This page is for historical reference.

UI protocol improvements (DONE)

Desirable Skills:

- C and related tools
- Familiar with event-loop programming model

Description:

Nvim GUI:s are implemented as processes communicating with Nvim over a protocol. Currently this protocol exposes most functionality as a terminal-like screen grid. A long term goal is enabling richer UIs (including embeddings in GUI editors, like VSCode) by refactoring the protocol towards semantic updates and letting the GUI actually draw buffer contents and other screen elements. Currently this has been implemented for a few specific elements, like the completion popup menu and the command line.

Expected Result:

The UI protocol has gained new capabilities. This could involve substantial changes such as the GUI receiving redraw updates for each window separately, so that the GUI could be responsible for managing the overall layout of windows and statuslines.

Alternatively, improvements could be done within the current global screen grid, such as the ability to display grid-aligned images in signs, buffers and statuslines. It could also involve adding semantic information to the grid, so that GUI:s can identify screen elements reliably rather than guessing it from highlights.

- Code license: Apache 2.0

Difficulty:

Medium to Hard

Student: (@UtkarshMe)

Mentor: Björn Linse (@bfredl)

Java client

Desirable Skills:

- Familiar with Vim/Nvim and Vim script (VimL)
- Moderate/High experience in Java
- Familiar with event-loop programming model

Description:

Implement a Nvim API client using Java.

Implement a client, written in Java, which allows Java applications to control Nvim using the Nvim RPC API. If you are familiar with AWS or any other SaaS, note that a Nvim API client is just like a SDK for a REST web service, except that Nvim uses msgpack, not HTTP/JSON.

The Nvmi RPC API is documented at :help api and :help rpc.

To correctly implement the client one needs to understand the msgpack-rpc protocol. Some sort of event-loop mechanism will be needed to handle notifications.

For reference, you can find clients in other languages at the related projects wiki page.

The ultimate goal is to have a library that can be used to create plugins for IntelliJ and Eclipse. **Minimizing third-party dependencies** may help there.

Expected Result:

- Java library that can be used to build Neovim extensions (UIs and other applications).
  - Method signatures generated from `nvim --api-info`.
  - Since `nvim` is already required (for `--api-info`), the Java code-generation script could be written in Lua (which is built-in to `nvim`).
- Passes the test suite used by the Nvim python-client.
  - Using the python-client tests as a guide, create equivalent tests using a Java testing framework.
  - Test suite should be runnable from the command-line (should not require an IDE) via maven/gradle (or some other industry-standard build-tool).
- Builds (and passes tests) on Linux (Travis CI) and Windows (AppVeyor)
- End-user deliverable should be compatible Java 6 (this is negotiable)
- Source-code can be latest version of Java (no backwards-compatibility requirement)
- Code license: Apache 2.0

Difficulty:

Medium

Mentor: Justin M Keyes (@justinmk)

C# client (DONE)

Desirable Skills:

- Familiar with Vim/Nvim and Vim script (VimL)
- Moderate/High experience in C#
- Familiar with event-loop programming model

Description:

Implement a Nvim API client using C#.

Implement a client, written in C#, which allows C# applications to control Nvim using the Nvim RPC API. If you are familiar with AWS or any other SaaS, note that a Nvim API client is just like a SDK for a REST web service, except that Nvim uses msgpack, not HTTP/JSON.

The Nvmi RPC API is documented at :help api and :help rpc.

To correctly implement the client one needs to understand the msgpack-rpc protocol. Some sort of event-loop mechanism will be needed to handle notifications (hint1, hint2).

For reference, you can find clients in other languages at the related projects wiki page.

The ultimate goal is to have a library that can be used to create plugins for Visual Studio. **Minimizing third-party dependencies** may help there.

Expected Result:

- C# library that can be used to create C#-based Neovim extensions (UIs and other applications).

    - Method signatures generated from `nvim --api-info`.
    - Since `nvim` is already required (for `--api-info`), the C# code-generation build script could be written in Lua (which is built-in to `nvim`).
- Passes the test suite used by the Nvim python-client.
    - Using the python-client tests as a guide, create equivalent tests using a C# testing framework.
    - Test suite should be runnable from the command-line (should not require an IDE) via MSBuild or some other industry-standard build-tool.
- Builds (and passes tests) on Linux (Travis CI) and Windows (AppVeyor)
- Builds against **.NET Standard 2.0**
- Deliverable should be easy to install as a NuGet (or other) package.
- Source-code can be latest version of C# (no backwards-compatibility requirement)
- Code license: Apache 2.0

Difficulty:

Medium

Student: (@b-r-o-c-k)

Mentor: Justin M Keyes (@justinmk)