Summary

TZIP (pronounce "tee-zip") stands for Tezos Improvement Proposal, which are documents that explain how the Tezos blockchain can be improved with new and updated standards and concepts, such as smart contract specifications.

Abstract

Tezos is a large and decentralized project, which can silo information and inhibit collaboration across different projects, companies and continents. This can lead to duplication of effort and low attention density on specific issues. At the same time, the distributed nature of Tezos gives the project vitality and dynamism.

What is a TZIP?

A TZIP is a design document providing information to the Tezos community, describing a feature for Tezos or its processes or environment, and supporting the formal protocol governance process.

The role of a TZIP is purely supplementary, hortative, or descriptive; the Tezos protocol is defined at all times by the implementation whose hash has been incorporated into the current protocol ID and approved by the Tezos quorum. TZIPs can and will diverge from and contradict one another, so long as all active TZIPs remain compatible with the current protocol.

A TZIP should contain a concise technical specification and rationale which unambiguously articulates what the proposal is, how it may be implemented, and why the proposal is an improvement. A TZIP should additionally contain an FAQ which documents, compares, and answers alternative options, opinions, and objections.

Rationale

We intend TZIPs to be one possible mechanism for proposing new features, for collecting community technical input on an issue, and for documenting the design decisions that have gone into Tezos. TZIPs are maintained as text files in a versioned repository; their revision history is a historical record of the feature proposal.

For Tezos developers, TZIPs can be a useful way to generate feedback, clearly communicate functionality to users, and track the progress of their work. Maintainers of Tezos tools and libraries can list the TZIPs that they have implemented, which can give the community a convenient way to know the current status of a given project.

There are many types of TZIP defined in TZIP-002. Some TZIPs, such as contract specifications, also have a *name* that consists of the TZIP type (e.g. `A`, `FA`, `I`), concatenated with a serial numbers which may be extended with a `.` character (e.g. `FA1.2` is an extension of `FA1`). Again, this is defined in details in TZIP-002.

It is highly recommended that a single TZIP contain a single key proposal or new idea corresponding to its type. A change to a single library or application may not require an TZIP, whereas a change that affects multiple independent libraries or applications probably does.

A TZIP must meet certain minimum criteria, which may change depending on the TZIP type. Some TZIP types may require an attached reference implementation of the proposed improvement. Other TZIP types may require two independent implementations. If so, the associated software should be of minimal possible complexity and be focused primarily on illustrating the contents of the single proposal to which it is attached. Reference implementations should show evidence of correctness, ideally via formal methods, but at minimum via comprehensive testing.

TZIP Workflow

A TZIP will always move along the following workflow:

TZIP workflow

Status Description

A TZIP can have the following statuses:

Status	**Actor**	**Description**	**Action(s) to get that Status**	**Next Status(es)**
`Work In Progress`	Author	This covers the initial work being done to create the TZIP and its accompanying resources	1. Author submits a MR to: - reserve a TZIP number, say *xxx*, in the README - add the *Work In Progress* status in the README - create a *tzip-xxx* folder in proposals 2. Author creates a new branch, *tzip-xx*, to work on it	Draft, Withdrawn
`Draft`	Author	The TZIP is now in good shape, and is ready to be shared with the community to get iterative feedback	1. Author submits a MR from the *tzip-xxx* branch, including the status change to *Draft* in the README 2. Reviewer merges the MR to *master*	Submitted, Withdrawn
`Withdrawn`	-	The authors decide that the TZIP is no longer needed, or the reviewers decide to reject it (e.g. for non-compliance)	1. Author or Reviewer closes the *tzip-xxx* branch 2. Author or Reviewer updates the TZIP and README with the *Withdrawn* status	-

Status	Actor	Description	Action(s) to get that Status	Next Status(es)
`Submitted`	Reviewer	The TZIP has been approved by the community and is submitted to the reviewers	Author updates the TZIP and README with the *Submitted* status	Final, Draft, Withdrawn
`Final`	Reviewer	The TZIP has been approved by the reviewers	Reviewer updates the TZIP and README with the *Final* status	Deprecated, Superseded
`Deprecated`	-	The TZIP is no longer valid and shouldn't be used anymore. It is kept here as a reference but hasn't been superseded by any newer TZIP	Reviewer updates the TZIP and README with the *Deprecated* status	-
`Superseded`	-	The TZIP is no longer valid, but unlike the *Deprecated* status, this TZIP has been superseded by a newer TZIP	Reviewer updates the TZIP and README with the *Superseded* status	-

TZIP Authors

Once the authors are happy with the initial work on their TZIP, they will move its status from *Work In Progress* to *Draft*. At that point, the authors should solicit feedback from the Tezos community before submitting the TZIP to the reviewers. As prior work on the same or similar topics may not be immediately apparent or accessible from publicly available information, inviting community advice can both save time and effort, and result in a stronger proposal.

Examples of appropriate public forums to gauge response to a TZIP:

- The Tezos subreddit
- The Issues page of this repository

Once the authors have gotten enough feedback from the community and updated the TZIP accordingly, they will move its status to *Submitted* for the reviewers to review.

TZIP Reviewers

Once the TZIP moves to *Submitted*, the reviewers will:

- Read the TZIP to check if it complies with all relevant standards articulated in this document and elsewhere, both global TZIP standards and those particular to the TZIP type.
- Edit the TZIP for prose (spelling, grammar, sentence structure, etc.), markup (GitLab Flavored Markdown).
- Review any accompanying resources, such as implementations, proofs, papers, links, etc.
- When relevant, determine whether the FAQ fairly expresses and responds to criticisms of the proposal.

If the TZIP does not meet the standards, the reviewers will send it back to *Draft*, with specific instructions, so the authors can update it. This process may repeat as many times as necessary. The reviewers may additionally refer the TZIP to outside reviewers with relevant knowledge. Reviewers may also outright reject a TZIP submission that is unsuitable for review or is otherwise non-compliant with the standards outlined in relevant active TZIPs.

Should the reviewers approve the TZIP, they will:

- Merge the corresponding merge request. The TZIP reviewers have sole ownership over the GitLab repository.
- Coordinate any further steps with the TZIP authors, as required by the specific TZIP type.

TZIP Sections

Each TZIP should have the following sections:

- **Header Preamble**: Headers containing metadata about the TZIP, including the TZIP type, a short title, and the author details.
- **Summary**: Provide a simplified and layman-accessible explanation of the TZIP.
- **Abstract**: A short (200-500 word) but comprehensive description of the issue being addressed and the proposed solution.
- **Motivation**: It should clearly explain why the existing implementation is inadequate to address the problem that the TZIP solves.
- **Specification**: The technical specification should describe the syntax and semantics of any new feature.
- **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, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.
- **Backwards Compatibility**: All TZIPs that introduce backwards incompatibilities or supersede other TZIPs must include a section describing these incompatibilities, their severity, and solutions.
- **Test Cases**: Test cases for an implementation are strongly recommended as are any proofs of correctness via formal methods.
- **Implementations**: Any implementation must be completed before the TZIP is given the status Submitted, but is not mandatory in *Work In Progress* and *Draft* status.
- **Appendix**: A list of references relevant to the proposal.
- **Copyright**: All TZIPs must be in the public domain, or under a permissive license substantially identical to placement in the public domain. See the bottom of this TZIP for an example copyright waiver.

TZIP Formats and Templates

TZIPs should be written in Markdown format. TZIP templates are available in the Templates folder. Additionally, the authors agree to follow and enforce the TZIP Code of Conduct, described in TZIP-003.

To make the text easier to review, all TZIPs should be hard-wrapped at 80 characters.

TZIP Header Preamble

Each TZIP must begin with an RFC 822 style header preamble, preceded and followed by three hyphens (`---`). The headers must appear in the following order.

- `tzip:` TZIP number. The number should be requested via a merge request as soon as the TZIP is in *Work In Progress* status
- `title:` A short descriptive title, maximum 44 characters. If the TZIP has a *name* (see TZIP-002), then it is added as a title prefix (e.g. *FA1 - Abstract Ledger*)
- `author:` Name(s) of author(s), ideally with their username(s) or email address(es). Examples: *John Doe*, *John Doe (@username)*, *John Doe <address@dom.ain>*
- `gratuity:` Optional. A Tezos address controlled by an author capable of receiving gratuities from grateful Tezos users
- `discussions-to:` Optional. A url pointing to the official discussion thread
- `status:` Can be: Work In Progress | Draft | Withdrawn | Submitted | Deprecated | Superseded
- `type:` TZIP type as defined in TZIP-002
- `created:` Date the TZIP was created, format yyyy-mm-dd
- `requires:` Optional. TZIP numbers, representing TZIPs that this TZIP depends on
- `replaces:` Optional. TZIP numbers, representing the TZIPs the current TZIP is replacing
- `superseded-by:` Optional. TZIP numbers, representing the TZIPS replacing the current TZIP

Headers that permit lists must separate elements with commas.

Transferring Primary Authorship

It occasionally becomes necessary to transfer primary authorship of a TZIP to a new primary author. In such case, the previous primary author will remain as a co-author or an author emeritus. This can occur by unilateral decision of the primary author, via a request of a co-author, or in extraordinary circumstances via a unilateral decision from the TZIP reviewers.

History

This document was derived from Ethereum's EIP-1 which was in turn derived from Bitcoin's BIP-0001 written by Amir Taaki and Python's PEP-0001, written by Barry Warsaw, Jeremy Hylton, and David Goodger.

None of the authors of EIP-1, BIP-0001, or PEP-0001 are responsible for its use in the Tezos Improvement Process, and should not be bothered with technical questions specific to TZIP.

Copyright