Package 'JuniperKernel'

February 5, 2018

Title Kernel for 'Jupyter'

Version 1.2.3.0

Date 2018-01-15

Copyright See the file COPYRIGHTS

Depends R (>= 3.2.0)

Imports utils, jsonlite, repr, Rcpp (>= 0.11.0), gdtools (>= 0.1.6), pbdZMQ (>= 0.3-2), data.table, methods

Suggests RUnit, roxygen2

Collate 'booter.R' 'comm_close.R' 'comm_info_request.R' 'comm_msg.R' 'comm_open.R' 'complete_request.R' 'execute_request.R' 'history_request.R' 'inspect_request.R' 'is_complete_request.R' 'juniper.R' 'kernel_info_request.R' 'kernelspec.R' 'RcppExports.R' 'request.R' 'shutdown_request.R' 'zzz.R'

LinkingTo Rcpp, gdtools

Description Provides a full implementation of the 'Jupyter' <http://jupyter.org/> messaging protocol in C++ by leveraging 'Rcpp' and 'Xeus' <https://github.com/QuantStack/xeus>.
'Jupyter' supplies an interactive computing environment and a messaging protocol defined over 'ZeroMQ' for multiple programming languages. This package implements the 'Jupyter' kernel interface so that 'R' is exposed to this interactive computing environment. 'ZeroMQ' functionality is provided by the 'pbdZMQ' package.
'Xeus' is a C++ library that facilitates the implementation of kernels for 'Jupyter'. Additionally, 'Xeus' provides an interface to libraries that exist in the 'Jupyter' ecosystem for building widgets,
plotting, and more <https://blog.jupyter.org/interactive-workflows-for-c-with-jupyter-fe9b54227d92>. 'JuniperKernel' uses 'Xeus' as a library for the 'Jupyter' messaging protocol.

URL https://github.com/JuniperKernel/JuniperKernel

License GPL (>= 2)

BugReports https://github.com/JuniperKernel/JuniperKernel/issues

RoxygenNote 6.0.1

NeedsCompilation yes

Author Spencer Aiello [aut, cre, cph],
Wei-Chen Chen [ctb],
Stephan Brumme [cph],
Jake Luciani [cph],
Tony Plate [cph],
Matthieu Decorde [cph],
RStudio (Hadley Wickham) [cph],
Sylvain Corlay [cph],
Johan Mabille [cph],
Niels Lohmann [cph]

Maintainer Spencer Aiello <spnrpa@gmail.com>

Repository CRAN

Date/Publication 2018-02-05 17:25:56 UTC

R topics documented:

`bootKernel` *Boot the Juniper Kernel*

Description

Juniper Kernel Booter

Usage

```
bootKernel()
```

Details

This method is invoked programatically by a Jupyter client. In fact, it's this method that appears in the `kernels.json` file at the install location of the Juniper kernel. This kernel expects that a Jupyter client will pass the connection file via the command line. A connection file contains json that is similar to:

```
{
  "stdin_port": 61144,
  "transport": "tcp",
  "ip": "127.0.0.1",
  "iopub_port": 61143,
  "hb_port": 61146,
  "key": "cc496d37-59a9-4c61-8900-d826985f564d",
  "shell_port": 61142,
  "control_port": 61145
}
```

This file is generated by the Jupyter client in the install directory of the kernel. For example, if the Juniper kernel were installed on macOS with the `--user` flag, the generated json file would have a file path of ~/Library/Jupyter/runtime/kernel-7a172737-797e-4b90-9e81-720eb8b999ab.json

See http://jupyter-client.readthedocs.io/en/latest/kernels.html#connection-files for more details.

Examples

```
## Not run:
  /path/to/R -e 'JuniperKernel::bootKernel()' --args /path/to/connection_file.json

## End(Not run)
```

`comm_close` *Comm Close*

Description

Handler for the comm_close Message Type

Usage

```
comm_close(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized comm_close message JSON.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#tearing-down-comms

Examples

```
## Not run:
  request_msg <- list("comm_id"="my_open_comm_id", "data"=list())
  comm_close(request_msg)

## End(Not run)
```

---

comm_info_request *Comm Info Request*

Description

Handler for the comm_info_request Message Type

Usage

```
comm_info_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized comm_info_request message JSON.

Value

A list having names msg_type and content. The msg_type is comm_info_reply, which corresponds to the comm_info_request message. The content field complies with the Jupyter wire message protocol specification for comm_info_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#comm-info

Examples

```
## Not run:
  request_msg <- list("target"=NULL, "target_name"="my_open_comm_id")
  comm_info_request(request_msg)

## End(Not run)
```

---

comm_msg *Comm Msg*

Description

Handler for the comm_msg Message Type

Usage

```
comm_msg(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized comm_msg message JSON.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#comm-messages

Examples

```
## Not run:
  request_msg <- list("comm_id"="my_open_comm_id", "data"=list())
  comm_request(request_msg)

## End(Not run)
```

comm_open *Comm Open*

Description

Handler for the comm_open Message Type

Usage

```
comm_open(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized comm_open message JSON.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#opening-a-comm

Examples

```
## Not run:
  request_msg <- list("comm_id"="uniq_comm_id", "target_name"="my_comm", "data"=list())
  comm_open(request_msg)

## End(Not run)
```

complete_request *Complete Handler*

Description

Handler for the complete_request Message Type

Usage

```
complete_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized complete_request message JSON.

Value

A list having names msg_type and content. The msg_type is complete_reply, which corresponds to the complete_request message. The content field complies with the Jupyter wire message protocol specification for complete_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#completion

Examples

```
## Not run:
  request_msg <- list("code"="print(\"hello\")", cursor_pos=4)
  complete_request(request_msg)

## End(Not run)
```

---

defaultDisplayName *Create the Default Kernel Display Name*

Description

Create Kernel Display Name

Usage

```
defaultDisplayName()
```

See Also

installJuniper.

Examples

```
## Not run:
  defaultDisplayName()

## End(Not run)
```

defaultKernelName *Create the Default Kernel Name*

Description

Create Kernel Name

Usage

```
defaultKernelName()
```

See Also

installJuniper.

Examples

```
## Not run:
  defaultKernelName()

## End(Not run)
```

doRequest *Handle Jupyter Requests*

Description

Top-level Request Driver for R

Usage

```
doRequest(handler, request_msg)
```

Arguments

handler	An R method that handles the message type of request_msg. This function is passed in by the RequestServer, which handles all of generic message handling such as validation and routing. This handler is one of c('kernel_info_request', 'execute_request', 'inspect_request', 'complete_request', 'history_request', 'is_complete_request', 'comm_info_request', 'comm_open', 'comm_close', 'comm_msg', 'shutdown_request').
request_msg	A list passed in from RequestServer representing the deserialized message JSON.

Details

All client requests are eventually funneled through this top-level request driver via the RequestServer. It is the job of the RequestServer to inspect messages and invoke doRequest with the appropriate handler. In other words, doRequest focuses purely on redirecting stdout/stderr and calling the handler. Message streams are detoured to a socketConnection hosted in the current thread and connected to by a separate thread polling on a ZMQ_STREAM socket. These details are all handled by the RequestServer, and all doRequest does is sink messages to the socket and perform cleanup. The port is passed as part of the request_msg, and is chosen randomly by the RequestServer.

Value

A list having names msg_type and content. The msg_type is the reply type corresponding to the request_msg's message type. For example, a kernel_info_request message produces a list with msg_type=kernel_info_reply. The content field of this list is dictated by the Jupyter wire message protocol. Note that the full reply to a Jupyter client is managed by the RequestServer.

Author(s)

Spencer Aiello

Examples

```
## Not run:
  handler <- execute_request
  request_msg <- list(stream_out_port=54321, stream_err_port=54322, list(code="rnorm(1000)"))
  doRequest(handler, request_msg)

## End(Not run)
```

---

execute_request *Execute Handler*

Description

Handler for the execute_request Message Type

Usage

```
execute_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized execute_request message JSON.

Value

A list having names msg_type and content. The msg_type is execute_reply, which corresponds to the execute_request message. The content field complies with the Jupyter wire message protocol specification for execute_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#execution-results

Examples

```
## Not run:
  request_msg <- list("code"="5+5")
  execute_request(request_msg)

## End(Not run)
```

---

history_request *History Request Handler*

Description

Handler for the history_request Message Type

Usage

```
history_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized history_request message JSON.

Value

A list having names msg_type and content. The msg_type is history_reply, which corresponds to the history_request message. The content field complies with the Jupyter wire message protocol specification for history_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#history

---

`inspect_request` *Inspect Handler*

Description

Handler for the inspect_request Message Type

Usage

```
inspect_request(request_msg)
```

Arguments

`request_msg`	A list passed in from `doRequest` representing the deserialized `inspect_request` message JSON.

Value

A list having names `msg_type` and `content`. The `msg_type` is `inspect_reply`, which corresponds to the `inspect_request` message. The `content` field complies with the Jupyter wire message protocol specification for `inspect_reply` messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#introspection

Examples

```
## Not run:
  request_msg <- list(code="print(99+rnorm(1))", cursor_pos=11, detail_level=0)
  inspect_request(request_msg)

## End(Not run)
```

`installJuniper` *Install the Juniper Kernel for Jupyter*

Description

Install Juniper Kernel

Usage

```
installJuniper(useJupyterDefault = FALSE, kernelName = defaultKernelName(),
  displayName = defaultDisplayName(), prefix = "")
```

Arguments

`useJupyterDefault`	If `TRUE`, install the kernel in a default Jupyter kernel location fashion. For macOS, the default Jupyter kernel location is `~/Library/Jupyter/kernels`. For Windows, the default Jupyter kernel location is `%APPDATA%\jupyter\kernels`. For Linux this directory is `~/.local/share/jupyter/kernels`. If `FALSE`, the kernel is installed system-wide. For unix-based machines, the system-level directory is `/usr/share/jupyter/kernels` or `/usr/local/share/jupyter/kernels`. For Windows, the location is `%PROGRAMDATA%\jupyter\kernels`. If the `prefix` argument is specified, then the `useJupyterDefault` parameter is ignored.
`kernelName`	A character string representing the location of the kernel. This is required to be made up of alphanumeric and `.`, `_`, `-` characters only. This is enforced with a check against this `^[a-zA-Z_][a-zA-Z0-9_.-]*$` regex. The case of this argument is always ignored and is `tolowered`; so while it's allowed to have mixed-case characters, the resulting location will not be. A warning will be issued if there is mixed-case characters. The default for this `juniper_r` concatenated with the `major.minor` version of R. For example, for R 3.4.0, the default would be `juniper_r3.4.0`.
`displayName`	A character string representing the name of the kernel in a client. There are no restrictions on the display name. The default for R 3.4.0 is `R 3.4.0 (Juniper)`.
`prefix`	A character string specifying the `virtual env` that this kernel should be installed to. The install location will be `prefix/share/jupyter/kernels`.

Details

Use this method to install the Juniper Kernel. After a successful invocation of this method, Juniper will be an available kernel for all Jupyter front-end clients (e.g., the dropdown selector in the Notebook interface). This method is essentially a wrapper on the function call `jupyter kernelspec install` with some extra configuration options. These options are detailed as the parameters below. One important note to make is that the kernel will depend the `R` environment that doing the invoking. In this way a user may install kernels for different versions of R by invoking this `installJuniper` method from each respective R. The defaults for `kernelName` and `displayName` are good for avoiding namespacing issues between versions of R, but installs having the same kernel name replace an existing kernel.

Examples

```
## Not run:
  installJuniper(useJupyterDefault = TRUE)  # install into default Jupyter kernel location

## End(Not run)
```

`is_complete_request` *Is Complete Handler*

Description

Handler for the is_complete_request Message Type

Usage

```
is_complete_request(request_msg)
```

Arguments

`request_msg`	A list passed in from `doRequest` representing the deserialized `is_complete_request` message JSON.

Value

A list having names `msg_type` and `content`. The `msg_type` is `is_complete_reply`, which corresponds to the `is_complete_request` message. The `content` field complies with the Jupyter wire message protocol specification for `is_complete_reply` messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#code-completeness

Examples

```
## Not run:
  request_msg <- list(code="print(")  'incomplete'
  is_complete_request(request_msg)

## End(Not run)

## Not run:
  request_msg <- list(code="print(5)")  # 'complete'
  is_complete_request(request_msg)

## End(Not run)
```

JuniperKernel *JuniperKernel: An R Kernel for Jupyter*

Description

JuniperKernel: An R Kernel for Jupyter

kernel_info_request *Kernel Info Request Handler*

Description

Response to the kernel_info_request Message Type

Usage

```
kernel_info_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized kernel_info_request message JSON.

Value

A list having names msg_type and content. The msg_type is kernel_info_reply, which corresponds to the kernel_info_request message. The content field complies with the Jupyter wire message protocol specification for kernel_info_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#kernel-info

Examples

```
## Not run:
  kernel_info_request(list())

## End(Not run)
```

listKernels *List Installed Jupyter Kernels*

Description

List Kernels

Usage

```
listKernels()
```

Details

Prints the currently installed kernels and their install locations.

Examples

```
## Not run:
  listKernels()

## End(Not run)
```

shutdown_request *Kernel Info Request Handler*

Description

Response to the shutdown_request Message Type

Usage

```
shutdown_request(request_msg)
```

Arguments

request_msg	A list passed in from doRequest representing the deserialized shutdown_request message JSON.

Value

A list having names msg_type and content. The msg_type is shutdown_reply, which corresponds to the shutdown_request message. The content field complies with the Jupyter wire message protocol specification for shutdown_reply messages.

Author(s)

Spencer Aiello

References

http://jupyter-client.readthedocs.io/en/latest/messaging.html#kernel-shutdown

Examples

```
## Not run:
  request_msg <- list(restart=FALSE)
  shutdown_request(request_msg)

## End(Not run)
```

---

the_xmock *The XMock*

Description

Get the xeus mock interpreter for interoperability with other projects.

Usage

```
the_xmock()
```

Author(s)

Spencer Aiello

Index