2025-09-18

Proposal Title

Pachá Doug Kelkhoff

Executive Summary

This work proposes exploration of a more complete implementation of the debug-adapter protocol ("Debug Adapter Protocol" 2024) (DAP) protocol, which we anticipate will require changes to R internals. We plan to prepare a patch to R to expand the capabilities of `options(browser.hook)` (enabled using feature flag `USE_BROWSER_HOOK`) to expose hooks at appropriate points that would permit a more comprehensive and direct implementation of the DAP.

The debug-adapter protocol is a widely supported standard, which specifies how an interactive development environment (IDE) can interact with a background process to manage language-specific debugging features. Hallmark features include breakpoint management, stepping through code, and viewing variables local to the scope being debugged. In R we have the luxury of debugging tools that are built-in to the core language, exposing all of these features at the command-line. However, this interface can be alienating to beginners, where the more visual interaction through an IDE can make these tools more accessible.

While there are existing projects that implement the debug-adapter protocol, namely `vscDebugger` ("vscDebugger" 2025) and `ark` ("Ark, an r Kernel" 2025), both are limited in their scope and bring both additional software requirements. Neither is capable of providing a debugging experience in a way that is fully decoupled from the IDE.

This work has been in slow development for some years. Author Doug Kelkhoff has produced a concept package ("Debugadapter" 2025) which proved the potential of the project. While a few challenges were identified early on, the major bottlenecks have been resolved in recent versions of R. Specifically, the introduction of the development flag `USE_BROWSER_HOOK` allows effective interaction with the state of the debugger such that we can manage the debug state natively in R, without a parent process intercepting output and without requiring a new standard for surfacing a console. However, even though this hook may make a proper debug interface possible, it is by no means convenient or stable. With only this hook, a implementation leveraging this hook is left to make many assumptions about the state of the debugger in order to present a intuitive browser-like prompt. Pursuing this work effectively requires dedicated time, closer communication with the R Core to use

and provide feedback on development features, and to thoroughly test the solution against a wide set of IDEs. To bring this project to fruition, I've partnered with Pachá. Together, we're pursuing funding to support Pachá while working on this project.

As a result of this work we plan to (1) implement a proof-of-concept DAP server that can be run as a part of an interactive R session, (2) propose changes to R's internals that would improve the ease and reliability of a DAP implementation (3) raise awareness within the R Core for the importance of having an R-native solution for this problem as a central part of any modern language ecosystem, and (4) provide feedback to the R Core team regarding the in-development tools for interfacing with the debug state.

Signatories

Project team

Doug Kelkhoff

Long time R advocate with a passion for developer tooling. Doug has been involved in R Consortium projects since 2018 when Doug joined the R Validation Hub ("R Validation Hub" 2025), which he now chairs. Doug represents Genentech, Inc. on the board of the R Consortium, however this work is of personal interest and is not affiliated with his involvement in this capacity. Doug has a history of working on deeply technical projects that interface with the internals of R including ("Typewriter" 2022) for runtime type-checking, ("Testex" 2024) for embedding testable code alongside documentation, and by implementing an R interpreter ("An Experimental Implementation of r, with Embellishments" 2024). Most relevant is his prior work on ("Debugadapter" 2025), which explored use of development features of R to support a more comprehensive implementation of the DAP.

Doug is committed to making R an accessible language that provides effective tools regardless of IDE.

Pachá

Contributors

Consulted

Lionel Henry

Davis Vaughan

The Problem

The Debug-Adapter Protocol

The debug-adapter protocol ("Debug Adapter Protocol" 2024) (DAP) is an open standard, proposed by Microsoft and popularized through VSCode. Although the R ecosystem has coallesced around a few choice IDEs, a broader look across all programming languages shows that VSCode has gathered an exceptionally large share of developers [1]. Many languages have provided an implementation of this protocol as part of language-specific VSCode extensions. Because of its wide adoption, many other IDEs and editors today support the DAP. These include, but are certainly not limited to, vim, neovim, emacs, helix, kakoune, and Eclipse IDE. Increasingly, a DAP implementation is a central part of an effective, portable language ecosystem.

[1] StackOverflow Developer Survey: "Integrated development environment"

R provides some powerful built-in debugging tools. However, they operate on the assumption that a typical user is interacting with them through an R prompt. Solutions that provide this experience in an IDE go to great lengths to either intercept and inject hidden calls to the user's R prompt or require background processes to manage debugging. Conversely, IDEs that leverage the DAP often assume that a static snippet of code is to be run and debugged, building a debugging experience around non-interactive code. We believe that the comfort and flexibility of R's command-line prompt can live alongside the DAP, allowing the DAP to attach to a running R session to use for debugging by hooking into the existing browser prompt.

Prior Art, Remaining Gaps

This approach is novel among existing implementations of the DAP, both in `vscDebugger` ("vscDebugger" 2025) and `ark` ("Ark, an r Kernel" 2025), which both come with additional software requirements and tight coupling to their respective IDEs. `vscDebugger` is designed to work specifically with VSCode and relies on having a running background R process which serves to intercept the debug prompt output and parse its content to resolve DAP requests. This mechanism is limited by what information is exposed in the debugger console output and can not attach to a running user session, meaning that any persistent state that R users might expect from their session is lost when debugging. `ark` addresses this problem by bundling the R kernel with the DAP server. However, this introduces another challenge because it now requires the IDE to communicate with what is effectively a new standard for serving both an interactive prompt and DAP server. This project aims to address both of these challenges by building an implementation that is R-native and can run within an active R session.

Notable hurdles remaining in these projects include:

- `ark`'s lack of support for breakpoints. `dap_server.rs#L349-351`

  > `README.md`
  > Note: Support for breakpoints is currently missing but you can use `debug()`, `debugonce()`, or `browser()` to drop into the debugger.

  it is worth noting that this feature is planned for the near future, though notable challenges remain with breakpoint management in code without ***`srcrefs`***

- `vscDebugger` recognizes that current limitations with the `browser()` prompt are a bottleneck to providing a pure R solution. `vscDebugger #33`

- `vscDebugger`'s implementation of attach mode (the DAPs mode of interacting with an existing interactive session) requires special care to manage debug state. `vscDebugger #193`

This work is not intended to fully replace these projects, but rather to provide an alternative that prioritizes IDE agnosticism and works with the R Core to address any hurdles which prevent a complete DAP implementation. While convergence of approaches would be ideal, this work would be considered successful if it consolidated open implementation questions through a central channel and aligned projects on solutions that would resolve these challenges across the board.

Possible R Core Changes

Although this project isn't predicated on changes to the R language itself, it is very possible that this work may inform future features for R's browser prompt, hooks and tracing functionality. Notably, R should be able to:

1. Run code upon pausing and providing a browser prompt to the user. We intend to execute code, opaque to the user, the sends information about the current frame back to the IDE.
2. Run code upon resuming execution. This will be used to signal that the IDE should visually convey that code is currently executing.
3. Set breakpoints in R functions in a way that does not interfere with the way that expression-level or line-level breakpoint placement is synchronized with the IDE.
4. Provide a mechanism of stepping into R code without a source file.

We hope to explore all of these capabilities, implement the best possible solution given R's current state and if necessary, propose changes that might make them more actionable in the future.

For development, we plan to use a version of R which enables the `USE_BROWSER_HOOK` flag, and possibly extend this feature to support additional code execution points. Seeing as this feature has been gated by a compilation flag, we plan to provide feedback about how this feature enables the implementation of a more integrated debugging experience.

The proposal

Overview

R's interactive prompt is the bedrock of R as an analytic language, allowing for experimentation in the context of code without long compilation steps and static scripts. A central part of this workflow is the debugger, which is provided with the language and allows for exceptionally dynamic interruption, modification and experimentation with code as it is running.

Despite R's long standing support for interactive debugging, the user experience has not changed significantly, largely resembling the debug experience of Common Lisp. Although improving, it remains difficult to build custom tooling around the existing R debugging experience. The introduction of *browser hooks*, specifically, have provided a mechanism to inject additional code which can allow for arbitrarily complex actions to be taken during debugging. These tools offer the possibility of overcoming some of the challenges of introspecting the debug frame that have had to be painstakingly engineered around in projects such as `vscDebugger` and `ark`, requiring a wrapping process that attempts to extract information from command-line output or an overseer that manages communication between an R process and DAP server.

We are uniquely positioned to test new R features and provide active feedback to the R Core to drive the debugging experience forward. We plan to implement an R-native debug adapter server that is truly IDE agnostic while improving on the user experience by leveraging recent R enhancements related to the `browser()`.

Detail

We plan to implement a debugger experience that extends the R `browser()` to the IDE, mirroring state and allowing for interaction through either interface. When implemented, a user may:

1. Use an IDE for standard debugging operations, such as setting breakpoints and stepping through code.
2. Step through the code in a running R Session, reflecting the debugger status in the IDE.
3. Inspect the debug frame from the IDE, or in a running R session.
4. Use a stateful R Session when debugging, allowing, for example, masked functions, variables and search path modification during debugging, in a way that is similar to the familiar browser prompt.
5. Use the DAP with any IDE that supports the protocol.
6. Install the DAP as an R package for easier distribution.
7. Build the DAP R package with minimal system dependencies.

We expect that this will require contributions to the R language itself, which we plan to first implement as a standalone, patched version of R with minimal changes in support of piloting the proposed capabilities. To proove the utility of these capabilities, we will develop a user-facing R package that implements a DAP server.

Minimum Viable Product

Although our goal is to explore the limitations of a full DAP implementation in R, providing a *useful* R-native DAP server is well within reach.

`debugadapter` ("Debugadapter" 2025) initially hit four major roadblocks that prevented the implementation of central DAP features.

- First, the DAP server should run in the background so that it remains responsive as the user continues to interact with the prompt. This challenge has been largely resolved by a transition to using the `later` package's event loop to handle messages in the background.
- Second, it should hook into the `browser()` prompt to emit DAP messages. This has been largely solved by use of the `browser.hook` option in R's devel version.
- Third, that existing browser hooks require cumbersome acrobatics to effectively tease out all the necessary debugging information at the appropriate time.
- Fourth, many breakpoints are difficult to set given that many copies of functions exist throughout an active R session and that not all code has proper source references.

A minimum viable product would require:

1. That a DAP hosted in a user's R process be able to handle DAP messages even while R is idling. We plan to use `later` to continue listening for messages.
2. Use the existing `browser.hook` to handle as much of the DAP protocol as possible.
3. Identify gaps that are either untennable or overly complicated to implement using only the existing `browser.hook`.

We expect to additionally deliver:

1. A mapping of the DAP protocol to the appropriate place in the `browser()` REPL where we would ideally be able to provide protocol responses.
2. A patched version of R that exposes hooks for executing code at these critical points.
3. An implementation of a DAP server as an R package to pilot these R internal changes.

Beyond our expectations, next steps would include:

1. Further developing the patched version of R to expose C-level callbacks in addition to R-level hooks.
2. Exploration of ways that R may internally help to manage an R session's set breakpoints.
3. R internal support for breakpoints that affect code without a known source reference.
4. Additional options for customizing the browser prompt text (ie, `Browse>`) allowing for more informative messaging to the user to indicate when code is executing or paused at a breakpoint.

Architecture

The majority of the DAP protocol itself is clearly specified. Because R provides a level of interactivity that isn't clearly defined within the scope of the DAP, some design details are left open for interpretation.

We plan to build a DAP server, whose architecture relies on the following core features:

1. A running, interactive R session, which manages interaction with a DAP server using a `{later}` loop, executed periodically while the R process is idle. We aim to make this transparent to a user, who may wish to start the DAP listener on startup as part of a `.RProfile`.
2. Various additional browser hooks, implemented as part of the R language, which can support retrieval of the breadth of the DAP request data.
3. Various R callback functions which hook into the `browser()` REPL's loop to listen and respond to DAP client requests.

These central pieces would amount to a native R DAP server implementation, which run in the background of a user-facing interactive R session. Our hope is that this allows for an experience akin to R's native `browser()` prompt, but with all the visual elements communicated through the supported UI of a user's preferred IDE.

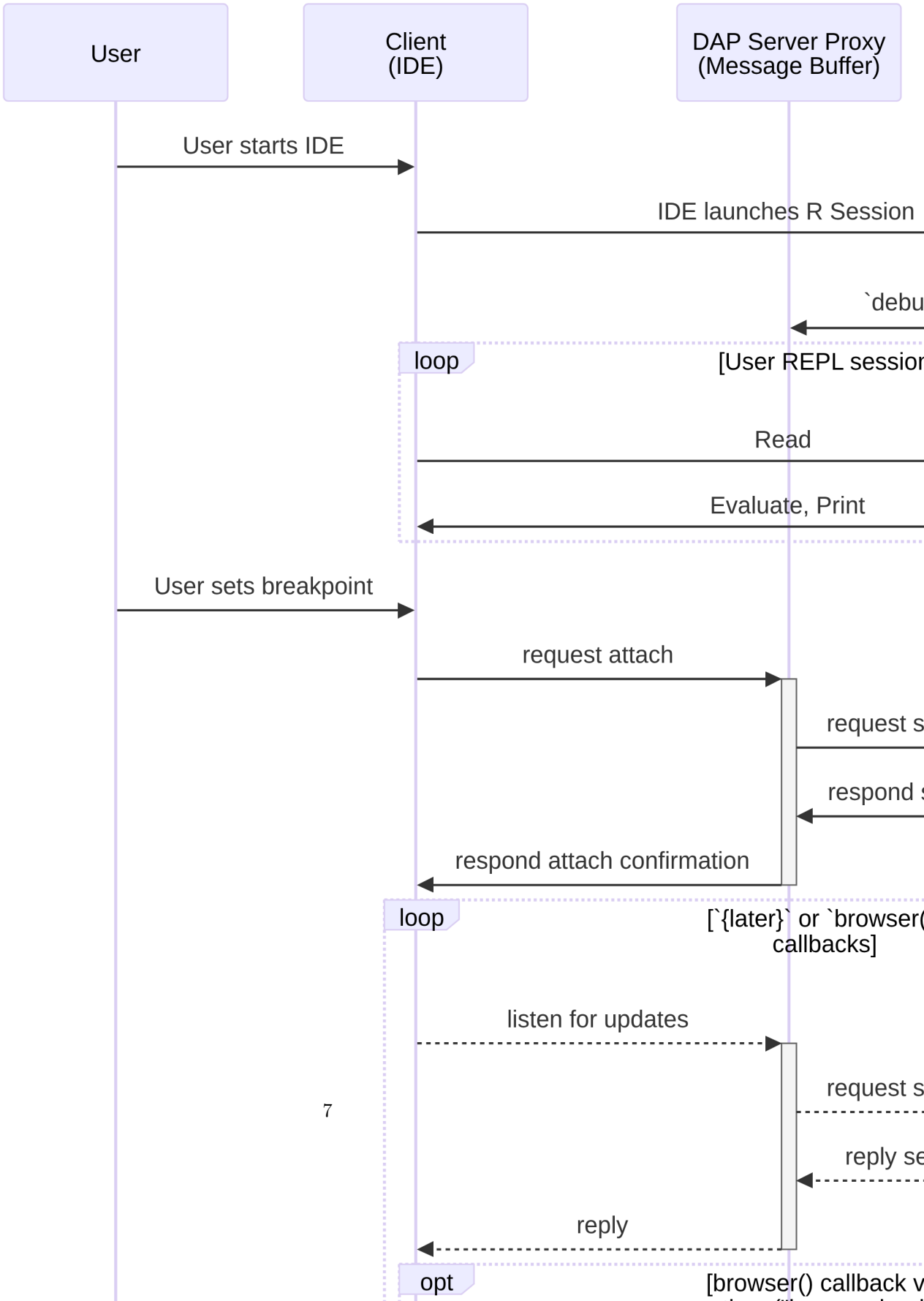
User
Client (IDE)
DAP Server Proxy (Message Buffer)
User starts IDE
IDE launches R Session
`debu
loop
[User REPL session
Read
Evaluate, Print
User sets breakpoint
request attach
request s
respond s
respond attach confirmation
loop
[`{later}` or `browser(
callbacks]
listen for updates
request s
reply se
reply
opt
[browser() callback v

Figure: A specialized architecture for an *attach mode* DAP server. Note that an alternative *launch mode* server would leverage many of the same components, differentiated only in that it doesn't attach to an existing R session.

Assumptions

There are no assumptions that would entirely invalidate this project, though there are assumptions which may limit its impact:

1. To make this work most effective, we hope that there is interest in the R Core for continuing to support and stabilize browser hooks.
2. We assume that `ark` continues to be limited by base R capabilities to permit effective breakpoint injection, and that our development here would help pave the way for both projects.
3. We assume that `ark` intends to continue being shipped as a standalone binary, requiring that users go beyond a simple package install to enable a debugger, leaving a gap for an implemntation as a more conventionally distributed R extension.
4. We assume that `ark` will continue to be provided as a combined Jupyter Kernel and DAP implementation, making it less accessible to IDEs beyond Positron, or limited by adoption of this protocol.
5. We assume that the goals of `vscDebugger` continue to prioritize VSCode, leaving a gap for a more agnostic solution.

External dependencies

- To improve the user experience, we plan to depend on R built with the `USE_BROWSER_HOOKS` flag enabled, allowing us to inject code when a browser prompt is hit. Without this feature, users may need to manually synchronize their debug state.
- `{later}`, to facilitate arbitrary code execution while the user's R session idles, allowing us to synchronize debug state without user interaction.
- `vscDebugger` and `ark`, though not dependencies, are similar projects and we plan to make sure any learnings from our project are distributed to these similar projects.

Project plan

Start-up phase

In some ways, this project started years ago. We already have a proof of concept implementation of the DAP using devel feature flags. Since then, we have contacted Posit to align on shortcomings and plan features that are preventing a more comprehensive solution in both Posit and other IDEs.

If funded, the first steps will be to inspect the DAP specification and map its various request messages to the place in the browser REPL where that information is best retreived. This map will inform which additional hooks may be necessary for a complete implementation and will help to prioritize further development.

After this planning phase, we will be able to provide a more clear outline of capabilities that would be necessary within R, which we plan to share with R core. Assuming that the R core team will not be immediately ready to jump into this development, we plan to advance the project as a patch that can support a proof-of-concept implementation of the DAP.

Technical delivery

Planning Deliverables (through week 1)

- Providing a mapping of the DAP specification to where in the R REPL loop the requested information is more meaningfully gathered. Subject to change, this is expected to be best represented as a workflow diagram that can be used as reference for other projects.
- Signatories will familiarize themselves more deeply with existing implementations and scope development next steps. This will largely be organized through issues filed alongside a source control project.

Preparatory Development (through month 1)

- Implement the necessary browser hooks in a patched version of R to fully cover the implementation of a DAP.
- Explore using `later` to listen for DAP requests while R idles.

DAP Server Proof-of-Concept Development (through month 3)

- Continue with the implementation of a proof-of-concept DAP server, implemented as an R package. Aim to support all reasonably achievable parts of the DAP specification.
- Share patched version of R for broader feedback. With support from R core, invite review and further development.
- Feature communication (see below)

Other aspects

Communication during development

As this is largely exploratory work, our primary stakeholders are active DAP developers. We plan to make materials broadly available, but with an emphasis on distributing to this particular audience. This includes our mapping of the DAP specification to the `browser()` REPL loop, R internal changes and pilot server implementation leveraging these changes.

Broader communication

If we garner positive feedback from R Core developers which would indicate a high likelihood of adoption of our proposed patch changes, we would want to write a technical blog post about the challenges faced across implementations, and how the ISC process provided a vehicle for an agnostic path forward.

Budget & funding plan

We are proposing work that we expect to take approximately three months. Given the availability of the consulting signatories, we are requesting funds to support one developer, Mauricio, at 50% time for three months. Given the technical breadth and depth of the work, we are requesting **USD $** to support them through this project.

This number was selected after reviewing previous years' ISC grant recipients and balancing a reasonably scoped ISC grant against salary expectations for a North America-based early-career developer.

Success

Measuring success

This work has a lot of room to grow well beyond what is achievable within three months.

We would consider this step of the process successful if we can provide a clear path to the necessary R internal changes required for the implementation of a DAP server.

Definition of done

Although aligning on a path forward is largely a communication goal, being confident in that path requires experimentation. We would consider our part done after we have proposed changes implemented in a patch to R, and have developed a pilot implementation to the point that we can be confident that it would be able to fully cover any reasonable parts of the DAP specification.

Ideally, users in any editor that supports the DAP would have a realistic path to communicating with our pilot project as the DAP server, and would be able to interact with an R session in a way that feels native to R.

Future work

We do not expect a patch that is developed primarily in service of exploration to be fully mature within these three months. To bring this work to fruition in R, our hope is that this work displays the importance of R internal support which would excite a more experienced R core developer to help us integrate this work into the language properly.

Moreover, we expect that broader review of our changes may introduce additional requirements. For example, discussions with the `ark` team have raised the importance of a C-level API to these injection points as a preferred interface from the `ark`.

Further, R's mechanism of attaching namespaces – and the proliferation of unique namespaces, especially through imported namespaces within loaded packages – means that setting breakpoints is a non-trivial task. Rigorously handling the synchronization of IDE-set breakpoints with browser, or traced R calls would likely require additional tooling within the R language itself. We do not plan to pursue this as part of this work, though it is necessary for a intuitive debugging experience when debugging in pathological cases.

"An Experimental Implementation of r, with Embellishments." 2024. github.com/dgkf. https://github.com/dgkf/R.
"Ark, an r Kernel." 2025. Posit. https://github.com/posit-dev/ark.
"Debug Adapter Protocol." 2024. Microsoft. https://microsoft.github.io/debug-adapter-protocol/.
"Debugadapter." 2025. codeberg.org/dgkf. https://codeberg.org/dgkf/debugadapter.
"R Validation Hub." 2025. R Validation Hub. https://pharmar.org/.
"Testex." 2024. github.com/dgkf. https://github.com/dgkf/testex.
"Typewriter." 2022. github.com/dgkf. https://github.com/dgkf/typewriter.
"vscDebugger." 2025. github.com/ManuelHentschel. https://manuelhentschel.github.io/vscDebugger/.