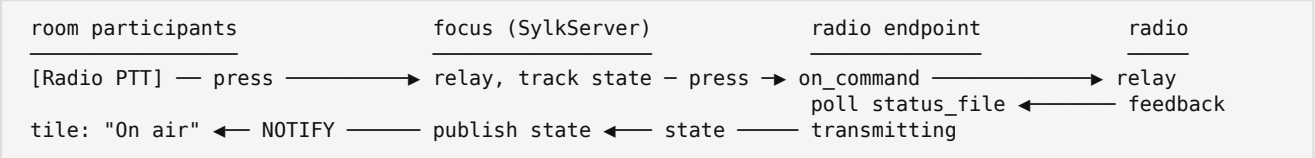
Radio PTT

A conference participant can sit next to a two-way radio and act as a bridge to people who are only reachable over the radio (e.g. someone in a forest with a handheld). Receive works without any help: the radio speaker is next to the participant's microphone, so the room hears the radio. Transmit needs the radio's push-to-talk key pressed while the room speaks.

This document specifies how a participant advertises that it can key a radio, how other participants ask it to, and how the actual on-air state is reported back to everyone.



No standard covers this end to end. BFCP (RFC 8855/8856) and 3GPP MCPTT (TS 24.380) solve a different problem — floor control, taking turns to talk — which is not wanted here: everyone in the room talks freely, and whatever the room says goes out over the radio while it is keyed. We use standard mechanisms for capability advertisement (RFC 3840 feature tag, RFC 4975 accept-wrapped-types) and state distribution (RFC 4575 conference-info), and a private MSRP content type for the commands.

Principles

- **The hardware is the source of truth, when there is one.** With a status file, a state is `transmitting` only when the radio says so (file present), never because a command was sent. Without a status file the endpoint reports what its commands did (see Without a status file).
- **No floor control.** Nobody takes turns and nothing is refused: any participant can press or release any radio at any time. The point is that everyone sees, on each leg that has a radio, whether it is on air.
- **The focus relays and publishes.** Clients send commands to the room, never to the radio endpoint directly; the focus forwards them to the right device and publishes the resulting state in conference-info.
- **Release never depends on signalling.** The relay hardware has its own timer; the endpoint has its own timeout. A lost `release` costs at most `timeout` seconds of air time.
- **Unknown clients are unaffected.** PTT messages are consumed by the focus and delivered only to the endpoint that advertised the capability.

1. Capability advertisement

The radio endpoint signals the capability in two ways; the focus accepts either.

Contact feature tag (RFC 3840), in the INVITE to the room (or the 200 OK when the focus calls the endpoint):

```
Contact: <sip:forest@1.2.3.4:5060;transport=tls>;+g.agp.radio-ptt
```

python3-sipsimple sends it via `Session.connect(..., contact_parameters= {'+g.agp.radio-ptt': None})` (also `Session.accept()`). sip-session3 uses it when the installed sipsimple has that argument.

MSRP accept-wrapped-types (RFC 4975): the PTT content type listed on the chat stream.

```
a=accept-wrapped-types:text/* image/* application/im-iscomposing+xml application/vnd.agp.radio-ptt+json
```

The endpoint has to accept the type anyway to receive commands, so this works with any sipsimple version. Only an explicit listing counts: `*` or `application/*` do not make an endpoint a radio.

The endpoint advertises only when PTT is fully configured locally (see Endpoint settings). The focus checks when the session joins the room and logs which signal it saw. As a last fallback, a state report received from a session (section 2) also marks it as a radio, covering a chat stream added after joining. A chat stream is required in every case: it carries the commands.

The capability belongs to the device (session), not to the AoR: the same account may be in the room from another device that has no radio.

2. Wire protocol

Content type: `application/vnd.agp.radio-ptt+json`, carried in CPIM over the existing MSRP chat stream. One JSON object per message.

Client → focus

The CPIM `To:` is the room identity (as for public chat).

```
{"action": "press",   "id": "<uuid>", "endpoint": "<endpoint entity>"}
{"action": "release", "id": "<uuid>", "endpoint": "<endpoint entity>"}
```

- `id` — generated by the client on `press`, reused on the matching `release`. Correlates everything that follows, including log lines.
- `endpoint` — identifies the radio device. The focus accepts, in order: the `<agp-conf:participant_id>` of the endpoint, the `entity` attribute of the `<endpoint>` element (its Contact URI), or a bare AoR when exactly one radio in the room has it.

Focus → radio endpoint

Sent on the radio endpoint's chat stream with sender = room identity.

```
{"action": "press",   "id": "<uuid>", "by": "ag@sylk.link"}
{"action": "release", "id": "<uuid>", "by": "ag@sylk.link"}
```

The endpoint acts on these only when the CPIM `From:` is the room identity and only on a session to a conference focus; anything else is logged and ignored. (The MSRP layer has already sent its 200 report at that point.) Since the focus never relays this content type between participants, the only legitimate sender is the focus.

Radio endpoint → focus

CPIM `To:` is the room identity.

```
        {"state": "keying",       "id": "<uuid>", "timeout": 30}
        {"state": "transmitting", "id": "<uuid>", "timeout": 30}
        {"state": "idle",         "id": "<uuid>", "reason": "released"}
        {"state": "error",        "id": "<uuid>", "reason": "not_confirmed"}
```

`id` is `null` for transitions not caused by a command (e.g. a local manual press). `reason` values:

reason	meaning
`released`	status file vanished after the endpoint ran `off_command`
`timeout`	the endpoint's own `timeout` fired and it ran `off_command`
`hardware`	status file vanished without the endpoint asking (relay timer)
`not_confirmed`	on_command ran but the status file did not appear in time
`command_failed`	on_command / `off_command` exited non-zero

MSRP reports

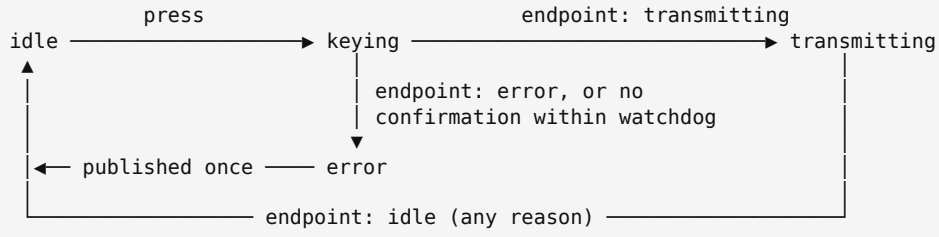
The focus answers each client message with an MSRP report: `200` when accepted, `404` when the endpoint is not in the room or has no radio capability. The resulting state change is distributed via conference-info, not via further MSRP messages to the requester.

3. Focus behaviour

Per radio endpoint the focus keeps:

```
state    idle | keying | transmitting | error
id       id of the last command, or null
by       AoR of whoever last pressed, or "local"
since    monotonic time of the last transition
timeout  as last reported by the endpoint
```

State machine:



Rules:

- `press` and `release` are accepted from any participant in any state and always forwarded. A `press` while already `keying` or `transmitting` extends the transmission (see endpoint behaviour); `by` becomes the latest presser.
- `release` is likewise accepted from anyone, regardless of who pressed.
- Endpoint reports win. If the endpoint reports `transmitting` with `id: null` (local press) while `idle`, the focus moves to `transmitting` with `by: "local"`.

- Watchdog: if the endpoint does not report `transmitting` or `error` within a few seconds of the focus forwarding `press` (default 5 s), the focus moves to `error` with reason `no_response`, then to `idle`.
- `error` is published once and then falls back to `idle`, so tiles can show the failure briefly.
- Radio endpoint leaves the room → state dropped (published as `idle`, reason `left`). If it was transmitting, the hardware timer releases the relay.
- Every transition calls `dispatch_conference_info()` and logs one line:

```
Room forest - radio PTT forest@sip2sip.info: idle -> keying id=1f3c… by=ag@sylk.link
Room forest - radio PTT forest@sip2sip.info: keying -> transmitting id=1f3c… (0.84s)
Room forest - radio PTT forest@sip2sip.info: transmitting -> idle id=1f3c… reason=hardware (30.02s)
```

4. conference-info

On the `<endpoint>` of the radio device, a new element in the `agp-conf` namespace (`urn:ag-projects:xml:ns:conference-info`), defined in `sylk/payloads/conference_info_extensions.py` like the existing extensions:

```
<endpoint entity="sip:forest@1.2.3.4:5060;transport=tls">
  ...
  <agp-conf:radio_ptt state="transmitting" by="ag@sylk.link" elapsed="4" timeout="30"/>
</endpoint>
```

attribute	presence	meaning
`state`	always	`idle`, `keying`, `transmitting`, `error`
`by`	keying / transmitting	AoR of last presser, or `local`
`elapsed`	keying / transmitting	whole seconds since entering the state, computed by the focus
`timeout`	transmitting	endpoint's timeout in seconds
`reason`	error, idle	see reason table; on idle, why the last transmission ended

`elapsed` is computed server-side on every build (same approach as `agp-conf:duration`), so clients with skewed clocks and late joiners render a correct countdown (`timeout - elapsed`) without comparing timestamps.

The presence of the element is the capability flag: an endpoint with `<agp-conf:radio_ptt state="idle"/>` can be keyed. The element is included in both the full and the bridge-stripped (SIP-only) variants of the payload.

5. Endpoint

Endpoint settings

In sipclients3 these are global settings, set with `sip-settings3 -g set radio_ptt.on_command="..."` etc.

setting	default	meaning
radio_ptt.on_command	none	shell command that presses the PTT relay
radio_ptt.off_command	none	optional; shell command that releases it
radio_ptt.status_file	none	optional; path that exists while the radio is transmitting
radio_ptt.timeout	30	seconds after transmitting at which the endpoint runs off_command
radio_ptt.confirm_timeout	2	seconds to wait for status_file after on_command
radio_ptt.poll_interval	0.1	status file poll period, seconds

Setting on_command enables the feature: the capability (feature tag and content type) is advertised whenever it is set. Everything else is optional.

Without off_command the relay is expected to release on its own timer. release and the endpoint timeout then run nothing: without a status file idle is reported at once; with one, idle follows when the file disappears.

Commands run without a shell-visible tty, with a hard execution limit, and receive context in the environment:

```
RADIO_PTT_ID       command id (empty for internal off at startup)
RADIO_PTT_BY       AoR of the presser
RADIO_PTT_TIMEOUT  configured timeout
RADIO_PTT_ROOM     room URI
```

Exit status 0 = success; anything else → error / command_failed.

Behaviour

- **press while on air** (status file present): do not run on_command again. If the endpoint keyed the radio, restart the timeout timer and report transmitting with the new id — a second press extends the transmission. If the radio was keyed by hand (local), only report transmitting; the endpoint does not take over a manual press.
- **press** otherwise: run on_command, report keying, then poll. File appears → report transmitting and start the timeout timer. File absent after confirm_timeout → run off_command, report error/not_confirmed.
- **release**: run off_command. The idle report is sent when the file disappears, with reason released.
- **timeout**: run off_command; idle is reported with reason timeout when the file disappears.
- **File disappears on its own** (hardware timer, manual release on the radio): report idle/hardware.
- **File appears on its own** (someone keys the radio by hand): report transmitting with id: null. The room then sees that the radio is on air and the forest cannot be heard meanwhile.
- **Startup**: if the file exists, run off_command once and log it — clears a relay left keyed by a crash.
- **Session end / chat stream lost** while keying or transmitting: run off_command. The hardware timer remains the backstop.
- The status file is polled continuously while the endpoint is in a conference that it advertised PTT to, not only after a command, so local presses are seen.

Without a status file

When `status_file` is not set there is no hardware feedback, and the command results stand in for it:

- **press**: report `keying`, run `on_command`; exit 0 → report `transmitting` and start the `timeout` timer; non-zero → `error` / `command_failed`. `confirm_timeout` is not used.
- **release** / **timeout**: run `off_command`; exit 0 → report `idle` with reason `released` / `timeout`.
- **release while `on_command` is still running**: report `idle` at once; if `on_command` then succeeds, `off_command` runs again so the relay is not left keyed unobserved.
- **Startup**: `off_command` runs once, unconditionally.
- Not detected: manual presses at the radio, and the relay's own timer releasing before the endpoint `timeout`. Keep `timeout` at or below the relay's timer so the published state does not show `transmitting` after the radio has stopped.

Endpoint log lines, UTC timestamps, same id as the focus:

```
2026-10-20 19:10:02.114 radio PTT press id=1f3c… by=ag@sylk.link -> on_command
2026-10-20 19:10:02.951 radio PTT transmitting id=1f3c…
2026-10-20 19:10:32.966 radio PTT idle id=1f3c… reason=hardware
```

6. Client UI

The reference implementation is Blink for macOS (Blink Cocoa). This section describes what it does so other clients (Blink Qt, Sylk) can match it.

6.1 Which participants have a radio

Parse `<agp-conf:radio_ptt>` on every `<endpoint>` of the conference-info document. Clients must register the element with their conference-info parser (Blink: `RadioPTT.py`, a copy of the definition in `sylk/payloads/conference_info_extensions.py`), otherwise it is silently dropped.

A participant *has a radio* when one of its endpoints carries `<agp-conf:radio_ptt>` **and** that same endpoint has an `audio` media. Endpoints with the element but no audio are ignored: the room cannot talk through them. When a user has several radio endpoints, the most active state wins: `transmitting` > `keying` > `error` > `idle`.

Everything below is re-evaluated on every conference-info NOTIFY and on tab switch.

6.2 Toolbar button

aspect	behaviour
present	only when the session is to a conference focus and at least one participant has a radio (6.1); removed otherwise
position	alone at the right edge of the toolbar, after a flexible space; every other button stays left aligned
label	“Push to Talk” (es “Pulsar para hablar”, nl “Druk om te praten”, pt “Pressione para falar”, ro “Apasă și vorbește”)
image	`icons/radio_ptt.png` (walkie-talkie, transparent background, 90×90)
on air	while any radio is `transmitting`, the image alternates every 0.5 s between `icons/radio_ptt_tx.png` (same picture with a red glow) and the plain one — a pulse
enabled	when the chat (MSRP) stream to the room is connected
tooltip	“Key the radio of conference participants”; “Radio on air, press to extend” while transmitting
click	sends one `press` per radio endpoint (6.1), see 6.4

There is no release button: the endpoint’s timeout and the relay’s own timer end the transmission. A press while on air extends it (section 5).

6.3 Participants list

- A participant with a radio shows the radio picture **instead of** the audio icon (no separate audio icon), drawn larger than the other media icons (28 pt in a 34 pt row) because the picture is tall and narrow.
- While that participant’s radio is `transmitting` the picture pulses with the toolbar (red glow on/off every 0.5 s). `keying`, `idle` and `error` show the plain picture.
- Participants without a radio keep the normal audio / on-hold icon.

6.4 Sending a press

For each radio endpoint, send on the chat stream to the room:

```
Content-Type (inside CPIM): application/vnd.agp.radio-ptt+json
{"action": "press", "id": "<new uuid4>", "endpoint": "<endpoint entity>"}
```

`endpoint` is the `entity` attribute of the `<endpoint>` element, e.g. `sip:24503987@167.108.12.180:41215;transport=tls`. No recipients (the room itself). The resulting state arrives later in conference-info; nothing is updated optimistically. Log one line per press.

6.5 Pulse timer

One timer per chat window, running only while the selected tab’s conference has a `transmitting` radio. Each tick toggles a shared on/off phase, sets the toolbar image and repaints the participants list. Stop it (and reset the phase to “on”) when no radio is transmitting, on tab switch to a session without one, and when the window closes.

6.6 sip-session3

`/ptt [press|release|status] [participant]`; a radio's state is shown in the roster as `[radio <state>]` and printed on every change: `Radio cyel@sip2sip.info: transmitting — speak now by enry01@sip2sip.info (30s left).`

6.7 Other conference chat window changes in Blink Cocoa

Made together with the radio work; listed so the Qt client can match them.

change	rule
window title	"Conference room: " when the session is to a focus: the room contact's display name from the contacts list, else the room address. A contact name equal to the address or its username does not count. One-to-one chats keep "Chat with …".
title placement	title on its own row above the toolbar (macOS expanded toolbar style), so the buttons get the full width
room notices	chat messages from the room carrying the CPIM header `agp:Message-Type: status` (joined/left, welcome) are shown as system messages, not chat bubbles, and are not saved to history or notified
Encryption button	hidden when the session is to a focus; shown for one-to-one chats
Smileys button	removed (the picker sits next to the input field)
Record, History buttons	removed from the chat window toolbar
Participants button	hidden while the participants drawer is open; the user slides the drawer closed and the button comes back
centred connecting spinner	removed (also its "Connecting…" label); the tab shows its own spinner
message category filter	hidden for conference chats (re-checked when the chat stream connects)
participants drawer	single column stretched to the full width (no empty header cell); no presence bar on participant rows

7. Audio notes (deployment, not protocol)

- Transmit audio path: the endpoint's audio output must reach the radio's microphone — acoustically, or better wired into the radio's mic/accessory input.
- The radio is half duplex: while transmitting, the forest cannot be heard. The tile state makes this visible to the whole room.
- While receiving, the radio speaker is picked up by the endpoint's microphone; the endpoint's echo canceller must cope with the room audio played back in the same space.

8. Out of scope / later

- WebRTC participants: the webrtcgateway would need a request to inject `press`/`release` on the participant's chat leg. Display of state works as soon as the gateway relays the new element.
- Automatic keying (server-side VOX): the focus already has per-participant audio levels and could send `press`/`release` itself when anyone other than the radio endpoint is speaking. The protocol needs no change; the focus becomes the requester with `by` set to the room.
- Several radios in one room: each radio endpoint has its own independent state; nothing in the protocol assumes one.

Implementation

piece	where
`<agp-conf:radio_ptt>` element	`sylk/payloads/conference_info_extensions.py` (server), `sipclient/radio_ptt.py` (client copy)
focus relay, state, watchdog	`sylk/applications/conference/radio_ptt.py`, hooked from `room.py`
radio endpoint	`sipclient/radio_ptt.py` (`RadioPTTEndpoint`), wired into `sip-session3`
endpoint settings	`sipclient/configuration/settings.py` (`radio_ptt` group)
Contact parameters	python3-sipsimple `Session.connect/accept(contact_parameters=...)`
requester	`sip-session3`: `/ptt [press|release|status] [participant]`, radio state shown in the roster and on change

Blink Cocoa: `RadioPTT.py`, `ChatController.py` (button state and press), `ChatWindowController.py` (toolbar presence and position, pulse timer, participant radio state), `ContactCell.py` (participant icon), `icons/radio_ptt.png`, `icons/radio_ptt_tx.png`.

Not done yet: Blink Qt and Sylk mobile; webrtcgateway relay for WebRTC participants.