IRCv3 Specifications

IRCv3 specifications build on top of the core IRC protocol. The primary core is described by the "Modern IRC" specification, which supersedes historical IRC protocol specifications (RFC1459, RFC2812 and RFC7194). To fully understand IRCv3, please read the "Modern IRC" specification followed by the IRCv3 specifications.

The IRCv3 specifications are released when they are stable and have been widely tested. In the past the WG released specifications as versioned bundles (IRCv3.1, IRCv3.2), but we no longer do this.

Errata updates may be submitted for our specifications. To do so, simply see our contribution document.

Capability Negotiation

Capabilities let us implement protocol changes in backwards-compatible ways. They also convey various information on joining a server. Capability negotiation is a vital part of IRCv3, and lets clients request and enable the CAPs they support once they've joined a server.

The Capability Negotiation spec conveys the basic listing and requesting of capabilities, and lays the framework which most IRCv3 specs use. It also goes over the 302 extensions, and `cap-notify` – a feature to let clients know when capabilities are added to or removed from the server (for example, if the SASL authentication layer disconnects, the associated capability may be disabled for a time).

Message Tags

Message tags extend the core framing used with IRC messages, and allow extended data to be sent with messages.

Message tags are widely used in the IRCv3 specifications. As such, most software implementing IRCv3 extensions will want to to implement the core Message Tags specification.

The Message Tags spec covers the new message format, how tag data is formatted and escaped, and how they are named. In addition, it extends the message length and lets clients and bots discover between each other, allowing new features to be developed and implemented independently from the IRC servers themselves (similar to extensions based on CTCP).

Note: Message tags themselves are used as a foundation for other extensions and do not themselves offer any user-facing features. Specific message tags are defined in the various IRCv3 specifications.

Account Authentication and Registration

Accounts have become a first-class feature on IRC. They store channel access and ownership information, preferences, settings, nickname ownership, and even more. Our extensions describe simple account creation and authentication.

The **work-in-progress** Account Registration spec lets clients build nice account registration interfaces, instead of making users manually send messages to service bots.

SASL let's users authenticate in a standardised way across different IRC networks. It gives a way to authenticate while connecting, without having to message service bots after they've joined the server. Because SASL allows authentication before they've finished connecting, it lets clients join private and restricted channels without having to setup complex join-wait systems.

The v3.1 SASL spec defines the `AUTHENTICATE` command and `sasl` cap, which work together to allow clients to authenticate to the network.

The v3.2 SASL spec defines a way to advertise the authentication methods available to clients, allows for clients to re-authenticate after services is lost and reconnects, and defines what to do if the authentication layer is disconnected or reconnected.

IRC SASL authentication primarily uses the same mechanisms as SASL in other protocols. Most commonly:

- PLAIN as defined by RFC 4616
- EXTERNAL as defined by RFC 4422
- SCRAM-SHA-256 as defined by RFC 7677

For further information on SASL mechanism support, see the SASL Mechanisms page.

Account Tracking

IRCv3 extensions allow clients to much more easily know when other users are logged into accounts. This allows for much greater integration between client bots and the network's authentication system, as well as better general display and authentication of client identities.

The `account-extban` spec defines an ISUPPORT token allowing clients to construct an EXTBAN targeting a specific user account, e.g. to prevent them from joining a channel, make it exempt from bans, or permanently allow it to an invite-only channel.

The `account-notify` spec defines a way for clients to be notified when other clients login to accounts. This spec defines the `ACCOUNT` message to enable this, use of the * WHOX token, as well as outlining the general rectification of account names not being * (as this is used to indicate logged-out).

The `account-tag` spec defines a way for clients to receive a message tag on messages specifying the current account that other client is logged into (or that they aren't logged into one at all). This is especially useful for letting bots make use of the network's authentication and account mechanisms.

The `extended-join` spec defines a way to request that extra client information (including that client's account) is sent when clients join a given channel. This allows better tracking of accounts, particularly when used with `account-notify`.

Away Notifications

The `away-notify` extension provides a way for clients to instantly know when other clients go away or come back. This improves responsiveness and the display of channels for IRC clients that display this information.

The `away-notify` spec describes how to sign up for these notifications and the `AWAY` message to enable this.

Batches

The `batch` extension provides a way for servers to mark certain messages as related. This can simplify the display of this information in clients as well as allow better post-processing on them.

The `batch` spec describes the naming of new batch types, the semantics of batches and how clients should process them.

The **work-in-progress** client initiated batch extension describes client-to-server batches.

Note: Batches themselves are used as a foundation for other extensions and do not themselves offer any user-facing features.

Here are the standalone batch types the IRCv3 WG defines:

- The `netsplit` and `netjoin` batch types allow clients to collapse netsplits and netjoins more effectively.
- The `chathistory` batch type allows replaying message history.

Bot Mode

The Bot mode lets bots mark themselves as such. Other users will see the client as a bot in various ways, and should see a tag on that client's messages if they have requested the `message-tags` capability.

The Bot Mode spec describes the mode and how to see bots that have set the mode on themselves.

Channel renaming

The `channel-rename` extension adds a new command sent by clients and servers that enables renaming a channel without closing it down and redirecting to a new one.

The **work-in-progress** `channel-rename` spec describes how to use the `RENAME` command to achieve this.

Persistence

The **work-in-progress** `chathistory` spec describes the syntax and semantics of the new `CHATHISTORY` command, which standardizes a mechanism for clients to request message history from servers or bouncers.

The **work-in-progress** `message-redaction` spec adds a new command to remove a message from the history, and indicate to other clients they should hide it.

The **work-in-progress** `read-marker` spec adds a new command to synchronize read markers between several clients of the same user.

The **work-in-progress** `pre-away` spec allows clients to send `AWAY` commands during connection registration.

Changing User Properties

Several extensions allow servers to notify clients of properties of other users that were historically communicated exchanging for the lifetime of a connection.

The `chghost` spec describes the new `CHGHOST` message which lets clients more easily see when other clients' usernames and/or hostnames are changed. This replaces the clunky method of sending a fake `QUIT`, and then one or more fake `JOIN` messages instead.

The `setname` spec describes the new `SETNAME` message which allows clients to update their realname (gecos) after connecting to the server, and see updates from other users. This is especially useful as some clients use the realname information for avatars and an additional name field.

Client-Only Tags

Client-only tags are message tags that are sent directly between clients with no server involvement. They're special in IRCv3 as they only apply to clients, and as such we detail them in their own section here.

Here are the client-only tags the IRCv3 WG defines:

- The `channel-context` client-only tag indicates the channel a private message should be displayed in.
- The `reply` client-only tag marks that a given message is intended as a reply to a specific sent message.
- The `react` client-only tag **(draft)** sends a reaction to a specific sent message, allowing such functionality from other chat systems.
- The `typing` client-only tag lets users know when another user is typing a message in their channel or private message.

Echo Message

The `echo-message` extension lets clients confirm when messages are sent, and see messages that other clients on their connection (say, via an IRC bouncer) have sent. It does this by echoing messages back to clients after they are sent, allowing for these extra functionalities.

The `echo-message` spec describes which messages are echo'd, and how they are interpreted by clients.

Invite Notify

The `invite-notify` extension allows privileged channel users to see when someone is invited to their channel. This helps channops better run their channels and see better information about what's going on.

The `invite-notify` spec describes the new `INVITE` reply which this extension uses, and how clients interpret these notifications.

ISUPPORT

The **work-in-progress** `extended-isupport` spec allows clients to fetch the `ISUPPORT` metadata prior to connection registration, introduces a new command to fetch `ISUPPORT` and a new batch type to delimit `ISUPPORT` bursts.

The **work-in-progress** network icon spec introduces a new `ISUPPORT` token to advertise a network icon.

Labeled Responses

The `labeled-response` extension allows clients to link returned numerics with sent commands. This allows for much richer/accurate implementations of `echo-message`, and lets clients generally correlate sent and received messages.

Additionally, this should also assist bouncers with correctly directing responses to the right connected client.

The `labeled-response` spec covers the `label` tag, and how clients should send and will receive labels.

Listing Users

IRCv3 specifications extend traditional queries (`NAMES`, `WHOIS`, `WHO`) to carry more information about other users in a given channel.

The `multi-prefix` spec details changes to these queries, which allow clients to see all the statuses (for example, chanop) that other clients have in a channel rather than just the highest. This improves data tracking for clients and bots, and allows clients to display the privilege level of other clients more correctly.

The `userhost-in-names` spec describes how the `NAMES` message changes with this capability active, and how clients should interpret the changes. This allows clients to more easily see the user/hostnames of other clients when joining channels. This allows clients to better track info and automate client features more easily.

The `WHOX` spec describes how the `WHO` message and its replies changes with this capability active. This allows clients to request more data, and how clients should interpret these changes.

The `no-implicit-names` spec allows clients to disable the implicit `NAMES` responses sent after `JOIN` in case they don't always need that information for all channels. Clients can still query that information as needed via the `NAMES` or `WHO` command.

The **work-in-progress** `oper-tag` spec defines a way for clients to receive a message tag on messages specifying that the source of the message is from an IRC operator. This is useful for letting users know that a message is from a trusted source.

Message IDs

The `message-ids` extension allows servers to provide a network-unique identifier on messages (including `PRIVMSG`/`NOTICE`). This allows clients to build new features that refer to specific messages, with the knowledge that these identifiers will be unique.

The `message-ids` spec covers the `msgid` tag, how servers should generate message IDs and how clients should treat them.

Note: Message IDs themselves are used as a foundation for other extensions and do not themselves offer any user-facing features. Specific IRCv3 extensions will note their use of (and dependency on) message IDs.

Metadata

The **work-in-progress** `metadata-2` specification is a framework to associate information to users. It succeeds the v3.2 `METADATA` command, which was found to have issues related to rate-limiting and excessive notifications, which made it impossible for servers in widespread use to implement.

Monitor

The `MONITOR` command acts as a standardized way for clients to be alerted when other clients enter or exit the network. This is in opposition to `ISON`, which does this through polling, and `WATCH`, which differs between vendor implementations.

The Monitor spec details this command, the relevant `RPL_ISUPPORT` token and the commands used with it.

The Extended Monitor spec builds upon the Monitor spec, and extends it to various events.

Multiline messages

The `multiline` extension adds a new batch type and tag sent by clients and servers to send messages that can exceed the usual byte length limit and that can contain line breaks.

The **work-in-progress** `multiline` spec describes how to use the `draft/multiline` batch type and `draft/multiline-concat` tag to achieve this.

Server Time

The `server-time` extension allows clients to see the exact time that messages were sent and received. This allows bouncers to replay information with more accurate time tracking.

The `server-time` spec describes the `time` tag, how to specify timestamps and how clients should parse incoming timestamps.

Server Name Indication (SNI)

SNI makes it easier for servers to send the correct TLS certificate to connecting clients.

The **work-in-progress** SNI spec provides guidelines for clients and servers, allowing them to better detect the TLS certificate to send based on the server's hostname.

Standard Replies

Standard Replies establishes a clean way to send note, warnings, and errors to users. This helps server/bouncer developers avoid creating new (possibly conflicting) numerics, and ensures clients will be able to display these messages to users nicely.

The standard replies spec describes the format of these messages, how to use them, and guidance on creating useful descriptions for users.

Strict Transport Security (STS)

STS allows clients to be automatically upgraded to use TLS encryption and prevents downgrade attacks. STS is intended as a replacement for the `STARTTLS` command with better security qualities.

The `sts` spec describes the `sts` capability, how it operates, and various implementation details for both clients and servers.

UTF8ONLY

The `UTF8ONLY` ISUPPORT token lets servers indicate that they only support UTF-8 traffic, allowing clients to set their incoming/outgoing encodings automatically.

The `UTF8ONLY` spec details the `RPL_ISUPPORT` token, associated messages, and functionality.

WebIRC

The `WEBIRC` command is widely used to provide the real IP address of users to the server when connecting through a gateway. This is common for current web-based IRC clients.

The `WEBIRC` spec describes how this command works, how to use it, and some best practices to keep in mind while implementing this feature.

WebSocket

The WebSocket spec describes conventions for transporting IRC lines over the WebSocket protocol. This is necessary for browser-based clients, which cannot make conventional TCP connections to IRC servers.

Deprecated Extensions

These extensions have been explicitly **deprecated**. We no longer recommend implementing them. Generally, these extensions have either been superseded, or other major implementation issues have been discovered with them.

STARTTLS

STARTTLS allows clients to upgrade their plaintext connections to use TLS encryption. In alignment with RFC8314, it is recommended that IRC networks use listeners designed for implicit TLS (such as those that operate on port 6697) and clients instead implement SSTS support.

The `tls` spec is still available for reference. It describes how the `STARTTLS` command works,