Nostr protocol in a single page

Last Updated: 2026-01-21 nostr

PAGE CONTENT

- NIPs
  - List
  - Event Kinds
  - Message types
  - Common Tags
  - Criteria for acceptance of NIPs
  - Is this repository a centralizing factor?
  - How this repository works
  - License
  - Contributors
- Basic protocol flow description
  - Events and signatures
  - Communication between clients and relays
- Follow List
  - Uses
- OpenTimestamps Attestations for Events
- Encrypted Direct Message
  - Security Warning
  - Client Implementation Warning

- Mapping Nostr keys to DNS-based internet identifiers
  - Finding users from their NIP-05 identifier
  - Notes
- Basic key derivation from mnemonic seed phrase
- `window.nostr` capability for web browsers
- Handling Mentions
- Event Deletion Request
  - Client Usage
  - Relay Usage
  - Deletion Request of a Deletion Request
- Text Notes and Threads
  - Abstract
  - Marked “e” tags (PREFERRED)
  - The “p” tag
  - Deprecated Positional “e” tags
- Relay Information Document
- Field Descriptions
- Extra Fields
- Generic Tag Queries
- Proof of Work
- Mining
- Example mined note
- Validating
- Delegated Proof of Work

- Subject tag in Text events
- Nostr Marketplace
  - Terms
  - Nostr Marketplace Clients
  - `Merchant` publishing/updating products (event)
  - Checkout events
  - Customize Marketplace
  - Auctions
  - Customer support events
  - Additional
- Event Treatment
- Private Direct Messages
  - Kind Definitions
  - File Message
  - Chat Rooms
  - Encrypting
  - Publishing
  - Relays
  - Benefits & Limitations
  - Examples
- Reposts
  - Quote Reposts
  - Generic Reposts
- bech32-encoded entities

- Bare keys and ids
- Shareable identifiers with extra metadata
- Examples
- Notes
- Command Results
- `nostr:` URI scheme
- Comment
- Examples
- Long-form Content
- Example Event
- Extra metadata fields and tags
- Reactions
- Tags
- External Content Reactions
- Custom Emoji Reaction
- Delegated Event Signing
- Text Note References
- Example of a profile mention process
- Verbose and probably unnecessary considerations
- Public Chat
- Kind 40: Create channel
- Kind 41: Set channel metadata
- Kind 42: Create channel message
- Kind 43: Hide message

- Kind 44: Mute user
- Relay recommendations
- Motivation
- Additional info
- Relay-based Groups
- Relay-generated events
- Group identifier
- The `h` tag
- Timeline references
- Late publication
- Group management
- Unmanaged groups
- Event definitions
- Implementation quirks
- Custom Emoji
- Dealing with unknown event kinds
- Labeling
- Label Namespace Tag
- Label Tag
- Label Target
- Content
- Self-Reporting
- Example events
- Other Notes

- Appendix: Known Ontologies
- Parameterized Replaceable Events
- `git` stuff
  - Repository announcements
  - Repository state announcements
  - Patches and Pull Requests (PRs)
  - Issues
  - Replies
  - Status
  - User grasp list
  - Possible things to be added later
- Torrents
  - Tags
  - Tag prefixes
  - Torrent Comments
  - Implementations
- Sensitive Content / Content Warning
- Draft Wraps
  - Relay List for Private Content
- User Statuses
  - Abstract
  - Live Statuses
  - Client behavior
  - Use Cases

- External Identities in Profiles
  - Abstract
  - `i` tag on a metadata event
  - Claim types
- Expiration Timestamp
- Client Behavior
- Relay Behavior
- Suggested Use Cases
- Authentication of clients to relays
  - Motivation
  - Definitions
  - Protocol flow
  - Signed Event Verification
- Relay Access Metadata and Requests
  - Membership Lists
  - Add User
  - Remove User
  - Join Request
  - Invite Request
  - Leave Request
  - Implementation
- Encrypted Payloads (Versioned)
  - Versions
  - Limitations

- Version 2
- Event Counts
- Motivation
- Filters and return values
- Examples
- Nostr Remote Signing
- Changes
- Rationale
- Terminology
- Overview
- Initiating a connection
- Request Events `kind: 24133`
- Response Events `kind:24133`
- Example flow for signing an event
- Auth Challenges
- Appendix
- Nostr Wallet Connect (NWC)
- Rationale
- Terms
- Theory of Operation
- Events
- Nostr Wallet Connect URI
- Commands
- Notifications

- Example pay invoice flow
- Encryption
- Using a dedicated relay
- Metadata
- Appendix
- Proxy Tags
- Private Key Encryption
- Symmetric Encryption Key derivation
- Encrypting a private key
- Test Data
- Password Unicode Normalization
- Encryption
- Decryption
- Discussion
- Recommendations
- Search Capability
- Abstract
- `search` filter field
- Extensions
- Lists
- Types of lists
- Examples
- Encryption process pseudocode
- Calendar Events

- Calendar Events
- Collaborative Calendar Event Requests
- Calendar
- Calendar Event RSVP
- Unsolved Limitations
- Intentionally Unsupported Scenarios

Live Activities

- Live Streaming
- Meeting Spaces

Wiki

- Articles
- `d` tag normalization rules
- Content
- Optional extra tags
- Merge Requests
- Redirects
- How to decide what article to display
- Forks
- Deference
- Why Asciidoc?
- Appendix 1: Merge requests

Android Signer Application

- Usage for Android applications
- Usage for Web Applications

- Reporting
- Tags
- Example events
- Client behavior
- Relay behavior
- Lightning Zaps
  - Protocol flow
  - Reference and examples
  - Future Work
- Badges
- Gift Wrap
  - Overview
  - Protocol Description
  - Encrypting Payloads
  - Other Considerations
  - An Example
  - Code Samples
- Cashu Wallets
  - High-level flow
  - Flow
  - Appendix 1: Validating proofs
- Nutzaps
  - High-level flow
  - Nutzap informational event

- Sending a nutzap
- Receiving nutzaps
- Request to Vanish
- Request to Vanish from Relay
- Global Request to Vanish
- Chess (Portable Game Notation)
- Note
- Client Behavior
- Relay Behavior
- Examples
- Resources
- Relay List Metadata
- Relay Discovery and Liveness Monitoring
- Relay Discovery Events
- Relay Monitor Announcements
- Picture-first feeds
- Picture Events
- Peer-to-peer Order events
- Abstract
- The event
- Tags
- Implementations
- References
- Protected Events

- The tag
- Example flow
- Why
- Video Events
- Video Events
- Addressable Video Events
- Addressable Event Example
- Referencing Addressable Events
- Moderated Communities (Reddit Style)
- Community Definition
- Posting to a community
- Moderation
- Cross-posting
- External Content IDs
- Supported IDs
- Examples
- Zap Goals
- Nostr Event
- Client behavior
- Use cases
- Negentropy Syncing
- High-Level Protocol Description
- Nostr Messages
- Appendix: Negentropy Protocol V1

- Arbitrary custom app data
  - Nostr event
  - Some use cases
- Highlights
  - Format
  - Quote Highlights
- Relay Management API
- Ecash Mint Discoverability
  - Rationale
  - Events
  - Ecash Mint Information
  - Example
- Polls
  - Events
- Recommended Application Handlers
  - Rationale
  - Events
  - Client tag
  - User flow
  - Example
- Data Vending Machine
  - Kinds
  - Rationale
  - Job request (`kind:5000-5999`)

- Job result (`kind:6000-6999`)
- Job feedback
- Protocol Flow
- Cancellation
- Appendix 1: Job chaining
- Appendix 2: Service provider discoverability

Media Attachments

- Example
- Recommended client behavior

File Metadata

- Event format
- Suggested use cases

HTTP File Storage Integration

- Introduction
- Server Adaptation
- Auth
- Upload
- Download
- Deletion
- Listing files
- Selecting a Server

HTTP Auth

- Nostr event
- Request Flow

Reference Implementations
Classified Listings
Example Event
References

NIPs

NIPs stand for **Nostr Implementation Possibilities**.

They exist to document what may be implemented by **Nostr** -compatible *relay* and *client* software.

- List
- Event Kinds
- Message Types
  - Client to Relay
  - Relay to Client
- Common Tags
- Criteria for acceptance of NIPs
- Is this repository a centralizing factor?
- How this repository works
- License

List

- NIP-01: Basic protocol flow description
- NIP-02: Follow List
- NIP-03: OpenTimestamps Attestations for Events

- **NIP-04: Encrypted Direct Message** — **unrecommended**: deprecated in favor of **NIP-17**
- **NIP-05: Mapping Nostr keys to DNS-based internet identifiers**
- **NIP-06: Basic key derivation from mnemonic seed phrase**
- **NIP-07: `window.nostr` capability for web browsers**
- **NIP-08: Handling Mentions** — **unrecommended**: deprecated in favor of **NIP-27**
- **NIP-09: Event Deletion Request**
- **NIP-10: Text Notes and Threads**
- **NIP-11: Relay Information Document**
- **NIP-13: Proof of Work**
- **NIP-14: Subject tag in text events**
- **NIP-15: Nostr Marketplace (for resilient marketplaces)**
- **NIP-17: Private Direct Messages**
- **NIP-18: Reposts**
- **NIP-19: bech32-encoded entities**
- **NIP-21: `nostr:` URI scheme**
- **NIP-22: Comment**
- **NIP-23: Long-form Content**
- **NIP-24: Extra metadata fields and tags**
- **NIP-25: Reactions**
- **NIP-26: Delegated Event Signing** — **unrecommended**: adds unnecessary burden for little gain
- **NIP-27: Text Note References**
- **NIP-28: Public Chat**
- **NIP-29: Relay-based Groups**
- **NIP-30: Custom Emoji**
- **NIP-31: Dealing with Unknown Events**
- **NIP-32: Labeling**
- **NIP-34: `git` stuff**

- **NIP-35: Torrents**
- **NIP-36: Sensitive Content**
- **NIP-37: Draft Events**
- **NIP-38: User Statuses**
- **NIP-39: External Identities in Profiles**
- **NIP-40: Expiration Timestamp**
- **NIP-42: Authentication of clients to relays**
- **NIP-43: Relay Access Metadata and Requests**
- **NIP-44: Encrypted Payloads (Versioned)**
- **NIP-45: Counting results**
- **NIP-46: Nostr Remote Signing**
- **NIP-47: Nostr Wallet Connect**
- **NIP-48: Proxy Tags**
- **NIP-49: Private Key Encryption**
- **NIP-50: Search Capability**
- **NIP-51: Lists**
- **NIP-52: Calendar Events**
- **NIP-53: Live Activities**
- **NIP-54: Wiki**
- **NIP-55: Android Signer Application**
- **NIP-56: Reporting**
- **NIP-57: Lightning Zaps**
- **NIP-58: Badges**
- **NIP-59: Gift Wrap**
- **NIP-60: Cashu Wallet**
- **NIP-61: Nutzaps**
- **NIP-62: Request to Vanish**
- **NIP-64: Chess (PGN)**
- **NIP-65: Relay List Metadata**

- **NIP-66: Relay Discovery and Liveness Monitoring**
- **NIP-68: Picture-first feeds**
- **NIP-69: Peer-to-peer Order events**
- **NIP-70: Protected Events**
- **NIP-71: Video Events**
- **NIP-72: Moderated Communities**
- **NIP-73: External Content IDs**
- **NIP-75: Zap Goals**
- **NIP-77: Negentropy Syncing**
- **NIP-78: Application-specific data**
- **NIP-7D: Threads**
- **NIP-84: Highlights**
- **NIP-86: Relay Management API**
- **NIP-87: Ecash Mint Discoverability**
- **NIP-88: Polls**
- **NIP-89: Recommended Application Handlers**
- **NIP-90: Data Vending Machines**
- **NIP-92: Media Attachments**
- **NIP-94: File Metadata**
- **NIP-96: HTTP File Storage Integration** — **unrecommended**: replaced by blossom APIs
- **NIP-98: HTTP Auth**
- **NIP-99: Classified Listings**
- **NIP-A0: Voice Messages**
- **NIP-A4: Public Messages**
- **NIP-B0: Web Bookmarks**
- **NIP-B7: Blossom**
- **NIP-BE: Nostr BLE Communications Protocol**
- **NIP-C0: Code Snippets**

- **NIP-C7: Chats**
- **NIP-EE: E2EE Messaging using MLS Protocol** — **unrecommended**: superseded by the **Marmot Protocol**

Event Kinds

kind	description	NIP
`0`	User Metadata	**01**
`1`	Short Text Note	**10**
`2`	Recommend Relay	01 (deprecated)
`3`	Follows	**02**
`4`	Encrypted Direct Messages	**04**
`5`	Event Deletion Request	**09**
`6`	Repost	**18**
`7`	Reaction	**25**
`8`	Badge Award	**58**
`9`	Chat Message	**C7**
`10`	Group Chat Threaded Reply	29 (deprecated)
`11`	Thread	**7D**
`12`	Group Thread Reply	29 (deprecated)
`13`	Seal	**59**
`14`	Direct Message	**17**
`15`	File Message	**17**
`16`	Generic Repost	**18**
`17`	Reaction to a website	**25**
`20`	Picture	**68**

`21`	Video Event	**71**
`22`	Short-form Portrait Video Event	**71**
`24`	Public Message	**A4**
`30`	internal reference	**NKBIP-03**
`31`	external web reference	**NKBIP-03**
`32`	hardcopy reference	**NKBIP-03**
`33`	prompt reference	**NKBIP-03**
`40`	Channel Creation	**28**
`41`	Channel Metadata	**28**
`42`	Channel Message	**28**
`43`	Channel Hide Message	**28**
`44`	Channel Mute User	**28**
`62`	Request to Vanish	**62**
`64`	Chess (PGN)	**64**
`443`	KeyPackage	**Marmot**
`444`	Welcome Message	**Marmot**
`445`	Group Event	**Marmot**
`818`	Merge Requests	**54**
`1018`	Poll Response	**88**
`1021`	Bid	**15**
`1022`	Bid confirmation	**15**
`1040`	OpenTimestamps	**03**
`1059`	Gift Wrap	**59**
`1063`	File Metadata	**94**

`1111`	Comment	**22**
`1222`	Voice Message	**A0**
`1244`	Voice Message Comment	**A0**
`1311`	Live Chat Message	**53**
`1337`	Code Snippet	**C0**
`1617`	Patches	**34**
`1618`	Pull Requests	**34**
`1619`	Pull Request Updates	**34**
`1621`	Issues	**34**
`1622`	Git Replies (deprecated)	**34**
`1630` - `1633`	Status	**34**
`1971`	Problem Tracker	**nostrocket**
`1984`	Reporting	**56**
`1985`	Label	**32**
`1986`	Relay reviews	
`1987`	AI Embeddings / Vector lists	**NKBIP-02**
`2003`	Torrent	**35**
`2004`	Torrent Comment	**35**
`2022`	Coinjoin Pool	**joinstr**
`4550`	Community Post Approval	**72**
`5000` - `5999`	Job Request	**90**
`6000` - `6999`	Job Result	**90**
`7000`	Job Feedback	**90**
`7374`	Reserved Cashu Wallet Tokens	**60**

`7376`	Cashu Wallet History	**60**
`7516`	Geocache log	**geocaching**
`7517`	Geocache proof of find	**geocaching**
`8000`	Add User	**43**
`8001`	Remove User	**43**
`9000` - `9030`	Group Control Events	**29**
`9041`	Zap Goal	**75**
`9321`	Nutzap	**61**
`9467`	Tidal login	**Tidal-nostr**
`9734`	Zap Request	**57**
`9735`	Zap	**57**
`9802`	Highlights	**84**
`10000`	Mute list	**51**
`10001`	Pin list	**51**
`10002`	Relay List Metadata	**65** , **51**
`10003`	Bookmark list	**51**
`10004`	Communities list	**51**
`10005`	Public chats list	**51**
`10006`	Blocked relays list	**51**
`10007`	Search relays list	**51**
`10009`	User groups	**51** , **29**
`10012`	Favorite relays list	**51**
`10013`	Private event relay list	**37**
`10015`	Interests list	**51**

10020	Media follows	**51**
10030	User emoji list	**51**
10050	Relay list to receive DMs	**51** , **17**
10051	KeyPackage Relays List	**Marmot**
10063	User server list	**Blossom**
10096	File storage server list	**96** (deprecated)
10166	Relay Monitor Announcement	**66**
10312	Room Presence	**53**
10377	Proxy Announcement	**Nostr Epoxy**
11111	Transport Method Announcement	**Nostr Epoxy**
13194	Wallet Info	**47**
13534	Membership Lists	**43**
14388	User Sound Effect Lists	**Corny Chat**
17375	Cashu Wallet Event	**60**
21000	Lightning Pub RPC	**Lightning.Pub**
22242	Client Authentication	**42**
23194	Wallet Request	**47**
23195	Wallet Response	**47**
24133	Nostr Connect	**46**
24242	Blobs stored on mediaservers	**Blossom**
27235	HTTP Auth	**98**
28934	Join Request	**43**
28935	Invite Request	**43**
28936	Leave Request	**43**

`30001`	Generic lists	51 (deprecated)
`30002`	Relay sets	**51**
`30003`	Bookmark sets	**51**
`30004`	Curation sets	**51**
`30005`	Video sets	**51**
`30006`	Picture sets	**51**
`30007`	Kind mute sets	**51**
`30008`	Profile Badges	**58**
`30009`	Badge Definition	**58**
`30015`	Interest sets	**51**
`30017`	Create or update a stall	**15**
`30018`	Create or update a product	**15**
`30019`	Marketplace UI/UX	**15**
`30020`	Product sold as an auction	**15**
`30023`	Long-form Content	**23**
`30024`	Draft Long-form Content	**23**
`30030`	Emoji sets	**51**
`30040`	Curated Publication Index	**NKBIP-01**
`30041`	Curated Publication Content	**NKBIP-01**
`30063`	Release artifact sets	**51**
`30078`	Application-specific Data	**78**
`30166`	Relay Discovery	**66**
`30267`	App curation sets	**51**
`30311`	Live Event	**53**

`30313`	Conference Event	**53**
`30315`	User Statuses	**38**
`30388`	Slide Set	**Corny Chat**
`30402`	Classified Listing	**99**
`30403`	Draft Classified Listing	**99**
`30617`	Repository announcements	**34**
`30618`	Repository state announcements	**34**
`30818`	Wiki article	**54**
`30819`	Redirects	**54**
`31234`	Draft Event	**37**
`31388`	Link Set	**Corny Chat**
`31890`	Feed	**NUD: Custom Feeds**
`31922`	Date-Based Calendar Event	**52**
`31923`	Time-Based Calendar Event	**52**
`31924`	Calendar	**52**
`31925`	Calendar Event RSVP	**52**
`31989`	Handler recommendation	**89**
`31990`	Handler information	**89**
`32267`	Software Application	
`32388`	User Room Favorites	**Corny Chat**
`33388`	High Scores	**Corny Chat**
`34235`	Addressable Video Event	**71**
`34236`	Addressable Short Video Event	**71**
`34388`	Sound Effects	**Corny Chat**

`38172`	Cashu Mint Announcement	**87**
`38173`	Fedimint Announcement	**87**
`37516`	Geocache listing	**geocaching**
`38383`	Peer-to-peer Order events	**69**
`39000–9`	Group metadata events	**29**
`39089`	Starter packs	**51**
`39092`	Media starter packs	**51**
`39701`	Web bookmarks	**B0**

Message types

Client to Relay

type	description	NIP
`EVENT`	used to publish events	**01**
`REQ`	used to request events and subscribe to new updates	**01**
`CLOSE`	used to stop previous subscriptions	**01**
`AUTH`	used to send authentication events	**42**
`COUNT`	used to request event counts	**45**

Relay to Client

type	description	NIP
`EOSE`	used to notify clients all stored events have been sent	**01**
`EVENT`	used to send events requested to clients	**01**
`NOTICE`	used to send human-readable messages to clients	**01**

`OK`	used to notify clients if an EVENT was successful	01
`CLOSED`	used to notify clients that a REQ was ended and why	01
`AUTH`	used to send authentication challenges	42
`COUNT`	used to send requested event counts to clients	45

Common Tags

name	value	other parameters	NIP
`a`	coordinates to an event	relay URL	01
`A`	root address	relay URL	22
`c`	commit id		34
`d`	identifier	–	01
`e`	event id (hex)	relay URL, marker, pubkey (hex)	01 , 10
`E`	root event id	relay URL	22
`f`	currency code	–	69
`g`	geohash	–	52
`h`	group id	–	29
`i`	external identity	proof, url hint	35 , 39 , 73
`I`	root external identity	–	22
`k`	kind	–	18 , 25 , 72 , 73
`K`	root scope	–	22
`l`	label, label namespace, language name	–	32 , C0

`L`	label namespace	–	**32**
`m`	MIME type	–	**94**
`p`	pubkey (hex)	relay URL, petname	**01** , **02** , **22**
`P`	pubkey (hex)	–	**22** , **57**
`q`	event id (hex)	relay URL, pubkey (hex)	**18**
`r`	a reference (URL, etc)	–	**24** , **25**
`r`	relay url	marker	**65**
`s`	status	–	**69**
`t`	hashtag	–	**24** , **34** , **35**
`u`	url	–	**61** , **98**
`x`	hash	–	**35** , **56**
`y`	platform	–	**69**
`z`	order number	–	**69**
`-`	-	–	**70**
`alt`	summary	–	**31**
`amount`	millisatoshis, stringified	–	**57**
`bolt11`	`bolt11` invoice	–	**57**
`branch-name`	branch name suggestion	–	**34**
`challenge`	challenge string	–	**42**
`client`	name, address	relay URL	**89**
`clone`	git clone URL	–	**34**

`delegation`	pubkey, conditions, delegation token	–	**26**
`dep`	Required dependency	–	**C0**
`description`	description	–	**34** , **57** , **58** , **C0**
`emoji`	shortcode, image URL	–	**30**
`encrypted`	–	–	**90**
`extension`	File extension	–	**C0**
`expiration`	unix timestamp (string)	–	**40**
`file`	full path (string)	–	**35**
`goal`	event id (hex)	relay URL	**75**
`merge-base`	commit id		**34**
`HEAD`	`ref: refs/heads/<branch-name>`		**34**
`image`	image URL	dimensions in pixels	**23** , **52** , **58**
`imeta`	inline metadata	–	**92**
`license`	License of the shared content	–	**C0**
`lnurl`	`bech32` encoded `lnurl`	–	**57**
`location`	location string	–	**52** , **99**
`name`	name	–	**34** , **58** , **72** , **C0**
`nonce`	random	difficulty	**13**
`preimage`	hash of `bolt11` invoice	–	**57**
`price`	price	currency, frequency	**99**

`proxy`	external ID	protocol	**48**
`published_at`	unix timestamp (string)	–	**23** , **B0**
`relay`	relay url	–	**42** , **17**
`relays`	relay list	–	**57**
`repo`	Reference to the origin repository	–	**C0**
`runtime`	Runtime or environment specification	–	**C0**
`server`	file storage server url	–	**96**
`sound`	shortcode, sound url, image url	–	**51**
`subject`	subject	–	**14** , **17** , **34**
`summary`	summary	–	**23** , **52**
`thumb`	badge thumbnail	dimensions in pixels	**58**
`title`	title	–	**23** , **B0**
`tracker`	torrent tracker URL	–	**35**
`web`	webpage URL	–	**34**
`zap`	pubkey (hex), relay URL	weight	**57**

Please update these lists when proposing new NIPs.

Criteria for acceptance of NIPs

1. They should be fully implemented in at least two clients and one relay – when applicable.
2. They should make sense.

3. They should be optional and backwards-compatible: care must be taken such that clients and relays that choose to not implement them do not stop working when interacting with the ones that choose to.
4. There should be no more than one way of doing the same thing.
5. Other rules will be made up when necessary.

Is this repository a centralizing factor?

To promote interoperability, we need standards that everybody can follow, and we need them to define a **single way of doing each thing** without ever hurting **backwards-compatibility**, and for that purpose there is no way around getting everybody to agree on the same thing and keep a centralized index of these standards. However the fact that such an index exists doesn't hurt the decentralization of Nostr. *At any point the central index can be challenged if it is failing to fulfill the needs of the protocol* and it can migrate to other places and be maintained by other people.

It can even fork into multiple versions, and then some clients would go one way, others would go another way, and some clients would adhere to both competing standards. This would hurt the simplicity, openness and interoperability of Nostr a little, but everything would still work in the short term.

There is a list of notable Nostr software developers who have commit access to this repository, but that exists mostly for practical reasons, as by the nature of the thing we're dealing with the repository owner can revoke membership and rewrite history as they want – and if these actions are unjustified or perceived as bad or evil the community must react.

How this repository works

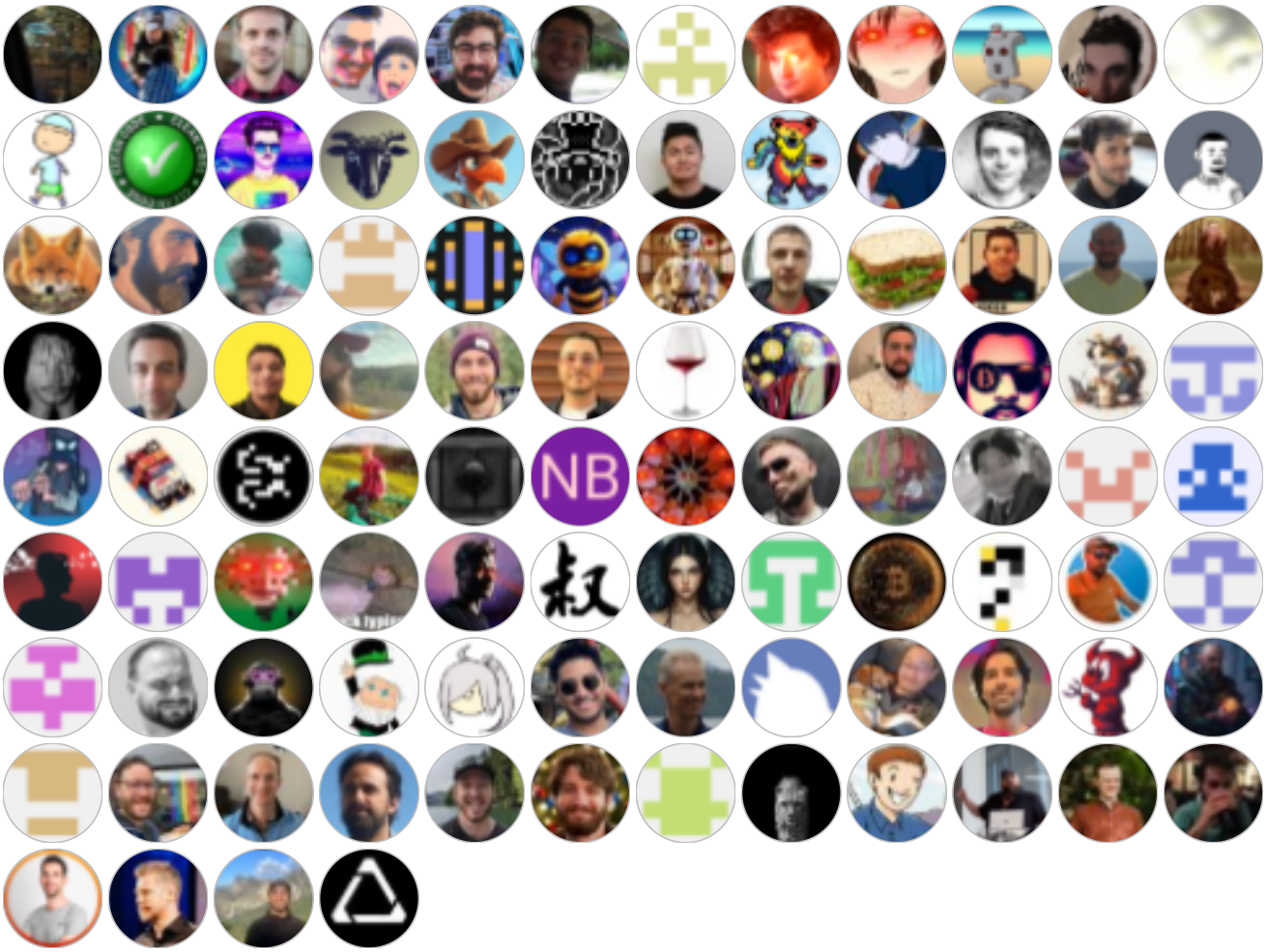
Standards may emerge in two ways: the first way is that someone starts doing something, then others copy it; the second way is that someone has an idea of a new standard that could benefit multiple clients and the protocol in general without breaking **backwards-compatibility** and the principle of having **a single**

way of doing things, then they write that idea and submit it to this repository, other interested parties read it and give their feedback, then once most people reasonably agree we codify that in a NIP which client and relay developers that are interested in the feature can proceed to implement.

These two ways of standardizing things are supported by this repository. Although the second is preferred, an effort will be made to codify standards emerged outside this repository into NIPs that can be later referenced and easily understood and implemented by others – but obviously as in any human system discretion may be applied when standards are considered harmful.

License

Contributors



NIP-01

Basic protocol flow description

`draft` `mandatory` `relay`

This NIP defines the basic protocol that should be implemented by everybody. New NIPs may add new optional (or mandatory) fields and messages and features to the structures and flows described here.

Events and signatures

Each user has a keypair. Signatures, public key, and encodings are done according to the **Schnorr signatures standard for the curve** `secp256k1` .

The only object type that exists is the `event`, which has the following format on the wire:

```
{
  "id": <32-bytes lowercase hex-encoded sha256 of the serialized event data>,
  "pubkey": <32-bytes lowercase hex-encoded public key of the event creator>,
  "created_at": <unix timestamp in seconds>,
  "kind": <integer between 0 and 65535>,
  "tags": [
    [<arbitrary string>...],
    // ...
  ],
  "content": <arbitrary string>,
  "sig": <64-bytes lowercase hex of the signature of the sha256 hash of the s
}
```

To obtain the `event.id`, we `sha256` the serialized event. The serialization is done over the UTF-8 JSON-serialized string (which is described below) of the following structure:

```
[
  0,
  <pubkey, as a lowercase hex string>,
  <created_at, as a number>,
  <kind, as a number>,
  <tags, as an array of arrays of non-null strings>,
  <content, as a string>
]
```

To prevent implementation differences from creating a different event ID for the same event, the following rules MUST be followed while serializing:

- UTF-8 should be used for encoding.
- Whitespace, line breaks or other unnecessary formatting should not be included in the output JSON.

- The following characters in the content field must be escaped as shown, and all other characters must be included verbatim:
  - A line break (`0x0A`), use `\n`
  - A double quote (`0x22`), use `\"`
  - A backslash (`0x5C`), use `\\`
  - A carriage return (`0x0D`), use `\r`
  - A tab character (`0x09`), use `\t`
  - A backspace, (`0x08`), use `\b`
  - A form feed, (`0x0C`), use `\f`

Tags

Each tag is an array of one or more strings, with some conventions around them. Take a look at the example below:

```
{
  "tags": [
    ["e", "5c83da77af1dec6d7289834998ad7aafbd9e2191396d75ec3cc27f5a77226f36",
    ["p", "f7234bd4c1394dda46d09f35bd384dd30cc552ad5541990f98844fb06676e9ca"]
    ["a", "30023:f7234bd4c1394dda46d09f35bd384dd30cc552ad5541990f98844fb06676
    ["alt", "reply"],
    // ...
  ],
  // ...
}
```

The first element of the tag array is referred to as the tag *name* or *key* and the second as the tag *value*. So we can safely say that the event above has an `e` tag set to `"5c83da77af1dec6d7289834998ad7aafbd9e2191396d75ec3cc27f5a77226f36"`, an `alt` tag set to `"reply"` and so on. All elements after the second do not have a conventional name.

This NIP defines 3 standard tags that can be used across all event kinds with the same meaning. They are as follows:

- The `e` tag, used to refer to an event: `["e", <32-bytes lowercase hex of the id of another event>, <recommended relay URL, optional>, <32-bytes lowercase hex of the author's pubkey, optional>]`
- The `p` tag, used to refer to another user: `["p", <32-bytes lowercase hex of a pubkey>, <recommended relay URL, optional>]`
- The `a` tag, used to refer to an addressable or replaceable event
  - for an addressable event: `["a", "<kind integer>:<32-bytes lowercase hex of a pubkey>:<d tag value>", <recommended relay URL, optional>]`
  - for a normal replaceable event: `["a", "<kind integer>:<32-bytes lowercase hex of a pubkey>:", <recommended relay URL, optional>]` (note: include the trailing colon)

As a convention, all single-letter (only english alphabet letters: a-z, A-Z) key tags are expected to be indexed by relays, such that it is possible, for example, to query or subscribe to events that reference the event `"5c83da77af1dec6d7289834998ad7aafbd9e2191396d75ec3cc27f5a77226f36"` by using the `{"#e": ["5c83da77af1dec6d7289834998ad7aafbd9e2191396d75ec3cc27f5a77226f36"]}` filter. Only the first value in any given tag is indexed.

Kinds

Kinds specify how clients should interpret the meaning of each event and the other fields of each event (e.g. an `"r"` tag may have a meaning in an event of kind 1 and an entirely different meaning in an event of kind 10002). Each NIP may define the meaning of a set of kinds that weren't defined elsewhere. **NIP-10** , for instance, specifies the `kind:1` text note for social media applications.

This NIP defines one basic kind:

- `0` : **user metadata**: the `content` is set to a stringified JSON object `{name: <nickname or full name>, about: <short bio>, picture: <url of the image>}` describing the user who created the event. **Extra metadata fields**

may be set. A relay may delete older events once it gets a new one for the same pubkey.

And also a convention for kind ranges that allow for easier experimentation and flexibility of relay implementation:

- for kind `n` such that `1000 <= n < 10000 || 4 <= n < 45 || n == 1 || n == 2`, events are **regular**, which means they're all expected to be stored by relays.
- for kind `n` such that `10000 <= n < 20000 || n == 0 || n == 3`, events are **replaceable**, which means that, for each combination of `pubkey` and `kind`, only the latest event MUST be stored by relays, older versions MAY be discarded.
- for kind `n` such that `20000 <= n < 30000`, events are **ephemeral**, which means they are not expected to be stored by relays.
- for kind `n` such that `30000 <= n < 40000`, events are **addressable** by their `kind`, `pubkey` and `d` tag value – which means that, for each combination of `kind`, `pubkey` and the `d` tag value, only the latest event MUST be stored by relays, older versions MAY be discarded.

In case of replaceable events with the same timestamp, the event with the lowest id (first in lexical order) should be retained, and the other discarded.

When answering to `REQ` messages for replaceable events such as `{"kinds":[0],"authors":[<hex-key>]}`, even if the relay has more than one version stored, it SHOULD return just the latest one.

These are just conventions and relay implementations may differ.

Communication between clients and relays

Relays expose a websocket endpoint to which clients can connect. Clients SHOULD open a single websocket connection to each relay and use it for all their subscriptions. Relays MAY limit number of connections from specific IP/client/etc.

From client to relay: sending events and creating subscriptions

Clients can send 3 types of messages, which must be JSON arrays, according to the following patterns:

- `["EVENT", <event JSON as defined above>]`, used to publish events.
- `["REQ", <subscription_id>, <filters1>, <filters2>, ...]`, used to request events and subscribe to new updates.
- `["CLOSE", <subscription_id>]`, used to stop previous subscriptions.

`<subscription_id>` is an arbitrary, non-empty string of max length 64 chars. It represents a subscription per connection. Relays MUST manage `<subscription_id>`s independently for each WebSocket connection. `<subscription_id>`s are not guaranteed to be globally unique.

`<filtersX>` is a JSON object that determines what events will be sent in that subscription, it can have the following attributes:

```
{
  "ids": <a list of event ids>,
  "authors": <a list of lowercase pubkeys, the pubkey of an event must be one
  "kinds": <a list of a kind numbers>,
  "#<single-letter (a-zA-Z)>": <a list of tag values, for #e — a list of even
  "since": <an integer unix timestamp in seconds. Events must have a created_
  "until": <an integer unix timestamp in seconds. Events must have a created_
  "limit": <maximum number of events relays SHOULD return in the initial quer
}
```

Upon receiving a `REQ` message, the relay SHOULD return events that match the filter. Any new events it receives SHOULD be sent to that same websocket until the connection is closed, a `CLOSE` event is received with the same `<subscription_id>`, or a new `REQ` is sent using the same `<subscription_id>` (in which case a new subscription is created, replacing the old one).

Filter attributes containing lists (`ids`, `authors`, `kinds` and tag filters like `#e`) are JSON arrays with one or more values. At least one of the arrays' values must match the relevant field in an event for the condition to be considered a match. For scalar event attributes such as `authors` and `kind`, the attribute from the event must be contained in the filter list. In the case of tag attributes such as `#e`, for which an event may have multiple values, the event and filter condition values must have at least one item in common.

The `ids`, `authors`, `#e` and `#p` filter lists MUST contain exact 64-character lowercase hex values.

The `since` and `until` properties can be used to specify the time range of events returned in the subscription. If a filter includes the `since` property, events with `created_at` greater than or equal to `since` are considered to match the filter. The `until` property is similar except that `created_at` must be less than or equal to `until`. In short, an event matches a filter if `since <= created_at <= until` holds.

All conditions of a filter that are specified must match for an event for it to pass the filter, i.e., multiple conditions are interpreted as `&&` conditions.

A `REQ` message may contain multiple filters. In this case, events that match any of the filters are to be returned, i.e., multiple filters are to be interpreted as `||` conditions.

The `limit` property of a filter is only valid for the initial query and MUST be ignored afterwards. When `limit: n` is present it is assumed that the events returned in the initial query will be the last `n` events ordered by the `created_at`. Newer events should appear first, and in the case of ties the event with the lowest id (first in lexical order) should be first. Relays SHOULD use the `limit` value to guide how many events are returned in the initial response. Returning fewer events is acceptable, but returning (much) more should be avoided to prevent overwhelming clients.

From relay to client: sending events and notices

Relays can send 5 types of messages, which must also be JSON arrays, according to the following patterns:

- `["EVENT", <subscription_id>, <event JSON as defined above>]`, used to send events requested by clients.
- `["OK", <event_id>, <true|false>, <message>]`, used to indicate acceptance or denial of an `EVENT` message.
- `["EOSE", <subscription_id>]`, used to indicate the *end of stored events* and the beginning of events newly received in real-time.
- `["CLOSED", <subscription_id>, <message>]`, used to indicate that a subscription was ended on the server side.
- `["NOTICE", <message>]`, used to send human-readable error messages or other things to clients.

This NIP defines no rules for how `NOTICE` messages should be sent or treated.

- `EVENT` messages MUST be sent only with a subscription ID related to a subscription previously initiated by the client (using the `REQ` message above).
- `OK` messages MUST be sent in response to `EVENT` messages received from clients, they must have the 3rd parameter set to `true` when an event has been accepted by the relay, `false` otherwise. The 4th parameter MUST always be present, but MAY be an empty string when the 3rd is `true`, otherwise it MUST be a string formed by a machine-readable single-word prefix followed by a `:` and then a human-readable message. Some examples:
  - `["OK", "b1a649ebe8...", true, ""]`
  - `["OK", "b1a649ebe8...", true, "pow: difficulty 25>=24"]`
  - `["OK", "b1a649ebe8...", true, "duplicate: already have this event"]`
  - `["OK", "b1a649ebe8...", false, "blocked: you are banned from posting here"]`
  - `["OK", "b1a649ebe8...", false, "blocked: please register your pubkey at https://my-expensive-relay.example.com"]`

    - `["OK", "b1a649ebe8...", false, "rate-limited: slow down there chief"]`
    - `["OK", "b1a649ebe8...", false, "invalid: event creation date is too far off from the current time"]`
    - `["OK", "b1a649ebe8...", false, "pow: difficulty 26 is less than 30"]`
    - `["OK", "b1a649ebe8...", false, "restricted: not allowed to write."]`
    - `["OK", "b1a649ebe8...", false, "error: could not connect to the database"]`
    - `["OK", "b1a649ebe8...", false, "mute: no one was listening to your ephemeral event and it wasn't handled in any way, it was ignored"]`
- `CLOSED` messages MUST be sent in response to a `REQ` when the relay refuses to fulfill it. It can also be sent when a relay decides to kill a subscription on its side before a client has disconnected or sent a `CLOSE`. This message uses the same pattern of `OK` messages with the machine-readable prefix and human-readable message. Some examples:
    - `["CLOSED", "sub1", "unsupported: filter contains unknown elements"]`
    - `["CLOSED", "sub1", "error: could not connect to the database"]`
    - `["CLOSED", "sub1", "error: shutting down idle subscription"]`
- The standardized machine-readable prefixes for `OK` and `CLOSED` are: `duplicate`, `pow`, `blocked`, `rate-limited`, `invalid`, `restricted`, `mute` and `error` for when none of that fits.

NIP-02

Follow List

`final` `optional`

A special event with kind `3`, meaning "follow list" is defined as having a list of `p` tags, one for each of the followed/known profiles one is following.

Each tag entry should contain the key for the profile, a relay URL where events from that key can be found (can be set to an empty string if not needed), and a local name (or "petname") for that profile (can also be set to an empty string or not provided), i.e., `["p", <32-bytes hex key>, <main relay URL>, <petname>]`.

The `.content` is not used.

For example:

```
{
  "kind": 3,
  "tags": [
    ["p", "91cf9..4e5ca", "wss://alicerelay.com/", "alice"],
    ["p", "14aeb..8dad4", "wss://bobrelay.com/nostr", "bob"],
    ["p", "612ae..e610f", "ws://carolrelay.com/ws", "carol"]
  ],
  "content": "",
  // other fields...
}
```

Every new following list that gets published overwrites the past ones, so it should contain all entries. Relays and clients SHOULD delete past following lists as soon as they receive a new one.

Whenever new follows are added to an existing list, clients SHOULD append them to the end of the list, so they are stored in chronological order.

Uses

Follow list backup

If one believes a relay will store their events for sufficient time, they can use this kind-3 event to backup their following list and recover on a different device.

Profile discovery and context augmentation

A client may rely on the kind-3 event to display a list of followed people by profiles one is browsing; make lists of suggestions on who to follow based on the follow lists of other people one might be following or browsing; or show the data in other contexts.

Relay sharing

A client may publish a follow list with good relays for each of their follows so other clients may use these to update their internal relay lists if needed, increasing censorship-resistance.

Petname scheme

The data from these follow lists can be used by clients to construct local **“petname”** tables derived from other people’s follow lists. This alleviates the need for global human-readable names. For example:

A user has an internal follow list that says

```
[
  ["p", "21df6d143fb96c2ec9d63726bf9edc71", "", "erin"]
]
```

And receives two follow lists, one from `21df6d143fb96c2ec9d63726bf9edc71` that says

```
[
  ["p", "a8bb3d884d5d90b413d9891fe4c4e46d", "", "david"]
]
```

and another from `a8bb3d884d5d90b413d9891fe4c4e46d` that says

```
[
  ["p", "f57f54057d2a7af0efecc8b0b66f5708", "", "frank"]
]
```

When the user sees `21df6d143fb96c2ec9d63726bf9edc71` the client can show *erin* instead; When the user sees `a8bb3d884d5d90b413d9891fe4c4e46d` the client can show *david.erin* instead; When the user sees `f57f54057d2a7af0efecc8b0b66f5708` the client can show *frank.david.erin* instead.

NIP-03

OpenTimestamps Attestations for Events

`draft` `optional`

This NIP defines an event with `kind:1040` that can contain an **OpenTimestamps** proof for any other event:

```
{
  "kind": 1040
  "tags": [
    ["e", <target-event-id>, <relay-url>],
    ["k", "<target-event-kind>"]
  ],
  "content": <base64-encoded OTS file data>
}
```

- The OpenTimestamps proof MUST prove the referenced `e` event id as its digest.
- The `content` MUST be the full content of an `.ots` file containing at least one Bitcoin attestation. This file SHOULD contain a **single** Bitcoin attestation (as not more than one valid attestation is necessary and less bytes is better than more) and no reference to "pending" attestations since they are useless in this context.

Example OpenTimestamps proof verification flow

Using `nak` , `jq` and `ots` :

```
~> nak req -i e71c6ea722987debdb60f81f9ea4f604b5ac0664120dd64fb9d23abc4ec7c32
> using an esplora server at https://blockstream.info/api
- sequence ending on block 810391 is valid
timestamp validated at block [810391]
```

> **Warning** `unrecommended`: deprecated in favor of **NIP-17**

NIP-04

Encrypted Direct Message

`final` `unrecommended` `optional` `relay`

A special event with kind `4`, meaning “encrypted direct message”. It is supposed to have the following attributes:

`content` MUST be equal to the base64-encoded, aes-256-cbc encrypted string of anything a user wants to write, encrypted using a shared cipher generated by combining the recipient’s public-key with the sender’s private-key; this appended by the base64-encoded initialization vector as if it was a querystring parameter named “iv”. The format is the following: `"content": "<encrypted_text>?iv=<initialization_vector>"`.

`tags` MUST contain an entry identifying the receiver of the message (such that relays may naturally forward this event to them), in the form `["p", "<pubkey, as a hex string>"]`.

`tags` MAY contain an entry identifying the previous message in a conversation or a message we are explicitly replying to (such that contextual, more organized conversations may happen), in the form `["e", "<event_id>"]`.

Note: By default in the **libsecp256k1** ECDH implementation, the secret is the SHA256 hash of the shared point (both X and Y coordinates). In Nostr, only the X coordinate of the shared point is used as the secret and it is NOT hashed. If using libsecp256k1, a custom function that copies the X coordinate must be passed as the `hashfp` argument in `secp256k1_ecdh`. See **here** .

Code sample for generating such an event in JavaScript:

```
import crypto from 'crypto'
import * as secp from '@noble/secp256k1'

let sharedPoint = secp.getSharedSecret(ourPrivateKey, '02' + theirPublicKey)
let sharedX = sharedPoint.slice(1, 33)

let iv = crypto.randomFillSync(new Uint8Array(16))
var cipher = crypto.createCipheriv(
  'aes-256-cbc',
  Buffer.from(sharedX),
  iv
)
let encryptedMessage = cipher.update(text, 'utf8', 'base64')
encryptedMessage += cipher.final('base64')
let ivBase64 = Buffer.from(iv.buffer).toString('base64')

let event = {
  pubkey: ourPubKey,
  created_at: Math.floor(Date.now() / 1000),
  kind: 4,
  tags: [['p', theirPublicKey]],
  content: encryptedMessage + '?iv=' + ivBase64
}
```

Security Warning

This standard does not go anywhere near what is considered the state-of-the-art in encrypted communication between peers, and it leaks metadata in the events, therefore it must not be used for anything you really need to keep secret, and only with relays that use `AUTH` to restrict who can fetch your `kind:4` events.

Client Implementation Warning

Clients *should not* search and replace public key or note references from the `.content`. If processed like a regular text note (where `@npub...` is replaced with

`#[0]` with a `["p", "..."]` tag) the tags are leaked and the mentioned user will receive the message in their inbox.

NIP-05

Mapping Nostr keys to DNS-based internet identifiers

`final` `optional`

On events of kind `0` (`user metadata`) one can specify the key `"nip05"` with an **internet identifier** (an email-like address) as the value. Although there is a link to a very liberal "internet identifier" specification above, NIP-05 assumes the `<local-part>` part will be restricted to the characters `a-z0-9-_.`, case-insensitive.

Upon seeing that, the client splits the identifier into `<local-part>` and `<domain>` and use these values to make a GET request to `https://<domain>/.well-known/nostr.json?name=<local-part>`.

The result should be a JSON document object with a key `"names"` that should then be a mapping of names to hex formatted public keys. If the public key for the given `<name>` matches the `pubkey` from the `user metadata` event, the client then concludes that the given pubkey can indeed be referenced by its identifier.

Example

If a client sees an event like this:

```
{
  "pubkey": "b0635d6a9851d3aed0cd6c495b282167acf761729078d975fc341b22650b07b9
  "kind": 0,
  "content": "{\"name\": \"bob\", \"nip05\": \"bob@example.com\"}"
  // other fields...
}
```

It will make a GET request to `https://example.com/.well-known/nostr.json?name=bob` and get back a response that will look like

```
{
  "names": {
    "bob": "b0635d6a9851d3aed0cd6c495b282167acf761729078d975fc341b22650b07b9"
  }
}
```

or with the **recommended** `"relays"` attribute:

```
{
  "names": {
    "bob": "b0635d6a9851d3aed0cd6c495b282167acf761729078d975fc341b22650b07b9"
  },
  "relays": {
    "b0635d6a9851d3aed0cd6c495b282167acf761729078d975fc341b22650b07b9": [ "ws
  }
}
```

If the pubkey matches the one given in `"names"` (as in the example above) that means the association is right and the `"nip05"` identifier is valid and can be displayed.

The recommended `"relays"` attribute may contain an object with public keys as properties and arrays of relay URLs as values. When present, that can be used to help clients learn in which relays the specific user may be found. Web servers

which serve `/.well-known/nostr.json` files dynamically based on the query string SHOULD also serve the relays data for any name they serve in the same reply when that is available.

Finding users from their NIP-05 identifier

A client may implement support for finding users' public keys from *internet identifiers*, the flow is the same as above, but reversed: first the client fetches the *well-known* URL and from there it gets the public key of the user, then it tries to fetch the kind `0` event for that user and check if it has a matching `"nip05"`.

Notes

Identification, not verification

The NIP-05 is not intended to *verify* a user, but only to *identify* them, for the purpose of facilitating the exchange of a contact or their search.
Exceptions are people who own (e.g., a company) or are connected (e.g., a project) to a well-known domain, who can exploit NIP-05 as an attestation of their relationship with it, and thus to the organization behind it, thereby gaining an element of trust.

User discovery implementation suggestion

A client can use this to allow users to search other profiles. If a client has a search box or something like that, a user may be able to type "**bob@example.com** " there and the client would recognize that and do the proper queries to obtain a pubkey and suggest that to the user.

Clients must always follow public keys, not NIP-05 addresses

For example, if after finding that `bob@bob.com` has the public key `abc...def`, the user clicks a button to follow that profile, the client must keep a primary reference

to `abc...def`, not `bob@bob.com`. If, for any reason, the address `https://bob.com/.well-known/nostr.json?name=bob` starts returning the public key `1d2...e3f` at any time in the future, the client must not replace `abc...def` in his list of followed profiles for the user (but it should stop displaying "**bob@bob.com**" for that user, as that will have become an invalid `"nip05"` property).

Public keys must be in hex format

Keys must be returned in hex format. Keys in NIP-19 `npub` format are only meant to be used for display in client UIs, not in this NIP.

Showing just the domain as an identifier

Clients may treat the identifier `_@domain` as the "root" identifier, and choose to display it as just the `<domain>`. For example, if Bob owns `bob.com`, he may not want an identifier like `bob@bob.com` as that is redundant. Instead, Bob can use the identifier `_@bob.com` and expect Nostr clients to show and treat that as just `bob.com` for all purposes.

Reasoning for the `/.well-known/nostr.json?name=<local-part>` format

By adding the `<local-part>` as a query string instead of as part of the path, the protocol can support both dynamic servers that can generate JSON on-demand and static servers with a JSON file in it that may contain multiple names.

Allowing access from JavaScript apps

JavaScript Nostr apps may be restricted by browser **CORS** policies that prevent them from accessing `/.well-known/nostr.json` on the user's domain. When CORS prevents JS from loading a resource, the JS program sees it as a network failure identical to the resource not existing, so it is not possible for a pure-JS app to tell the user for certain that the failure was caused by a CORS issue. JS Nostr apps that

see network failures requesting `/.well-known/nostr.json` files may want to recommend to users that they check the CORS policy of their servers, e.g.:

```
$ curl -sI https://example.com/.well-known/nostr.json?name=bob | grep -i ^Acc
Access-Control-Allow-Origin: *
```

Users should ensure that their `/.well-known/nostr.json` is served with the HTTP header `Access-Control-Allow-Origin: *` to ensure it can be validated by pure JS apps running in modern browsers.

Security Constraints

The `/.well-known/nostr.json` endpoint MUST NOT return any HTTP redirects.

Fetchers MUST ignore any HTTP redirects given by the `/.well-known/nostr.json` endpoint.

NIP-06

Basic key derivation from mnemonic seed phrase

`draft` `optional`

BIP39 is used to generate mnemonic seed words and derive a binary seed from them.

BIP32 is used to derive the path `m/44'/1237'/<account>'/0/0` (according to the Nostr entry on **SLIP44**).

A basic client can simply use an `account` of `0` to derive a single key. For more advanced use-cases you can increment `account`, allowing generation of practically infinite keys from the 5-level path with hardened derivation.

Other types of clients can still get fancy and use other derivation paths for their own other purposes.

Test vectors

mnemonic: leader monkey parrot ring guide accident before fence cannon height naive bean
private key (hex):
7f7ff03d123792d6ac594bfa67bf6d0c0ab55b6b1fdb6249303fe861f1ccba9a
nsec: nsec10allq0gjx7fddtzef0ax00mdps9t2kmtrldkyjfs8l5xruwvh2dq0lhhkp
public key (hex):
17162c921dc4d2518f9a101db33695df1afb56ab82f5ff3e5da6eec3ca5cd917
npub: npub1zutzeysacnf9rru6zqwmxd54mud0k44tst6l70ja5mhv8jjumytsd2x7nu

---

mnemonic: what bleak badge arrange retreat wolf trade produce cricket blur garlic valid proud rude strong choose busy staff weather area salt hollow arm fade
private key (hex):
c15d739894c81a2fcfd3a2df85a0d2c0dbc47a280d092799f144d73d7ae78add
nsec: nsec1c9wh8xy5eqdzln7n5t0ctgxjcrdug73gp5yj0x03gntn67h83twssdfhel
public key (hex):
d41b22899549e1f3d335a31002cfd382174006e166d3e658e3a5eecdb6463573
npub: npub16sdj9zv4f8sl85e45vgq9n7nsgt5qphpvmf7vk8r5hhvmdjxx4es8rq74h

NIP-07

`window.nostr` capability for web browsers

`draft` `optional`

The `window.nostr` object may be made available by web browsers or extensions and websites or web-apps may make use of it after checking its availability.

That object must define the following methods:

```
async window.nostr.getPublicKey(): string // returns a public key as hex
async window.nostr.signEvent(event: { created_at: number, kind: number, tags:
```

Aside from these two basic above, the following functions can also be implemented optionally:

```
async window.nostr.nip04.encrypt(pubkey, plaintext): string // returns cipher
async window.nostr.nip04.decrypt(pubkey, ciphertext): string // takes ciphert
async window.nostr.nip44.encrypt(pubkey, plaintext): string // returns cipher
async window.nostr.nip44.decrypt(pubkey, ciphertext): string // takes ciphert
```

Recommendation to Extension Authors

To make sure that the `window.nostr` is available to nostr clients on page load, the authors who create Chromium and Firefox extensions should load their scripts by specifying `"run_at": "document_end"` in the extension's manifest.

Implementation

See **https://github.com/aljazceru/awesome-nostr#nip-07-browser-extensions** .

> **Warning** `unrecommended`: deprecated in favor of **NIP-27**

NIP-08

Handling Mentions

`final` `unrecommended` `optional`

This document standardizes the treatment given by clients of inline mentions of other events and pubkeys inside the content of `text_note`s.

Clients that want to allow tagged mentions they MUST show an autocomplete component or something analogous to that whenever the user starts typing a special key (for example, "@") or presses some button to include a mention etc – or these clients can come up with other ways to unambiguously differentiate between mentions and normal text.

Once a mention is identified, for example, the pubkey `27866e9d854c78ae625b867eefdfa9580434bc3e675be08d2acb526610d96fbe`, the client MUST add that pubkey to the `.tags` with the tag `p`, then replace its textual reference (inside `.content`) with the notation `#[index]` in which "index" is equal to the 0-based index of the related tag in the tags array.

The same process applies for mentioning event IDs.

A client that receives a `text_note` event with such `#[index]` mentions in its `.content` CAN do a search-and-replace using the actual contents from the `.tags` array with the actual pubkey or event ID that is mentioned, doing any desired context augmentation (for example, linking to the pubkey or showing a preview of the mentioned event contents) it wants in the process.

Where `#[index]` has an `index` that is outside the range of the tags array or points to a tag that is not an `e` or `p` tag or a tag otherwise declared to support this notation, the client MUST NOT perform such replacement or augmentation, but instead display it as normal text.

NIP-09

Event Deletion Request

`draft` `optional` `relay`

A special event with kind `5`, meaning "deletion request" is defined as having a list of one or more `e` or `a` tags, each referencing an event the author is requesting to be deleted. Deletion requests SHOULD include a `k` tag for the kind of each event being requested for deletion.

The event's `content` field MAY contain a text note describing the reason for the deletion request.

For example:

```
{
  "kind": 5,
  "pubkey": <32-bytes hex-encoded public key of the event creator>,
  "tags": [
    ["e", "dcd59..464a2"],
    ["e", "968c5..ad7a4"],
    ["a", "<kind>:<pubkey>:<d-identifier>"],
    ["k", "1"],
    ["k", "30023"]
  ],
  "content": "these posts were published by accident",
  // other fields...
}
```

Relays SHOULD delete or stop publishing any referenced events that have an identical `pubkey` as the deletion request. Clients SHOULD hide or otherwise indicate a deletion request status for referenced events.

Relays SHOULD continue to publish/share the deletion request events indefinitely, as clients may already have the event that's intended to be deleted. Additionally, clients SHOULD broadcast deletion request events to other relays which don't have it.

When an `a` tag is used, relays SHOULD delete all versions of the replaceable event up to the `created_at` timestamp of the deletion request event.

Client Usage

Clients MAY choose to fully hide any events that are referenced by valid deletion request events. This includes text notes, direct messages, or other yet-to-be defined event kinds. Alternatively, they MAY show the event along with an icon or other indication that the author has "disowned" the event. The `content` field MAY also be used to replace the deleted events' own content, although a user interface should clearly indicate that this is a deletion request reason, not the original content.

A client MUST validate that each event `pubkey` referenced in the `e` tag of the deletion request is identical to the deletion request `pubkey`, before hiding or deleting any event. Relays can not, in general, perform this validation and should not be treated as authoritative.

Clients display the deletion request event itself in any way they choose, e.g., not at all, or with a prominent notice.

Clients MAY choose to inform the user that their request for deletion does not guarantee deletion because it is impossible to delete events from all relays and clients.

Relay Usage

Relays MAY validate that a deletion request event only references events that have the same `pubkey` as the deletion request itself, however this is not required since relays may not have knowledge of all referenced events.

Deletion Request of a Deletion Request

Publishing a deletion request event against a deletion request has no effect. Clients and relays are not obliged to support "unrequest deletion" functionality.

NIP-10

Text Notes and Threads

`draft` `optional`

This NIP defines `kind:1` as a simple plaintext note.

Abstract

The `.content` property contains some human-readable text.

`e` tags can be used to define note thread roots and replies. They SHOULD be sorted by the reply stack from root to the direct parent.

`q` tags MAY be used when citing events in the `.content` with **NIP-21** .

```
["q", "<event-id> or <event-address>", "<relay-url>", "<pubkey-if-a-regular-e
```

Authors of the `e` and `q` tags SHOULD be added as `p` tags to notify of a new reply or quote.

Markup languages such as markdown and HTML SHOULD NOT be used.

Marked “e” tags (PREFERRED)

Kind 1 events with `e` tags are replies to other kind 1 events. Kind 1 replies MUST NOT be used to reply to other kinds, use **NIP-22** instead.

`["e", <event-id>, <relay-url>, <marker>, <pubkey>]`

Where:

- `<event-id>` is the id of the event being referenced.
- `<relay-url>` is the URL of a recommended relay associated with the reference. Clients SHOULD add a valid `<relay-url>` field, but may instead leave it as `""`.
- `<marker>` is optional and if present is one of `"reply"`, `"root"`.
- `<pubkey>` is optional, SHOULD be the pubkey of the author of the referenced event

Those marked with `"reply"` denote the id of the reply event being responded to. Those marked with `"root"` denote the root id of the reply thread being responded to. For top level replies (those replying directly to the root event), only the `"root"` marker should be used.

A direct reply to the root of a thread should have a single marked "e" tag of type "root".

> This scheme is preferred because it allows events to mention others without confusing them with `<reply-id>` or `<root-id>`.

`<pubkey>` SHOULD be the pubkey of the author of the `e` tagged event, this is used in the outbox model to search for that event from the authors write relays where relay hints did not resolve the event.

The "p" tag

Used in a text event contains a list of pubkeys used to record who is involved in a reply thread.

When replying to a text event E the reply event's "p" tags should contain all of E's "p" tags as well as the `"pubkey"` of the event being replied to.

Example: Given a text event authored by `a1` with "p" tags [`p1`, `p2`, `p3`] then the "p" tags of the reply should be [`a1`, `p1`, `p2`, `p3`] in no particular order.

Deprecated Positional “e” tags

This scheme is not in common use anymore and is here just to keep backward compatibility with older events on the network.

Positional `e` tags are deprecated because they create ambiguities that are difficult, or impossible to resolve when an event references another but is not a reply.

They use simple `e` tags without any marker.

`["e", <event-id>, <relay-url>]` as per NIP-01.

Where:

- `<event-id>` is the id of the event being referenced.
- `<relay-url>` is the URL of a recommended relay associated with the reference. Many clients treat this field as optional.

The positions of the “e” tags within the event denote specific meanings as follows:

- No “e” tag:
  This event is not a reply to, nor does it refer to, any other event.

- One “e” tag:
  `["e", <id>]`: The id of the event to which this event is a reply.

- Two “e” tags: `["e", <root-id>]`, `["e", <reply-id>]`
  `<root-id>` is the id of the event at the root of the reply chain. `<reply-id>` is the id of the article to which this event is a reply.

- Many “e” tags: `["e", <root-id>]` `["e", <mention-id>]`, ..., `["e", <reply-id>]`
  There may be any number of `<mention-ids>`. These are the ids of events

which may, or may not be in the reply chain. They are citing from this event. `root-id` and `reply-id` are as above. NIP-11 ======

Relay Information Document

`draft` `optional` `relay`

Relays may provide server metadata to clients to inform them of capabilities, administrative contacts, and various server attributes. This is made available as a JSON document over HTTP, on the same URI as the relay's websocket.

When a relay receives an HTTP(s) request with an `Accept` header of `application/nostr+json` to a URI supporting WebSocket upgrades, they SHOULD return a document with the following structure.

```
{
  "name": <string identifying relay>,
  "description": <string with detailed information>,
  "banner": <a link to an image (e.g. in .jpg, or .png format)>,
  "icon": <a link to an icon (e.g. in .jpg, or .png format>,
  "pubkey": <administrative contact pubkey>,
  "self": <relay's own pubkey>,
  "contact": <administrative alternate contact>,
  "supported_nips": <a list of NIP numbers supported by the relay>,
  "software": <string identifying relay software URL>,
  "version": <string version identifier>,
  "privacy_policy": <a link to a text file describing the relay's privacy pol
  "terms_of_service": <a link to a text file describing the relay's term of s
}
```

Any field may be omitted, and clients MUST ignore any additional fields they do not understand. Relays MUST accept CORS requests by sending `Access-Control-Allow-Origin`, `Access-Control-Allow-Headers`, and `Access-Control-Allow-Methods` headers.

Field Descriptions

Name

A relay may select a `name` for use in client software. This is a string, and SHOULD be less than 30 characters to avoid client truncation.

Description

Detailed plain-text information about the relay may be contained in the `description` string. It is recommended that this contain no markup, formatting or line breaks for word wrapping, and simply use double newline characters to separate paragraphs. There are no limitations on length.

Banner

To make nostr relay management more user friendly, an effort should be made by relay owners to communicate with non-dev non-technical nostr end users. A banner is a visual representation of the relay. It should aim to visually communicate the brand of the relay, complementing the text `Description`. **Here is an example banner** mockup as visualized in Damus iOS relay view of the Damus relay.

Icon

Icon is a compact visual representation of the relay for use in UI with limited real estate such as a nostr user's relay list view. Below is an example URL pointing to an image to be used as an icon for the relay. Recommended to be squared in shape.

```
{
  "icon": "https://nostr.build/i/53866b44135a27d624e99c6165cabd76ac8f72797209
  // other fields...
}
```

Pubkey

An administrative contact may be listed with a `pubkey`, in the same format as Nostr events (32-byte hex for a `secp256k1` public key). If a contact is listed, this provides clients with a recommended address to send encrypted direct messages (See **NIP-17**) to a system administrator. Expected uses of this address are to report abuse or illegal content, file bug reports, or request other technical assistance.

Relay operators have no obligation to respond to direct messages.

Self

A relay MAY maintain an identity independent from its administrator using the `self` field, which MUST be a 32-byte hex public key. This allows relays to respond to requests with events published either in advance or on demand by their own key.

Contact

An alternative contact may be listed under the `contact` field as well, with the same purpose as `pubkey`. Use of a Nostr public key and direct message SHOULD be preferred over this. Contents of this field SHOULD be a URI, using schemes such as `mailto` or `https` to provide users with a means of contact.

Supported NIPs

As the Nostr protocol evolves, some functionality may only be available by relays that implement a specific `NIP`. This field is an array of the integer identifiers of `NIP`s that are implemented in the relay. Examples would include `1`, for `"NIP-01"` and `9`, for `"NIP-09"`. Client-side `NIPs` SHOULD NOT be advertised, and can be ignored by clients.

Software

The relay server implementation MAY be provided in the `software` attribute. If present, this MUST be a URL to the project's homepage.

Version

The relay MAY choose to publish its software version as a string attribute. The string format is defined by the relay implementation. It is recommended this be a version number or commit identifier.

Privacy Policy

The relay owner/admin MAY choose to link to a privacy policy document, which describes how the relay utilizes user data. Data collection, data usage, data retention, monetization of data, and third party data sharing SHOULD be included.

Terms of Service

The relay owner/admin MAY choose to link to a terms of service document.

Extra Fields

Server Limitations

These are limitations imposed by the relay on clients. Your client should expect that requests exceed these *practical* limitations are rejected or fail immediately.

```
{
  "limitation": {
    "max_message_length": 16384,
    "max_subscriptions": 300,
    "max_limit": 5000,
    "max_subid_length": 100,
    "max_event_tags": 100,
    "max_content_length": 8196,
    "min_pow_difficulty": 30,
    "auth_required": true,
    "payment_required": true,
    "restricted_writes": true,
    "created_at_lower_limit": 31536000,
    "created_at_upper_limit": 3,
    "default_limit": 500
  },
  // other fields...
}
```

- `max_message_length`: the maximum number of bytes for incoming JSON that the relay will attempt to decode and act upon. When you send large subscriptions, you will be limited by this value. It also effectively limits the maximum size of any event. Value is calculated from `[` to `]` after UTF-8 serialization (so some unicode characters will cost 2-3 bytes). It is equal to the maximum size of the WebSocket message frame.

- `max_subscriptions`: total number of subscriptions that may be active on a single websocket connection to this relay. Authenticated clients with a (paid) relationship to the relay may have higher limits.

- `max_subid_length`: maximum length of subscription id as a string.

- `max_limit`: the relay server will clamp each filter's `limit` value to this number. This means the client won't be able to get more than this number of events from a single subscription filter. This clamping is typically done silently

by the relay, but with this number, you can know that there are additional results if you narrow your filter's time range or other parameters.

- `max_event_tags`: in any event, this is the maximum number of elements in the `tags` list.
- `max_content_length`: maximum number of characters in the `content` field of any event. This is a count of unicode characters. After serializing into JSON it may be larger (in bytes), and is still subject to the `max_message_length`, if defined.
- `min_pow_difficulty`: new events will require at least this difficulty of PoW, based on **NIP-13**, or they will be rejected by this server.
- `auth_required`: this relay requires **NIP-42** authentication to happen before a new connection may perform any other action. Even if set to False, authentication may be required for specific actions.
- `payment_required`: this relay requires payment before a new connection may perform any action.
- `restricted_writes`: this relay requires some kind of condition to be fulfilled to accept events (not necessarily, but including `payment_required` and `min_pow_difficulty`). This should only be set to `true` when users are expected to know the relay policy before trying to write to it – like belonging to a special pubkey-based whitelist or writing only events of a specific niche kind or content. Normal anti-spam heuristics, for example, do not qualify.
- `created_at_lower_limit`: 'created_at' lower limit
- `created_at_upper_limit`: 'created_at' upper limit
- `default_limit`: The maximum returned events if you send a filter without a `limit`.

Event Retention

There may be a cost associated with storing data forever, so relays may wish to state retention times. The values stated here are defaults for unauthenticated users and visitors. Paid users would likely have other policies.

Retention times are given in seconds, with `null` indicating infinity. If zero is provided, this means the event will not be stored at all, and preferably an error will be provided when those are received.

```
{
  "retention": [
    {"kinds": [0, 1, [5, 7], [40, 49]], "time": 3600},
    {"kinds": [[40000, 49999]], "time": 100},
    {"kinds": [[30000, 39999]], "count": 1000},
    {"time": 3600, "count": 10000}
  ],
  // other fields...
}
```

`retention` is a list of specifications: each will apply to either all kinds, or a subset of kinds. Ranges may be specified for the kind field as a tuple of inclusive start and end values. Events of indicated kind (or all) are then limited to a `count` and/or time period.

It is possible to effectively blacklist Nostr-based protocols that rely on a specific `kind` number, by giving a retention time of zero for those `kind` values. While that is unfortunate, it does allow clients to discover servers that will support their protocol quickly via a single HTTP fetch.

There is no need to specify retention times for *ephemeral events* since they are not retained.

Content Limitations

Some relays may be governed by the arbitrary laws of a nation state. This may limit what content can be stored in clear-text on those relays. All clients are encouraged

to use encryption to work around this limitation.

It is not possible to describe the limitations of each country's laws and policies which themselves are typically vague and constantly shifting.

Therefore, this field allows the relay operator to indicate which countries' laws might end up being enforced on them, and then indirectly on their users' content.

Users should be able to avoid relays in countries they don't like, and/or select relays in more favorable zones. Exposing this flexibility is up to the client software.

```
{
  "relay_countries": [ "CA", "US" ],
  // other fields...
}
```

- `relay_countries`: a list of two-level ISO country codes (ISO 3166-1 alpha-2) whose laws and policies may affect this relay. `EU` may be used for European Union countries. A `*` can be used for global relays.

Remember that a relay may be hosted in a country which is not the country of the legal entities who own the relay, so it's very likely a number of countries are involved.

Community Preferences

For public text notes at least, a relay may try to foster a local community. This would encourage users to follow the global feed on that relay, in addition to their usual individual follows. To support this goal, relays MAY specify some of the following values.

```
{
  "language_tags": ["en", "en-419"],
  "tags": ["sfw-only", "bitcoin-only", "anime"],
  "posting_policy": "https://example.com/posting-policy.html",
  // other fields...
}
```

- `language_tags` is an ordered list of **IETF language tags** indicating the major languages spoken on the relay. A `*` can be used for global relays.
- `tags` is a list of limitations on the topics to be discussed. For example `sfw-only` indicates that only "Safe For Work" content is encouraged on this relay. This relies on assumptions of what the "work" "community" feels "safe" talking about. In time, a common set of tags may emerge that allow users to find relays that suit their needs, and client software will be able to parse these tags easily. The `bitcoin-only` tag indicates that any *altcoin*, *"crypto"* or *blockchain* comments will be ridiculed without mercy.
- `posting_policy` is a link to a human-readable page which specifies the community policies for the relay. In cases where `sfw-only` is True, it's important to link to a page which gets into the specifics of your posting policy.

The `description` field should be used to describe your community goals and values, in brief. The `posting_policy` is for additional detail and legal terms. Use the `tags` field to signify limitations on content, or topics to be discussed, which could be machine processed by appropriate client software.

Pay-to-Relay

Relays that require payments may want to expose their fee schedules.

```
{
  "payments_url": "https://my-relay/payments",
  "fees": {
    "admission": [{ "amount": 1000000, "unit": "msats" }],
    "subscription": [{ "amount": 5000000, "unit": "msats", "period": 2592000
    "publication": [{ "kinds": [4], "amount": 100, "unit": "msats" }],
  },
  // other fields...
}
```

Examples

As of 25 March 2025 the following command provided these results:

```
curl -H "Accept: application/nostr+json" https://jellyfish.land | jq
```

```
{
  "name": "JellyFish",
  "description": "Stay Immortal!",
  "banner": "https://image.nostr.build/7fdefea2dec1f1ec25b8ce69362566c13b2b7f
  "pubkey": "bf2bee5281149c7c350f5d12ae32f514c7864ff10805182f4178538c2c421007
  "contact": "hi@dezh.tech",
  "software": "https://github.com/dezh-tech/immortal",
  "supported_nips": [
    1,
    9,
    11,
    13,
    17,
    40,
    42,
    59,
    62,
    70
  ],
  "version": "immortal - 0.0.9",
  "relay_countries": [
    "*"
  ],
  "language_tags": [
    "*"
  ],
  "tags": [],
  "posting_policy": "https://jellyfish.land/tos.txt",
  "payments_url": "https://jellyfish.land/relay",
  "icon": "https://image.nostr.build/2547e9ec4b23589e09bc7071e0806c3d4293f762
  "retention": [],
  "fees": {
    "subscription": [
      {
        "amount": 3000,
        "period": 2628003,
        "unit": "sats"
      },
      {
        "amount": 8000,
```