Ethereum Classic Improvement Proposals

ECIP 1000: ECIP Process

Author	Wei Tang
Discussions-To	https://github.com/ethereumclassic/ECIPs/issues/58
Status	Active
Type	Meta
Created	2017-06-29

Abstract

An Ethereum Classic Improvement Proposal (ECIP) is a design document providing information to the Ethereum Classic community, or describing a new feature for Ethereum Classic or its processes or environment. The ECIP should provide a concise technical specification of the feature and a rationale for the feature.

We intend ECIPs to be the primary mechanism for proposing new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Ethereum Classic. The **ECIP author** is responsible for building consensus within the community and documenting dissenting opinions.

Because the ECIPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal.

Copyright

ECIP Workflow

Introduction

The ECIP process begins with a new idea for Ethereum Classic. Each potential ECIP must have **a champion** – someone who writes the ECIP using the style and format described below, shepherds the discussions in the appropriate forums, and attempts to build community consensus around the idea. The **ECIP champion (a.k.a. Author)** should first attempt to ascertain whether the idea is ECIP-able.

Small enhancements or patches to a particular piece of software often don't require standardization between multiple projects; these don't need an ECIP and should be injected into the relevant project-specific development workflow with a patch submission to the applicable issue tracker.

Additionally, many ideas have been brought forward for changing Ethereum Classic that have been rejected for various reasons.

Steps

1: The first step should be to search past discussions to see if an idea has been considered before, and if so, what issues arose in its progression.

After investigating past work, the best way to proceed is by posting about the new idea to the Ethereum Classic Discord #ecips Channel.

Vetting an idea publicly before going as far as writing a ECIP is meant to save both the potential author and the **wider community** time.

Asking the Ethereum Classic community first if an idea is original helps prevent too much time being spent on something that is guaranteed to be rejected based on prior discussions (searching the internet does not always do the trick).

It also helps to make sure the idea is applicable to the entire community and not just the author. Just because an idea sounds good to the author does not mean it will work for most people in most areas where Ethereum Classic is used.

2: Once the champion has asked the Ethereum Classic community as to whether an idea has any chance of acceptance, a **draft ECIP** should be submitted to the ECIPs git repository as a pull request. This gives the author a chance to flesh out the draft ECIP to make it properly formatted, of high quality, and to address additional concerns about the proposal. This draft must be written in ECIP style as described below, and named with an alias such as "ecip-johndoe-infinitecoins" until an editor has assigned it an ECIP number (authors MUST NOT self-assign ECIP numbers).

ECIP authors **are responsible for collecting community feedback** on both the initial idea and the ECIP before submitting it for review. However, wherever possible, long open-ended discussions on public groups or mailing lists should be avoided. Strategies to keep the discussions efficient include: setting up a separate SIG mailing list for the topic, having the ECIP author accept private comments in the early design phases, setting up a wiki page or git repository, etc. ECIP authors should use their discretion here.

It is highly recommended that **a single ECIP contain a single key proposal or new idea**. The more focused the ECIP, the more successful it tends to be. If in doubt, split your ECIP into several well-focused ones. Please check out the ECIPs in this repository and search for similar ones that have already been created.

3: When the ECIP draft is complete, an ECIP editor will assign the ECIP a number, label it as Standards Track, Informational, or Process, and merge the pull request to the ECIPs git repository.

The ECIP editor will not unreasonably reject an ECIP.

Reasons for rejecting ECIPs include duplication of effort, disregard for formatting rules, being too unfocused or too broad, being technically unsound, not providing proper motivation or addressing backwards compatibility, or not in keeping with the Ethereum Classic philosophy.

For a ECIP to be accepted it must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement. The proposed implementation, if applicable, must be solid and must not complicate the protocol unduly.

4: During the life cycle of the ECIP, the author may update the draft as necessary in the git repository. **Updates to drafts should also be submitted by the author as pull requests.**

Transferring ECIP Ownership

It occasionally becomes necessary to transfer ownership of ECIPs to a **new champion**. In general, we'd like to retain the original author **as a co-author** of the transferred ECIP, but that's really up to the original author.

A good reason to transfer ownership is because the original author no longer has the time or interest in updating it or following through with the ECIP process, or has fallen off the face of the net (i.e. is unreachable or not responding to email). A bad reason to transfer ownership is because you don't agree with the direction of the ECIP. We try to build consensus around an ECIP, but if that's not possible, you can always submit a competing ECIP.

If you are interested in assuming ownership of a ECIP, send a message asking to take over, addressed to both the original author and the ECIP editor. If the original author doesn't respond to email in a timely manner, the ECIP editor will make a unilateral decision (it's not like such decisions can't be reversed :).

ECIP Editors

Current ECIP editors:

- Antoine Toulme (@atoulme)
- gitr0n1n (@gitr0n1n)
- Istora Mandiri (@IstoraMandiri)
- Mr. Meows D. Bits (@meowsbits)
- Talha Cross (@soc1c)

Former ECIP editors:

- Wei Tang (@sorpaas)
- Yaz Khoury (@YazzyYaz)
- Cody Burns (@realcodywburns)
- Stevan Lohja (@stevanlohja)
- Zachary Belford (@BelfordZ)
- Felipe Faraggi (@faraggi)

ECIP Editor Responsibilities & Workflow

Introduction

ECIP editors are intended to fulfill administrative and editorial responsibilities. ECIP editors monitor ECIP changes, and update ECIP headers as appropriate.

Steps

1: Each new ECIP should first be submitted as a **"pull request"** to the ECIPs git repository. Then, an editor does the following:

- Read the ECIP to check if it is ready: sound and complete. The ideas must make technical sense, even if it doesn't seem likely to be accepted.
- The title should accurately describe the content.
- Motivation and backward compatibility (when applicable) must be addressed.
- The defined Layer header must be correctly assigned for the given specification.
- Licensing terms must be acceptable for ECIPs.

2: If the ECIP isn't ready, the editor will send it back to the author for revision with specific instructions.

3: Once the ECIP is ready, the ECIP editor will:

- Assign a ECIP number in the pull request.
- Merge the pull request when it is ready.
- List the ECIP in [[README.mediawiki]]

ECIP Format and Structure

Specification

ECIPs should be written in **mediawiki** or **markdown** format.

Each ECIP should have the following parts:

- **Preamble** – Headers containing metadata about the ECIP see below.
- **Abstract** – A short (~200 word) description of the technical issue being addressed.

- **Copyright** – The ECIP must be explicitly licensed under acceptable copyright terms see below
- **Specification** – The technical specification should describe the syntax and semantics of any new feature. It should be detailed enough to allow competing, interoperable implementations for any of the current Ethereum Classic platforms.
- **Motivation** – The motivation is critical for ECIPs that want to change the Ethereum Classic protocol. It should clearly explain why the existing protocol is inadequate to address the problem that the ECIP solves.
- **Rationale** – The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work. It should provide evidence of consensus within the community and discuss important objections or concerns raised during discussion.
- **Backwards compatibility** – All ECIPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The ECIP must explain how the author proposes to deal with these incompatibilities.
- **Reference implementation** – The reference implementation must be completed before any ECIP is given **"Final"** status, but it need not be completed before the ECIP is **"Accepted"**. It is better to finish the specification and rationale first and reach consensus on it before writing code. The final implementation must include test code and documentation appropriate for the Ethereum Classic protocol.

ECIP Header Preamble

Each ECIP must begin with an RFC 822 style header preamble. The headers must appear in the following order. Headers marked with "*" are optional and are described below. All other headers are required.

- **ECIP:** (ECIP number, or "?" before being assigned)
-

Layer: (Consensu	Consen sus	P e e r S	A	Ap p l i c

s (soft fork)	(hard fork)	e r v i c e s		a t i o n s)

- **Title:** (ECIP title; maximum 44 characters)
- **Author:** (authors real names and email addrs)
- **Discussions-To:** (email address)
- **Comments-Summary:** (summary tone)
- **Comments-URI:** (links to wiki page for comments)
-

St a t u s : (D r a f t		A		D	R	F	W

-

Type: (Standards Track	Informational	Proces s)

- **Created:** (date created on, in ISO 8601 (yyyy-mm-dd) format)
- **License:** (abbreviation for approved license(s))
- **License-Code:** (abbreviation for code under different approved license(s))
- **Requires:** (ECIP number(s))
- **Replaces:** (ECIP number)
- **Superseded-By:** (ECIP number)

The Layer header (only for Standards Track ECIPs) documents which layer of Ethereum Classic the ECIP applies to.

The Author header lists the names and email addresses of all the authors/owners of the ECIP. The format of the Author header value must be:

Random J. User <123@dom.ain>

If there are multiple authors, each should be on a separate line following RFC 2822 continuation line conventions.

While an ECIP is in private discussions (usually during the initial Draft phase), aDiscussions-To header will indicate the mailing list or URL where the ECIP is being discussed. No Discussions-To header is necessary if the ECIP is being discussed privately with the author.

The Type header specifies the type of ECIP: Standards Track, Informational, or Process.

The Created header records the date that the ECIP was assigned a number.

ECIPs may have a Requires header, indicating the ECIP numbers that this ECIP depends on.

ECIPs may also have a Superseded-By header indicating that a ECIP has been rendered obsolete by a later document; the value is the number of the ECIP that replaces the current document. The newer ECIP must have a Replaces header containing the number of the ECIP that it rendered obsolete.

Auxiliary Files

ECIPs may include auxiliary files such as diagrams. If if requires images, the image files should be included in a subdirectory of the `assets` folder for that ECIP as follow: `assets/ecip-X`. When linking to an image in the ECIP, use the related links such as `./assets/ecip-X/image.png`.

ECIP Types

There are three types of ECIP:

- A **Standard Track ECIP** describes any change that affects most or all Ethereum Classic implementations, such as a change to the network protocol, a change in block or transaction validity rules, proposed application standards/conventions, or any change or addition that affects the interoperability of applications using Ethereum Classic. Furthermore, Standard Track ECIPs can be broken down into the following categories:
    - **Core** - improvements requiring a consensus fork, as well as changes that are not necessarily consensus critical but may be relevant to core developer discussions.
    - **Networking** - improvements to networking protocol specifications.
    - **Interface** - improvements around client [API/RPC] specifications and standards, and also certain language-level standards like method names and contract ABIs.
    - **ECBP (Ethereum Classic Best Practice)** - application-level standards and conventions, including contract standards such as token standards, name registries, URI schemes, library/package formats, and wallet formats.
- A **Meta ECIP** describes a **process** surrounding Ethereum Classic or proposes a change to a process (or an event like the next hard fork). Process ECIPs are like Standard Track ECIPs, but apply to areas other than the Ethereum Classic protocol itself. They may propose an implementation, but not to Ethereum Classic's codebase; they often require community consensus; unlike Informational ECIPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process (e.g. this ECIP-1000 process document), and changes to the tools or environment used in Ethereum Classic development. If a Meta ECIP is for a hard fork, changes are

to be updated with PRs to existing Meta ECIPs. *Any meta-ECIP is also considered a Process ECIP.*

- An **Informational ECIP** describes an Ethereum Classic design issue, or provides general guidelines or information to the Ethereum Classic community, but does not propose a new feature. Informational ECIPs do not necessarily represent Ethereum Classic community consensus or a recommendation, so users and implementors are free to ignore Informational ECIPs or follow their advice.

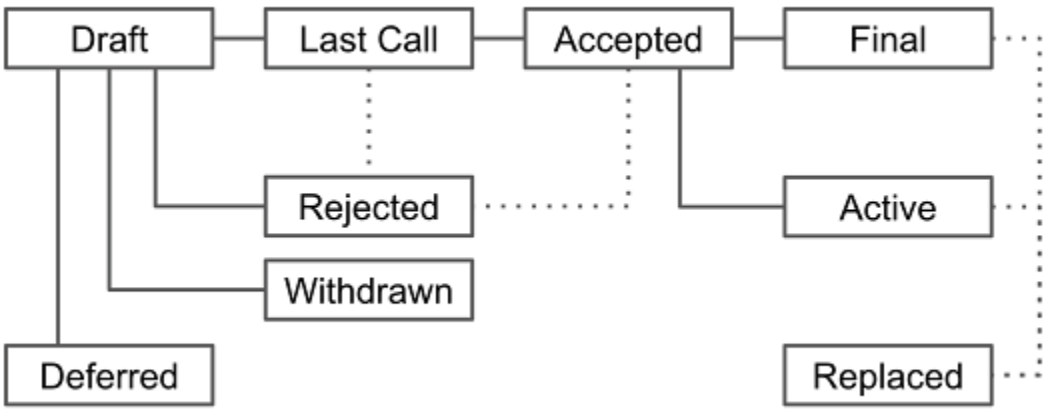
It is highly recommended that a single ECIP contain a single key proposal or new idea. The more focused the ECIP, the more successful it tends to be. A change to one client doesn't require an ECIP; a change that affects multiple clients, or defines a standard for multiple apps to use, does.

An ECIP must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement.

ECIP Status Field

Specification

The typical paths of the status of ECIPs are as follows:



Champions of an ECIP may decide on their own to change the status between Draft, Deferred, or Withdrawn.

The ECIP editor may also change the status to Deferred when no progress is being made on the ECIP.

An ECIP may only change status from Draft (or Rejected) to Last Call, when the author deems it is complete, has a working implementation (where applicable), and has community plans to progress it to Final status.

ECIPs should be changed from Draft or Last Call status, to Rejected, upon request by any person, if they have not made progress in three years. Such a ECIP may be changed to Draft status if the champion provides revisions that meaningfully address public criticism of the proposal, or to Last Call if it meets the criteria required as described in the previous paragraph.

An Accepted ECIP may progress to Final only when specific criteria reflecting real-world adoption has occurred. This is different for each ECIP depending on the nature of its proposed changes, which will be expanded on below. Evaluation of this status change should be **objectively verifiable**, and/or be discussed on the development calls, Discord channel, other groups or the mailing list.

When a Final ECIP is no longer relevant, its status may be changed to Replaced. This change must also be **objectively verifiable** and/or discussed.

Some Informational ECIPs, which are considered process ECIPs, may also be moved to a status of Active instead of Final if they are never meant to be completed, e.g. this ECIP-1000.

Draft ECIPs which may be in a very early stage may be entered as WIP ECIPs, which means they are a work in progress.

A process ECIP may change status from Draft to Final when it achieves rough consensus on the discussion process. Such a proposal is said to have rough consensus if it has been open to discussion on the development calls, Discord channel, other groups or the mailing list for at least one month, and no person maintains any unaddressed substantiated objections to it. Addressed or obstructive objections may be ignored/overruled by general agreement that they have been sufficiently addressed, but clear reasoning must be given in such circumstances.

Progression to Final status

Peer services ECIPs should be observed to be adopted by at least 1% of public listening nodes for one month.

API/RPC and application layer ECIPs must be implemented by at least two independent and compatible software applications.

Software authors are encouraged to publish summaries of what ECIPs their software supports to aid in verification of status changes. Good examples of this, at the time of writing this ECIP, can be observed in Bitcoin Core's doc/BIPs.md file as well as Bitcoin Wallet for Android's wallet/README.specs file.

These criteria are considered objective ways to observe the de facto adoption of the ECIP, and are not to be used as reasons to oppose or reject an ECIP. Should an ECIP become actually and unambiguously adopted despite not meeting the criteria outlined here, it should still be updated to `Final` status.

Rationale

Why is this necessary at all?

- Many ECIPs with significant real-world use have been left as `Draft` or `Last Call` status longer than appropriate. By giving objective criteria to judge the progression of ECIPs, this proposal aims to help keep the **Status** accurate and up-to-date.

What is the ideal percentage of listening nodes needed to adopt peer services proposals?

- This is unknown, and set rather arbitrarily at this time. For a random selection of peers to have at least one other peer implementing the extension, 13% or more would be necessary, but nodes could continue to scan the network for such peers with perhaps some reasonable success. Furthermore, service bits exist to help identification upfront.

Why is it necessary for at least two software projects to release an implementation of API/RPC and application layer ECIPs, before they become Final?

- If there is only one implementation of a specification, there is no other program for which a standard interface is used with or needed.
- Even if there are only two projects rather than more, some standard coordination between them exists.

What if a ECIP is proposed that only makes sense for a single specific project?

- The ECIP process exists for standardisation between independent projects. If something only affects one project, it should be done through that project's own internal processes, and never be proposed as a ECIP in the first place.

ECIP comments

Specification

Each ECIP should, in its preamble, link to a public wiki page with a summary tone of the comments on that page.

Reviewers of the ECIP who consider themselves qualified, should post their own comments on this wiki page.

The comments page should generally only be used to post final comments for a completed ECIP.

If an ECIP is not yet completed, reviewers should instead post on the applicable development Discord group or mailing list thread to allow the ECIP author(s) to address any concerns or problems pointed out by the review.

Some ECIPs receive exposure outside the development community prior to completion, and other ECIPs might not be completed at all. To avoid a situation where critical ECIP reviews may go unnoticed during this period, reviewers may, at their option, still post their review on the comments page, provided they first post it to the mailing list and plan to later remove or revise it as applicable based on the completed version. Such revisions should be made by editing the previous review and updating the timestamp. Reviews made prior to the complete version may be removed if they are no longer applicable and have not been updated in a timely manner (eg, within one month).

Pages must be named after the full ECIP number (eg, "ECIP 0001") and placed in the "Comments" namespace.

For example, the link for ECIP 1 will be:

https://github.com/ethereumclassic/ECIPs/wiki/Comments:ECIP-0001 .

Comments posted to this wiki should use the following format:

<Your opinion> --<Your name>, <Date of posting, as YYYY-MM-DD>

ECIPs may also choose to list a second forum for ECIP comments, in addition to the ECIPs wiki. In this case, the second forum's URI should be listed below the primary wiki's URI.

After some time, the ECIP itself may be updated with a summary tone of the comments. Summary tones may be chosen from the following, but this ECIP does not intend to cover all possible nuances and other summaries may be used as needed:

- No comments yet.
- Unanimously Recommended for implementation
- Unanimously Discourage for implementation
- Mostly Recommended for implementation, with some Discouragement
- Mostly Discouraged for implementation, with some Recommendation

For example, the preamble to ECIP 1 might be updated to include the line:

Comments-Summary: No comments yet.
Comments-URI: https://github.com/ethereumclassic/ECIPs/wiki/Comments:ECIP-1000
https://some-other-wiki.org/ECIP_1_Comments

These fields must follow the **Discussions-To** header defined in ECIP 1 (if that header is not present, it should follow the position where it would be present; generally this is immediately above the **Status** header).

To avoid doubt: comments and status are unrelated metrics to judge an ECIP, and neither should be directly influencing the other.

Rationale

What is the purpose of ECIP comments?

- Various ECIPs have been adopted (the criteria required for Final status) despite being considered generally inadvisable. Some presently regard ECIPs as a "good idea" simply by virtue of them being assigned an ECIP number. Due to the low barrier of entry for submission of new ECIPs, it seems advisable for a way for reviewers to express their opinions on them in a way

that is consumable to the public without needing to review the entire development discussion.

Will ECIP comments be censored or limited to particular participants/"experts"?

- Participants should freely refrain from commenting outside of their area of knowledge or expertise. However, comments should not be censored, and participation should be open to the public.

ECIP Licensing

Specification

New ECIPs may be accepted with the following licenses. Each new ECIP must identify at least one acceptable license in its preamble. The License header in the preamble must be placed after the Created header. Each license must be referenced by their respective abbreviation given below.

For example, a preamble might include the following License header:

```
License: BSD-2-Clause
    GNU-All-Permissive
```

In this case, the ECIP text is fully licensed under both the OSI-approved BSD 2-clause license as well as the GNU All-Permissive License, and anyone may modify and redistribute the text provided they comply with the terms of *either* license. In other words, the license list is an "OR choice", not an "AND also" requirement.

It is also possible to license source code differently from the ECIP text. A optional License-Code header is placed after the License header. Again, each license must be referenced by their respective abbreviation given below.

For example, a preamble specifying the optional License-Code header might look like:

```
License: BSD-2-Clause
    GNU-All-Permissive
License-Code: GPL-2.0+
```

In this case, the code in the ECIP is not available under the BSD or All-Permissive licenses, but only under the terms of the GNU General Public License (GPL), version 2 or newer. If the code were to be available under *only* version 2 exactly, the "+" symbol should be removed from the license abbreviation. For a later version (eg, GPL 3.0), you would increase the version number (and retain or remove the "+" depending on intent).

License-Code: GPL-2.0 # This refers to GPL v2.0 *only*, no later license versions are acceptable.
License-Code: GPL-2.0+ # This refers to GPL v2.0 *or later*.
License-Code: GPL-3.0 # This refers to GPL v3.0 *only*, no later license versions are acceptable.
License-Code: GPL-3.0+ # This refers to GPL v3.0 *or later*.

In the event that the licensing for the text or code is too complicated to express with a simple list of alternatives, the list should instead be replaced with the single term "Complex". In all cases, details of the licensing terms must be provided in the Copyright section of the ECIP.

ECIPs are not required to be *exclusively* licensed under approved terms, and may also be licensed under unacceptable licenses *in addition to* at least one acceptable license. In this case, only the acceptable license(s) should be listed in the License and License-Code headers.

Recommended licenses

- Apache-2.0: Apache License, version 2.0
- BSD-2-Clause: OSI-approved BSD 2-clause license
- BSD-3-Clause: OSI-approved BSD 3-clause license
- CC0-1.0: Creative Commons CC0 1.0 Universal
- GNU-All-Permissive: GNU All-Permissive License

In addition, it is recommended that literal code included in the ECIP be dual-licensed under the same license terms as the project it modifies. For example, literal code intended for Ethereum Classic Core would ideally be dual-licensed under the MIT license terms as well as one of the above with the rest of the ECIP text.

Not recommended, but acceptable licenses

- BSL-1.0: Boost Software License, version 1.0

- CC-BY-4.0: Creative Commons Attribution 4.0 International
- CC-BY-SA-4.0: Creative Commons Attribution-ShareAlike 4.0 International
- MIT: Expat/MIT/X11 license
- AGPL-3.0+: GNU Affero General Public License (AGPL), version 3 or newer
- FDL-1.3: GNU Free Documentation License, version 1.3
- GPL-2.0+: GNU General Public License (GPL), version 2 or newer
- LGPL-2.1+: GNU Lesser General Public License (LGPL), version 2.1 or newer

Rationale

Why are there software licenses included?

- Some ECIPs, especially consensus layer, may include literal code in the ECIP itself which may not be available under the exact license terms of the ECIP.
- Despite this, not all software licenses would be acceptable for content included in ECIPs.

See Also

- RFC 7282: On Consensus and Humming in the IETF

Ethereum Classic Improvement Proposals

- Ethereum Classic Improvement Proposals
- ECIPs Github

Ethereum Classic Improvement Proposals (ECIPs) describe standards for the Ethereum Classic platform, including core protocol specifications, client APIs, and contract standards.