[RFC] Enable Widgets in Product

Summary: Allow community to build widgets to expand the use cases of product.

Created: Jan 22, 2019
Current Version: 1.0.4
Target Version: 1.1.0
PRD: Link to PRD if applicable

Status: WIP | In-Review | Approved | Obsolete

Owner: email@hashicorp.com
Contributors: email@hashicorp.com
Other stakeholders: email@hashicorp.com
Approvers: person@hashicorp.com

The RFC begins with a brief overview. This section should be one or two paragraphs that just explains what the goal of this RFC is going to be, but without diving too deeply into the "why", "why now", "how", etc. Ensure anyone opening the document will form a clear understanding of the RFCs intent from reading this paragraph(s).

Background

The next section is the "Background" section. This section should be at least two paragraphs and can take up to a whole page in some cases. The **guiding goal of the background section** is: as a newcomer to this project (new employee, team transfer), can I read the background section and follow any links to get the full context of why this change is necessary?

If you can't show a random engineer the background section and have them acquire nearly full context on the necessity for the RFC, then the background section is not full enough. To help achieve this, link to prior RFCs, discussions, and more here as necessary to provide context so you don't have to simply repeat yourself.

Proposal

The next required section is "Proposal" or "Goal". Given the background above, this section proposes a solution. This should be an overview of the "how" for the solution, but for details further sections will be used.

Abandoned Ideas (Optional)

As RFCs evolve, it is common that there are ideas that are abandoned. Rather than simply deleting them from the document, you should try to organize them into sections that make it clear they're abandoned while explaining *why* they were abandoned.

When sharing your RFC with others or having someone look back on your RFC in the future, it is common to walk the same path and fall into the same pitfalls that we've since matured from. Abandoned ideas are a way to recognize that path and explain the pitfalls and why they were abandoned.

Sections (Heading 2)

From this point onwards, the sections and headers are generally freeform depending on the RFC. Sections are styled as "Heading 2". Try to organize your information into self-contained sections that answer some critical question, and organize your sections into an order that builds up knowledge necessary (rather than forcing a reader to jump around to gain context).

Sections often are split further into sub-sections styled "Heading 3". These sub-sections just further help to organize data to ease reading and discussion.

[Example] Implementation

Many RFCs have an "implementation" section which details how the implementation will work. This section should explain the rough API changes (internal and external), package changes, etc. The goal is to give an idea to reviews about the subsystems that require change and the surface area of those changes.

This knowledge can result in recommendations for alternate approaches that perhaps are idiomatic to the project or result in less packages touched. Or, it may result in the realization that the proposed solution in this RFC is too complex given the problem.

For the RFC author, typing out the implementation in a high-level often serves as "rubber duck debugging" and you can catch a lot of issues or unknown unknowns prior to writing any real code.

[Example] UX

If there are user-impacting changes by this RFC, it is important to have a "UI/UX" section. User-impacting changes include external API changes, configuration format changes, CLI output changes, etc.

This section is effectively the "implementation" section for the user experience. The goal is to explain the changes necessary, any impacts to backwards compatibility, any impacts to normal workflow, etc.

As a reviewer, this section should be checked to see if the proposed changes *feel* like the project in question. For example, if the UX changes are proposing a flag "-foo_bar" but all our flags use hyphens like "-foo-bar", then that is a noteworthy review comment. Further, if the

breaking changes are intolerable or there is a way to make a change while preserving compatibility, that should be explored.

[Example] UI

Will this RFC have implications for the web UI? If so, be sure to collaborate with a frontend engineer and/or product designer. They can add UI design assets (user flows, wireframes, mockups or prototypes) to this document, and if changes are substantial, they may wish to create a separate RFC to dive further into details on the UI changes.

Style Notes

All RFCs should follow similar styling and structure to ease reading. "Beautiful is better" is a core principle of HashiCorp and we care about the details.

Heading Styles

"Heading 2" should be used for section titles. We *do not use* "Heading 1" because aesthetically the text is too large. Google Docs will use Heading 2 as the outermost headers in the generated outline.

"Heading 3" should be used for sub-sections.

Further heading styles can be used for nested sections, however it is rare that a RFC goes beyond "Heading 4," and rare itself that "Heading 4" is reached.

Lists

When making lists, it is common to bold the first phrase/sentence/word to bring some category or point to attention. For example, a list of API considerations:

- **Format** should be widgets
- **Protocol** should be widgets-rpc
- **Backwards compatibility** should be considered.

Typeface

Type size should use this template's default configuration (11pt for body text, larger for headings), and the type family should be Arial. No other typeface customization (eg color, highlight) should be made other than italics, bold, and underline.

Code Samples

Code samples should be indented (tab or spaces are fine as long as it is consistent) text using the Courier New font. Syntax highlighting can be included if possible but isn't necessary. Please

ensure the highlighted syntax is the proper font size and using the font Courier New so non-highlighted samples don't appear out of place.

CLI output samples are similar to code samples but should be highlighted with the color they'll output if it is known so that the RFC could also cover formatting as part of the user experience.

```
func example() {
  <-make(chan struct{})
}
```