Tcl in Jupyter

Achievements and to-dos

Stefan Soberinig
On Jupyter (1)

- Jupyter is a widely used interactive literate-programming environment (Data Science, and beyond).
- A Jupyter *notebook* is both an interactive, literate-programming document and, when integrated with a "kernel", an application that executes the document.
- The notebook format uses JSON to store all of its contents in “.ipynb" files.
- A notebook is composed of cells, which can be of three types: code, Markdown, and raw. A code cell contains executable code used to produce results.
- By default, Jupyter displays text, images (PNG, JPG, and SVG), *HTML with JavaScript*, and Markdown; extensions may add to these display types.
On Jupyter (2)

- A Jupyter *kernel* executes code cells in a REPL manner.
- During the execution of a cell, the kernel communicates with Jupyter to display intermediate and final results.
- Notebooks are just one example of possible *frontends* to a Jupyter kernel; others include console applications, any HTTP or WebSocket clients, etc.
- Multiple frontends may be connected to the same kernel (e.g. a console and a notebook)!
All credits go to Mark Janssen!

Visit https://github.com/mpcjanssen/tcljupyter
Connectors and message types

Kernel and frontends communicate via five different connectors (*ZeroMQ sockets*) which realise the *Jupyter kernel messaging protocol*:

- **shell**: implements the main REPL behaviour via *action* requests/replies between one or more frontends and a given kernel (*message types*: execute, introspection, completion, history, kernel info)
- **iopub**: side effects are broadcasted from the kernel to one or more frontends (*message types*: streams for stderr and stdout, displays carry data for rendering/visualisation in the frontend)
- **control**: allows for controlling the kernel without interfering with shell actions (*message types*: shutdown, restart, debugging)
- **stdin**: kernel can request user-provided input data from the frontend
- **h(eart)!(:eat)** allows for frontends and kernels to signal their liveliness to each other;
Overview of component interactions

... using a PlantUML sequence diagram
Overview of component interactions

... using a PlantUML sequence diagram

```d
In [140]:
set seqDiagram {
    participant "Frontend" as FR
    participant "Kernel" as K
    participant "Session\nThread" as ST
    participant "Session\nInterp" as SI

    FR --> K : execute_request (via shell)
    K --> ST: handle_msg
    ST --> SI: eval
    activate SI
    SI --> ST: display
    ST --> K: display
    K --> FR: display (via iopub)
    SI --> ST: result
    deactivate SI
    ST --> K: result
    FR <-- K : execute_reply (via shell)
};;
```
In [141]: plantuml $seqDiagram

Frontend -> Kernel: execute_request (via shell)

Kernel <- Session Thread: handle_msg

Session Thread -> Session Interp: eval

Session Interp <- Session Thread: display

Session Thread <- Session Interp: result

Session Interp -> Session Thread: result

Session Thread <- Session Interp: display

Session Thread -> Kernel: display (via iopub)

Kernel <- Frontend: execute_reply (via shell)
Noteworthy Tcl features used

- Runs a child `interp` (potentially, a safe or restricted interp)
- hosted by a Tcl "userland" thread via `thread::create`.
- "Dealer" thread and "session" thread communicate via `thread::send -async`.
- Standard I/O from code cells (stdout, stderr) is indirected using channel transforms.

Tcl packages used

- `rl_json` for marshalling/ unmarshalling
- `tcllib`: `uuid` and `sha256` (for message signing)
- `Thread` to maintain the session thread
- `tclzmq` as a Tcl binding to ZeroMQ
Display Data

- Send back data computed by the code cells within the kernel to become displayed in the frontends (text, html, svg, etc.).
- An own message type at the messaging level (display_data that travels via the iopub connector).
- `tcljupyter` offers dedicated commands available to Tcl scripts in code cells to send display data to the frontend:
  - `jupyter::html`
  - `jupyter::updatehtml`
  - `jupyter::update`
Display Data

- Send back data computed by the code cells within the kernel to become displayed in the frontends (text, html, svg, etc.).
- An own message type at the messaging level (display_data that travels via the iopub connector).
- `tcljupyter` offers dedicated commands available to Tcl scripts in code cells to send display data to the frontend:
  - `jupyter::html`
  - `jupyter::updatehtml`
  - `jupyter::update`

```
In [142]: set displayId [jupyter::html {<b>Say, Tcl 9 is out!</b>}];
Say, Tcl 9 is out!
```
Display Data

- Send back data computed by the code cells within the kernel to become displayed in the frontends (text, html, svg, etc.).
- An own message type at the messaging level (display_data that travels via the iopub connector).
- `tcljupyter` offers dedicated commands available to Tcl scripts in code cells to send display data to the frontend:
  - `jupyter::html`
  - `jupyter::updatehtml`
  - `jupyter::update`

```
In [142]: set displayId [jupyter::html {<b>Say, Tcl 9 is out!</b>}];

Say, Tcl 9 is out!

In [143]: jupyter::updatehtml $displayId {<s>Say, Tcl 9 is out!</s>};
```
In [144]:

```python
set displayId [jupyter::html {
    <span>Tcl 🪄 is around the corner!</span>
}];

after 2000 [list jupyter::updatehtml $displayId {
    <b>Tcl 🪄 is around the corner!</b>
}];
```

Tcl 🪄 is around the corner!
Integrating `ticklecharts` via Display Data

See https://github.com/nico-robert/ticklecharts

```
In [145]: package req ticklecharts
Out[145]: 3.1.5
```
Example 1: Conference stats
Example 1: Conference stats

In [147]:

```python
set chart [ticklecharts::chart new]

$chart SetOptions -tooltip {
    show "True" trigger "axis"
    axisPointer {type "shadow"}
}
-legend {}
-grid {
    left "3%" right "4%"
    bottom "3%" containLabel "True"}

$chart Xaxis -data {{2022 2023}}
$chart Yaxis

$chart Add "barSeries" -name "Participants" \
    -data {{35 49}} \
    -emphasis {focus "series"}

$chart Add "barSeries" -name "Talks" \
    -data {{19 21}} \
    -emphasis {focus "series"}
```
In [148]: $chart RenderJupyter -renderer svg
Example 2: OpenACS diff stats
Example 2: OpenACS diff stats

```python
In [149]:
    set chart2 [ticklecharts::chart new]
    $chart2 Xaxis -data [list "5.9.0" "5.9.1" "5.10.0" "5.10.1"]
    $chart2 Yaxis

    $chart2 Add "lineSeries" \ 
          -data {{3658 3548 3445 2886}} \ 
          -areaStyle {}

    $chart2 Add "lineSeries" \ 
          -data {{120800 113292 215464 197060}} \ 
          -areaStyle {}

    $chart2 Add "lineSeries" -data {{97617 90507 193642 181613}} \ 
          -areaStyle {}
```
In [150]: $chart2 RenderJupyter -renderer svg
Alternative environments & kernels

- Christian Werner's Taygete Scrap Book (TSB): Tcl-based interactive, literate programming environment based on a webview frontend;
- Alternative Jupyter kernel: Is built using a Python "wrapper kernel" which reuses Tcl interp hosted by Python's Tkinter
- RStudio Rmarkdown notebooks: No Tcl integration so far (would require a knitr language engine, for instance)
Roadmap:

- Messaging infrastructure: Re-use or re-build?
  - Update tclzmq?
  - Complete tcljupyters pure-socket implementation (mind the ZeroMQ socket semantics)?
  - Use a thin wrapper kernel in Python to host a tcljupyters backend?
- Complete support for all message types (i.e., kernel functions)
- Deployment:
  - Distribution via a single executable (kit) for the main platforms plus self-installer?
  - How to deal with "wrapper kernel" in Python?
  - Batteries (tcllib, ticklecharts, tDOM, ...)
- Tests (jupyter_kernel_test) + documentation (along the way);
Summing up

*Tcl in Jupyter* ...

- contributes to the overall community goal to "making it easier for people to get and try Tcl" (Steve Landers);
- makes Tcl and its eco-system accessible to a non-Tcl audience;
- helps Tclers join the mainstream of interactive, literate programming environments;
- immediately useful to Tclers for the sake of *Tcl*ing:
  - to demonstrate your Tcl programs;
  - to create interactive presentations *(RISE)*;
  - to create interactive documentation (e.g., Arjen's Jupyter port of the Tcl tutorial)
  - as an interactive development environment
- 🤔 YOUR IDEAS? 🤔
Kudos 👏 to Tcl community members

- 👉👉👉 Mark Janssen for tcljupyter 👈👈👈
- Nico Robert for ticklecharts
- Jos Decoster for tclzmq
References