Package 'provDebugR'

October 14, 2022

Title A Time-Travelling Debugger

Version 1.0.1

Date 2021-04-20

Description Uses provenance post-execution to help the user understand and debug their script by providing functions to look at intermediate steps and data values, their forwards and backwards lineage, and to understand the steps leading up to warning and error messages. 'provDebugR' uses provenance produced by 'rdtLite' (available on CRAN), stored in PROV-JSON format.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

Imports httr, jsonlite, provGraphR, provParseR, textutils,

Suggests knitr, rdtLite, rdt, rmarkdown, testthat

Additional_repositories https://end-to-end-provenance.github.io/drat/

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Orenna Brand [aut],
Elizabeth Fong [aut],
Barbara Lerner [cre],
Rose Sheehan [aut],
Joseph Wonsil [aut],
Emery Boose [aut]

Maintainer Barbara Lerner <blerner@mtholyoke.edu>

Repository CRAN

Date/Publication 2021-04-22 15:00:05 UTC

R topics documented:

`debug.error` *Tracking the Lineage of Errors and Warnings*

Description

debug.error displays the backwards lineage of (the statements leading up to) an error that occurred when R code was executed.

debug.warning displays the backwards lineage of (the statements leading up to) one or more warnings that occurred when R code was executed.

Usage

```
debug.error(stack.overflow = FALSE)

debug.warning(..., all = FALSE)
```

Arguments

`stack.overflow`	If TRUE, the error message will be searched for on Stack Overflow.
`...`	The warning(s) to be queried.
`all`	If TRUE, the lineages of all warnings are returned.

Details

These functions are part of the provDebugR package. To use them, you must first initialise the debugger using one its initialisation functions: `prov.debug`, `prov.debug.file`, or `prov.debug.run`.

The lineage is represented with a data frame that shows the R statements whose execution led to the error or warning. More specifically, each row of the data frame corresponds to one line of code. The columns of the data frame are:

- scriptNum: The script number the statement is from.
- scriptName: The name of the script the statement is from.
- startLine: The line number for the statement If the statement spans multiple lines, this will be the first line.
- code: The statement itself. If the statement is long, this will just be the start of the statement.

Value

debug.error returns a data frame representing the backwards lineage of the error in the execution, if any.

debug.warning returns a list of data frames of lineages for the queried warnings.

See Also

provDebugR Initialisation Functions:

prov.debug

prov.debug.file

prov.debug.run

Other provDebugR Functions (non-initialisation):

debug.line: Returns all immediate inputs and outputs for the line(s) queried.

debug.lineage: Returns the forwards or backwards lineage of the data object(s) queried. The forwards lineage shows how the data object was used, and the backwards lineage shows how it was produced.

debug.state: Returns the state at the line(s) queried, after the line had been executed. The state is the list of all variables and their values in the environment at the queried line.

debug.type.changes: Returns a data frame for each variable in the execution containing the instances where the data type changed.

debug.variable: Returns a data frame showing all instances of the variable(s) queried.

debug.view: Opens and displays the contents of each file or variable or variable queried.

Examples

```
## Not run:
prov.debug.run("test.R")

debug.error()
debug.error(stack.overflow = TRUE)

debug.warning(1)
debug.warning(2,3)
debug.warning(all = TRUE)

## End(Not run)
```

debug.line *Displaying variable values*

Description

debug.line displays the values of variables that are either used or set on a particular line.

debug.state displays the values of all variables in the global environment after execution of a particular line.

debug.variable shows all values that a particular variable has during execution of a script.

Usage

```
debug.line(..., script.num = 1, all = FALSE)

debug.state(..., script.num = 1, showType = FALSE)

debug.variable(
  ...,
  val.type = "all",
  script.num = "all",
  all = FALSE,
  showType = FALSE
)
```

Arguments

...	The variable names to be queried.
script.num	The script number of the queried variables. Defaults to "all".
all	If TRUE, results for all variables of the specified script will be returned.
showType	If TRUE, variable container, dimension, and type are displayed.
val.type	If not "all", this filters the results to contain only instances where the valType (container or type) has the queried type. Only one type may be queried per function call.

Details

These functions are part of the provDebugR package. To use them, you must first initialise the debugger using one its initialisation functions: prov.debug, prov.debug.file, or prov.debug.run.

For each line number queried, debug.line returns a data frame of the data that the procedure in that line inputs and outputs. Each data frame contains the following columns:

- name: The name of the data.
- value: The value of the data.
- container: The type of the container of the data, such as vector or data frame.

- dimension: The size of the container.
- type: The data type(s) contained within the container.

For each queried line, debug.state returns a data frame showing the state at that line, after it has been executed. Each data frame contains the following columns:

- name: The names of variables in the state.
- value: The value of each variable.
- container: The type of the container of each variable.
- dimension: The size of the container.
- type: The data type(s) contained within the container.
- scriptNum: The script number associated with each variable.
- scriptName: The name of the script the variable is associated with.
- startLine: The line number associated with each variable.

If no paramters are given, debug.state will return the state at the end of execution.

For each variable queried, debug.variable returns a data frame of all instances (data nodes) of that variable. Each data frame contains the following columns:

- value: The value of the variable.
- container: The type of the container of the variable.
- dimension: The size of the container.
- type: The data type(s) contained within the container.
- scriptNum: The script number the variable is associated with.
- scriptName: The name of the script the variable or file is associated with.
- startLine: The line number the variable is associated with.
- code: The code this variable is associated with.

Value

debug.line returns a list of data frames showing the inputs and outputs for the procedure in each line queried.

debug.state returns a list of data frames of states for each queried line number, or the state at the end of execution if no parameters are given to the function.

debug.variable returns a list of data frames showing all instances of each variable queried.

See Also

provDebugR Initialisation Functions:

`prov.debug`

`prov.debug.file`

`prov.debug.run`

Other provDebugR Functions (non-initialisation):

debug.error: Returns the backwards lineage of the error, if any. The error may be queried on StackOverflow.

debug.lineage: Returns the forwards or backwards lineage of the data object(s) queried. The forwards lineage shows how the data object was used, and the backwards lineage shows how it was produced.

debug.type.changes: Returns a data frame for each variable in the execution containing the instances where the data type changed.

debug.warning: Returns the backwards lineage of the queried warning(s), if any.

Examples

```
## Not run:
prov.debug.run("test.R")
debug.line(5)
debug.line(all = TRUE)
debug.line(5, 10, script.num = 2)
debug.line(3, script.num = "all")

## End(Not run)

## Not run:
prov.debug.run("test.R")
debug.state()
debug.state(5)
debug.state(10, 20, script.num = 2)
debug.state(5, script.num = "all")

## End(Not run)

## Not run:
prov.debug.run("test.R")
debug.variable(x)
debug.variable(all = TRUE)
debug.variable("a", b, "x", val.type = "logical")
debug.variable("a", "b", x, script.num = 3)

## End(Not run)
```

---

debug.lineage *The Lineage of a Variable/Data Node.*

Description

For each data node queried, debug.lineage returns a data frame representing its forwards (how the data is used), or backwards (how the data was generated) lineage. Each data frame contains the following columns:

- scriptNum: The script number the data node is associated with.

- scriptName: The name of the script the data node is associated with.
- startLine: The line number the data node is associated with.
- code: The line of code which used/produced the data node.

Usage

```
debug.lineage(
  ...,
  start.line = NA,
  script.num = 1,
  all = FALSE,
  forward = FALSE
)
```

Arguments

`...`	The names of data nodes to be queried.
`start.line`	The line number of the queried data nodes. Optional.
`script.num`	The script number of the queried data nodes. Defaults to script number 1 (main script).
`all`	If TRUE, this function returns the linages of all data node names.
`forward`	If TRUE, this function returns the forwards lineage (how the data is used) instead of the backwards lineage (how the data was generated).

Details

debug.lineage belongs to provDebugR, a debugger which utilises provenance collected post-execution to facilitate understanding of the execution and aid in debugging.

This function may be used only after the debugger has been initialised using one its initialisation functions (listed below).

Value

A list of data frames showing the forwards or backwards lineage of all queried data nodes.

See Also

provDebugR Initialisation Functions:

`prov.debug`

`prov.debug.file`

`prov.debug.run`

Other provDebugR Functions (non-initialisation):

`debug.error`: Returns the backwards lineage of the error, if any. The error may be queried on StackOverflow.

`debug.line`: Returns all immediate inputs and outputs for the line(s) queried.

debug.state: Returns the state at the line(s) queried, after the line had been executed. The state is the list of all variables and their values in the environment at the queried line.

debug.type.changes: Returns a data frame for each variable in the execution containing the instances where the data type changed.

debug.variable: Returns a data frame showing all instances of the variable(s) queried.

debug.view: Opens and displays the contents of each file or variable or variable queried.

debug.warning: Returns the backwards lineage of the queried warning(s), if any.

Examples

```
## Not run:
prov.debug.run("test.R")
debug.lineage(x)
debug.lineage("x", start.line = 5, script.num = 2)
debug.lineage("a", b, forward = TRUE)
debug.lineage(all = TRUE)

## End(Not run)
```

---

debug.type.changes *Tracking Type Changes*

Description

Returns a data frame for each variable in the execution containing the instances where the data type changed. Each data frame contains the following columns:

- value: The value of the variable.
- container: The type of the container of the variable.
- dimension: The size of the container.
- type: The data type(s) contained within the container.
- code: The line of code associated with the variable.
- scriptNum: The script number associated with the variable.
- scriptName: The name of the script associated with the variable.
- startLine: The line number associated with the variable.

Usage

```
debug.type.changes(...)
```

Arguments

...	Optional. Variable name(s) to be queried. If variables are given (not NULL), the results will be filtered to show only those with the given variable name.

Details

debug.type.changes belongs to provDebugR, a debugger which utilises provenance collected post-execution to facilitate understanding of the execution and aid in debugging.

This function may be used only after the debugger has been initialised using one its initialisation functions (listed below).

Value

A list of data frames for each variable with at least 1 data type change.

See Also

provDebugR Initialisation Functions:

prov.debug

prov.debug.file

prov.debug.run

Other provDebugR Functions (non-initialisation):

debug.error: Returns the backwards lineage of the error, if any. The error may be queried on StackOverflow.

debug.line: Returns all immediate inputs and outputs for the line(s) queried.

debug.lineage: Returns the forwards or backwards lineage of the data object(s) queried. The forwards lineage shows how the data object was used, and the backwards lineage shows how it was produced.

debug.state: Returns the state at the line(s) queried, after the line had been executed. The state is the list of all variables and their values in the environment at the queried line.

debug.variable: Returns a data frame showing all instances of the variable(s) queried.

debug.view: Opens and displays the contents of each file or variable or variable queried.

debug.warning: Returns the backwards lineage of the queried warning(s), if any.

Examples

```
## Not run:
prov.debug.run("test.R")
debug.type.changes()
debug.type.changes(x)
debug.type.changes("a", "b")

## End(Not run)
```

debug.view *debug.view*

Description

debug.view displays the contents of a file or variable at a particular line of code in a separate view panel. This is best for large values like data frames and matrices.

Usage

```
debug.view(..., start.line = "all", script.num = "all")
```

Arguments

...	The variable names or file names to be queried.
start.line	The line number of the queried variables or files.
script.num	The script number of the queried variables or files.

Details

debug.view displays the contents of each file or variable queried. For snapshots or files with the file extension of .csv or .txt, the data will be loaded into the debugger environment before it is viewed. Otherwise, the data will be viewed using the system's default program for that type of file.

Value

debug.view returns a data frame containing the information that is displayed, which contains the following columns:

- name: The name of the variable or file being viewed.
- startLine: The line number the variable or file is associated with.
- scriptNum: The script number the variable or file is associated with.
- scriptName: The name of the script the variable or file is associated with.
- title: The title of the variable or file when viewed.
- notes: Will display PARTIAL if the variable is a partial snapshot, or indicate that the provenance directory or a file is not found. NA otherwise.

If there is no data to display, NULL is returned.

Examples

```
## Not run:
prov.debug.run("test.R")
debug.view()
debug.view(x)
debug.view("x", y, start.line = 5, script.num = 2)
```

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

---

prov.debug *A Time-Travelling Debugger for R - Debugger Initialization*

Description

prov.debug uses the provenance from the last execution of prov.run to initialise the debugger.

prov.debug.file reads a PROV-JSON file to initialise the debugger.

prov.debug.run executes a R or Rmd script, collects provenance, and initialises the debugger using the collected provenance.

Usage

```
prov.debug()

prov.debug.file(prov.file)

prov.debug.run(script, ...)
```

Arguments

prov.file	Path to a PROV-JSON file.
script	Path to an R script.
...	extra parameters are passed to the provenance collector. See rdt's prov.run function or rdtLites's prov.run function for details.

Details

Provenance is a detailed record of the execution of a script which includes information about the steps that were excecuted and the intermediate data values that were used and/or created. After it is collected, it can be used in a variety of ways to better understand the execution.

This package, provDebugR, is one such application, using provenance post-execution to help the user understand and debug their script by providing functions to look at intermediate steps and data values, as well as their forwards or backwards lineage. These functions may be used only after provDebugR has been initialised using one of the initialisation functions above.

The forwards lineage of a data object is the list of steps showing how the data object was used. The backwards lineage of a data object is the list of steps showing how the data object was produced.

provDebugR uses provenance produced by rdtLite (a provenance collection package available on CRAN), stored in PROV-JSON format.

Value

No return value.

References

rdtLite (Provenance Collection Tool): https://CRAN.R-project.org/package=rdtLite

PROV-JSON output produced by rdtLite: https://github.com/End-to-end-provenance/ExtendedProvJson/blob/master/JSON-format.md

PROV-JSON standard: https://www.w3.org/Submission/2013/SUBM-prov-json-20130424/

See Also

Other provDebugR Functions (non-initialisation):

debug.error: Returns the backwards lineage of the error, if any. The error may be queried on StackOverflow.

debug.line: Returns all immediate inputs and outputs for the line(s) queried.

debug.lineage: Returns the forwards or backwards lineage of the data object(s) queried. The forwards lineage shows how the data object was used, and the backwards lineage shows how it was produced.

debug.state: Returns the state at the line(s) queried, after the line had been executed. The state is the list of all variables and their values in the environment at the queried line.

debug.type.changes: Returns a data frame for each variable in the execution containing the instances where the data type changed.

debug.variable: Returns a data frame showing all instances of the variable(s) queried.

debug.view: Opens and displays the contents of each file or variable or variable queried.

debug.warning: Returns the backwards lineage of the queried warning(s), if any.

Other tools that use provenance: https://github.com/End-to-end-provenance

Examples

```
## Not run:
rdtLite::prov.run("test.R")
prov.debug()
## End(Not run)

## Not run:
prov.debug.file("prov_test/prov.json")
## End(Not run)

## Not run:
prov.debug.run("test.R", snapshot.size = 100)
## End(Not run)
```

Index