file>` passed as command line argument using `--settings`
2. `inputfile` passed as argument to function
3. `PECAN_SETTINGS` environment variable `PECAN_SETTINGS` pointing to a specific file
4. `./pecan.xml` `pecan.xml` in the current folder

Usage

```
read.settings(inputfile = "pecan.xml")
```

Arguments

inputfile	the PEcAn settings file to be used.

Details

Once the function finds a valid file, it will not look further. Thus, if `inputfile` is supplied, `PECAN_SETTINGS` will be ignored. Even if a `file` argument is passed, it will be ignored if a file is passed through a higher priority method.

Value

list of all settings as loaded from the XML file(s)

Author(s)

Shawn Serbin

Rob Kooper

David LeBauer

Ryan Kelly

Betsy Cowdery

Examples

```
## Not run:
## bash shell:
## example workflow.R and pecan.xml files in pecan/tests
R --vanilla -- --settings path/to/mypecan.xml < workflow.R

## R:

settings <- read.settings()
settings <- read.settings(file="willowcreek.xml")
test.settings.file <- system.file("tests/test.xml", package = "PEcAn.all")
settings <- read.settings(test.settings.file)

## End(Not run)
```

SafeList *Create a SafeList object*

Description

`SafeList` is a wrapper class for the normal R list. It should behave identically, except for the `$` operator being overridden to require exact matches.

Usage

```
SafeList(...)

as.SafeList(x)

is.SafeList(x)
```

Arguments

...	A list to upgrade to SafeList, or elements to be added to a new SafeList
x	list object to be tested or coerced

Details

The constructor works identical to `list()` unless:

1. The only argument is a list, in which case the result is the same list, with its class attribute updated to include 'SafeList', or
2. The only argument is a SafeList, in which case that argument is returned unchanged

Value

The resulting SafeList

a SafeList version of x

logical

Functions

- `as.SafeList()`: Coerce an object to SafeList.
- `is.SafeList()`: Test if object is already a SafeList.

Author(s)

Ryan Kelly

`setDates` *Set the Dates of PEcAn Settings*

Description

Sets the run, ensemble, and sensitivity analysis dates of PEcAn Settings

Usage

```
setDates(settings, startDate, endDate)
```

Arguments

`settings`	A `Settings` object
`startDate`, `endDate`	The desired start and end dates

Details

Sets the start/end dates in `settings$run` to the specified dates, and sets the corresponding years for `settings$ensemble` and `settings$sensitivity.analysis`. Either date can be omitted to leave it unchanged.

Value

The original `Settings` object with updated dates

Author(s)

Ryan Kelly

`setOutDir` *Set the Output Directories of PEcAn Settings*

Description

Sets the main output directory and nulls out the others

Usage

```
setOutDir(settings, outDir)
```

Arguments

`settings`	A `Settings` object
`outDir`	The desired output directory

Details

Sets the main output directory (`settings$outdir`) to `outDir`, and sets numerous others (`settings$modeloutdir`, `settings$host$rundir`, `settings$host$outdir`, `settings$host$modeloutdir`) to NULL so they will revert to defaults when `check.settings` is run.

Value

The original `Settings` object with updated output directories

Author(s)

Ryan Kelly

settingNames *function that can retrieve or update the names of multi-settings.*

Description

function that can retrieve or update the names of multi-settings.

Usage

```
settingNames(multiSettings, settingNames)
```

Arguments

multiSettings	object for which to retrieve or set the names.
settingNames	names to be set for the multi-settings object.

Settings *Create a PEcAn Settings object*

Description

Create a PEcAn Settings object

Usage

```
Settings(...)

as.Settings(x)

is.Settings(x)
```

Arguments

...	objects to concatenate
x	object to test or coerce

Value

a list containing all objects in ..., with class c("Settings", "SafeList", "list").

Functions

- as.Settings(): coerce an object to Settings
- is.Settings(): test if object is already a Settings

Author(s)

Ryan Kelly

`site.pft.link.settings` *site.pft.link.settings*

Description

This function reads in a pecan setting and check for the pft.site xml tag under run>inputs. If a path or a ID for the input is defined then, it will be used for linking sites with the pfts.

Usage

```
site.pft.link.settings(settings)
```

Arguments

`settings`	settings list

Value

pecan xml setting file

`site.pft.linkage` *site.pft.linkage*

Description

This function creates the required tags inside pecan.xml to link sites with pfts given a look up table. If the required tags are already defined in the pecan xml then they will be updated. If there are multiple pfts that they need to be used for a site, each pft needs to have a separate row in the lookup table, resulting multiple rows for a site.

Usage

```
site.pft.linkage(settings, site.pft.links)
```

Arguments

`settings`	pecan settings list.
`site.pft.links`	dataframe. Your look up table should have two columns of site and pft with site ids under site column and pft names under pft column.

Value

pecan setting list

Examples

```
## Not run:
#setting up the Look up tables
site.pft.links <-tribble(
 ~site, ~pft,
 "1000025731", "temperate.broadleaf.deciduous1",
 "1000025731", "temperate.needleleaf.evergreen",
 "1000000048", "temperate.broadleaf.deciduous2",
 "772", "temperate.broadleaf.deciduous3",
 "763", "temperate.broadleaf.deciduous4"
)

# sending a multi- setting xml file to the function
site.pft.linkage(settings,site.pft.links)

## End(Not run)
```

---

update.settings *Update Settings*

Description

Updates a pecan.xml file to match new layout. This will take care of the conversion to the latest pecan.xml file.

Usage

```
## S3 method for class 'settings'
update(settings, force = FALSE)
```

Arguments

settings	settings file
force	Logical: update even if settings have previously been updated?.

Value

will return the updated settings values

Author(s)

Rob Kooper

write.settings *Write settings*

Description

Takes in a settings object, performs a series of checks, fixes & updates settings and produces pecan.CHECKED.xml

Usage

```
write.settings(settings, outputfile, outputdir = settings$outdir)
```

Arguments

settings	settings list
outputfile	the file name to write to
outputdir	the directory to write to

Author(s)

Ryan Kelly

Betsy Cowdery

$.SafeList *Extract SafeList component by name*

Description

Extract SafeList component by name

Usage

```
## S3 method for class 'SafeList'
x$name
```

Arguments

x	the SafeList object
name	the name of the component

Details

Overrides `$.list`, and works just like it except forces exact match (i.e., makes `x$name` behave exactly like `x[[name, exact=T]]`)

Value

The specified component

Author(s)

Ryan Kelly

Index