The Federation API

Final Solution's most prominent feature over other imageboard software is its ability to federate across multiple nodes. This document describes the object format and public API for this federation system.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

Table of Contents

The message format

All federation messages contain a few required parameters.

```
{
    "nodeid": "NODEID", // The node identifier.
    "type": "newpost", // The type of the message.
    "content-type": "application/json", // The content type of this message.
    "content": "{ ... }", // The content of the node's message.
    "signature": "...", // Base64 encoded PGP signature
}
```

The parameters are given in more detail below:

`nodeid`
: The short identifier of the node. This will be between 3 and 6 characters. Node IDs shorter than 3 characters will be rejected by the reference implementation. This ID MUST be known and trusted by the node receiving the message. The protocol for trusting is explained in The federation trust mechanism.

`type`
: The type of the message being sent. See Message types.

`content-type`
: The type of the content inside the content variable. See Parsing message contents.

`content`
: The full content of the message. The data will be parsed based on the content-type parameter.

`signature`
: Base64-encoded PGP signature for the content string. This signature MUST be created with the publicly advertised RSA key as defined by The federation trust mechanism.

Parsing message contents

The contents of the message MUST be parsed based on the `content-type` parameter. The `content-type` parameter MUST be in two formats:

- A string containing just a mime-type, such as `application/json`. The encoding is assumed to be `plain`.
- A string containing a mime-type and an encoding, separated with a semicolon, i.e. `application/json;base64`.

The following mime-types are standard and MUST be supported by a compliant node:

`text/plain`
: Plaintext content to be used verbatim.

`application/json`
: A JSON encoded value, such as an object or an array.

The following encodings are standard and MUST be supported by a compliant node:

`plain`
: No special encoding is used.

`base64`
: The content is encoded with the Base64 format as defined by RFC4648 Section 4.

`urlencoded`
: The content is percent-encoded as defined by RFC3986 Section 2.

The federation trust mechanism

In order for a federation node to get its messages accepted by other nodes, those nodes MUST first trust the node. For this to happen, the federation node MUST advertise its NodeID and public key via the HTTP API as specified in /federation/node.

When a federation node decides to trust another node, it MUST retrieve the information about this node via a `GET` request to /federation/node, and then store this information for future use. However, if the node has already trusted another node with the same NodeID before, the node MUST NOT trust the foreign node unless approved explicitly by the node administrator.

Federation nodes MUST advertise nodes that they have EXPLICITLY trusted via the HTTP API as specified in /federation/trusted. Other nodes which already trust this node MAY automatically trust the advertised nodes. However, if a node decides that it will no longer trust a specific node, it SHOULD stop trusting nodes it implicitly trusted.

Nodes MAY recursively trust nodes implicitly based on their explicitly trusted nodes; if a node decides to recursively trust other nodes, it SHOULD set a limit on the recursion depth. The recommended recursion depth limit is 2.

Post and thread identifier format

On federated boards, multiple federation nodes will be collaborating on the same board. To prevent race conditions, the identifiers are in a `NODEID:POSTID` format for all threads and posts.

Post and thread attachments

New posts and threads can have a number of files. A node receiving new posts and threads SHOULD list the files alongside them.

The structure of a single file is as follows:

```
{
    "filename": "image.jpg", // The original name of the file.
    "url": "https://imageboard.xyz/files/a0d5c...7a9.jpg", // The full path to the file.
    "thumbnail": "https://imageboard.xyz/thumbs/a0d5c...7a9.jpg", // The full path to the file's thumbnail.
    "mime": "image/jpeg", // The MIME-type of the file.
    "hash": "a0d5c...7a9", // The SHA256 hash of the file.
    "attributes": { // MAY contain any of the following attributes.
        "width": 200, // Width of the file, if applicable. May not exist.
        "height": 200, // Height of the file, if applicable. May not exist.
        "duration": 300, // Duration of the file, if applicable. May not exist.
        "pages": 10, // The page count of the file, if applicable. May not exist.
    },
    "spoiler": false, // Whether the file was spoilered.
}
```

Files are stored in their originating nodes. The receiving node MAY mirror the file, but it is not required. If a file is not stored locally and is deleted on the originating node, the originating node MUST send out a filedelete message to the other nodes, described below.

Message types

`newpost`

A new post was made to a board.

Parameters:

`board_uri`

: The unique URI of the board which the content was posted to. The receiving node MAY ignore this message if they are not serving the board which this post was made on. The receiving node SHOULD ignore this message if they have explicitly disabled/blacklisted the board the post was made on.

`thread_id`

: The ID of the thread, as described in Post and thread identifier format. The receiving node MUST reject this message and return an error if the thread does not exist on the board. The receiving node SHOULD ignore this message if the thread was locally removed.

`post_id`

: The ID of the post as an integer. The receiving node MUST reject this message and return an error if another post with the same ID already exists from the same node.

`name`

: The name field of the post as a string. This value MUST be 32 characters long at most.

`subject`

: The subject field of the post as a string. This value MUST be 64 characters long at most.

`options`

: The options field of the post as a string. This value MUST be 64 characters long at most.

`body`

: The body of the post. The body content can be parsed by the receiving node in any way needed. The node SHOULD truncate the post body if its length is over the limit specified by the node.

`body_html`

The body of the post in HTML format. The receiving node MAY ignore this field if it decides to parse the post body by itself. The node SHOULD truncate the body HTML if its length is over the limit specified by the node.

`poster_id`

The thread- and node-specific ID of the poster. The receiving node MUST accept posts with the same poster ID coming from a certain node as the same user, and act accordingly.

`files`

A list of files, the format of which is specified in Post and thread attachments.

`newthread`

A new thread was created on a board.

Parameters:

`board_uri`

The unique URI of the board which the content was posted to. The receiving node MAY ignore this message if they are not serving the board which this post was made on. The receiving node SHOULD ignore this message if they have explicitly disabled/blacklisted the board the post was made on.

`thread_id`

The ID of the thread as an integer. The receiving node MUST reject this message and return an error if another thread with the same ID already exists from the same node.

`name`

The name field of the thread as a string. This value MUST be 32 characters long at most.

`subject`

The subject field of the thread as a string. This value MUST be 64 characters long at most.

`options`

The options field of the thread as a string. This value MUST be 64 characters long at most.

`body`

The body of the thread. The body content can be parsed by the receiving node in any way needed. The node SHOULD truncate the thread body if its length is over the limit specified by the node.

`body_html`

The body of the thread in HTML format. The receiving node MAY ignore this field if it decides to parse the thread body by itself. The node SHOULD truncate the body HTML if its length is over the limit specified by the node.

`poster_id`

The thread- and node-specific ID of the poster. The receiving node MUST accept posts with the same poster ID coming from a certain node as the same user, and act accordingly.

`files`

A list of files, the format of which is specified in Post and thread attachments.

`filedelete`

One or more files were deleted from a certain node.

Parameters:

`hashes` A list of SHA256 hashes which correspond to the files that have been deleted.

The Federation HTTP API

All API request and response content MUST be in JSON format (content type `application/json`).

`/federation/`

POST

The target of all federation messages. Other nodes must send the messages described in this document to this endpoint.

`/federation/node`

GET

Retrieve information about the federation node.

Example response:

```
{
    "nodeid": "NODEID",
    // -----BEGIN PGP PUBLIC KEY BLOCK-----...
    "pubkey": "LS0tLS1CRUdJTiBQR1AgUFVCTElDIEtFWSBCTE9DSy0tLS0t...",
    "fingerprint": "0123456789ABCDEF0123456789ABCDEF01234567"
}
```

Parameters:

`nodeid`
: The unique identifier of this node. Other nodes will not trust this node if another node with the same identifier was previously trusted, so the identifier must not exist on the network beforehand. See The federation trust mechanism.

`pubkey`
: The base64-encoded PGP public key block for this node. This public key will get stored by other nodes and will be used to verify the authenticity of the messages from this node.

`fingerprint`
: The RSA fingerprint of the PGP key for this node. The string MUST be exactly 40 characters long.

`/federation/trusted`

GET

Retrieve the federation nodes this node explicitly trusts.

Example response:

```
{
    "nodes": [
        {
            "nodeid": "NODEA",
            "info": "https://api.achan.org/federation/node"
        },
        {
            "nodeid": "NODEB",
```

```
            "info": "https://api.bchan.org/federation/node"
        },
        // ...
    ]
}
```

Parameters:

`nodes`

A list of nodes this node explicitly trusts. Each item is an object containing the following properties:

`nodeid`

The ID of the trusted node.

`info`

An absolute URL pointing to the */federation/node* endpoint of the trusted node.