TETRAcontrol HTTP-API

Version 5.4 · Stand 09/2026 · Status 3 IT GmbH

Diese Anleitung beschreibt die HTTP- und WebSocket-Schnittstelle von TETRAcontrol. Sie richtet sich an Entwickler, die eigene Anwendungen an TETRAcontrol anbinden — lesend (Status, Positionen, Gespräche, SDS) wie schreibend (SDS- und Alarmierungsversand, Gerätesteuerung).

Inhalt

1 Grundlagen 4

1.1 Web-Server einrichten 4

1.2 Authentifizierung 4

Ohne Anmeldung erreichbar 4

Antwort ohne gültige Anmeldung 5

1.3 Berechtigungen 5

1.4 Benutzerspezifische Filterung 6

1.5 Zeichenkodierung 6

1.6 Parameterübergabe 6

1.7 Datums-, Farb- und Zahlenformate 6

1.8 Zwischenspeicherung 7

1.9 CORS 7

2 Übersicht der Adressen 8

2.1 Gemeinsame Antwortcodes 8

3 Senden: SDS, Status, Alarmierung, Positionsabfrage 10

3.1 `/api/sds`, `/api/status`, `/api/lip` 10

Parameter 10

Zusätzliche Parameter bei Alarmierung (`typ=195`) 10

Protokoll-IDs (`typ`) 11

Antwort 11

Beispiele 11

3.2 `/api/result/sds` — Senden mit Zustellquittung 11

3.3 `/api/divera247alarm` — Alarmübernahme von divera247 12

4 Abfragen im JSON-Format 13

4.1 `/api/whoami.json` — angemeldeter Benutzer 13

4.2 `/api/issi.json` — Teilnehmer 13

4.3 `/api/statusgps.json` — Teilnehmer mit Status oder Position 13

4.4 `/api/radio.json` — Funkgeräte 13

4.5 `/api/gssi.json` — Gruppen, Favoriten, Ordner und Scanlisten 14

4.6 `/api/statustexte.json` — Statustabelle 14

4.7 Sirenen: `/api/tse.json` und `/api/tsestatus.json` 14

4.8 Meldeempfänger: `/api/tme.json` und `/api/costatus.json` 15
4.9 `/api/getstatustxt` — Zustand der Funkgeräte 15
5 Gerätesteuerung 16
5.1 `/api/control` 16
Antwort 16
Gerätesperren 16
Beispiele 17
6 Daten aktualisieren 18
6.1 `/api/issiupd` — Status und Position eines Teilnehmers setzen 18
7 Dateien und Listen 19
7.1 Listen als GZIP: `/api/issi.zip`, `/api/gssi.zip`, `/api/tse.zip` 19
7.2 Positionen als KML: `/api/loc.kml` und `/api/locupd.kml` 19
7.3 Statische Dateien 19
7.4 `/robots.txt` — Suchmaschinen aussperren 19
`X-Robots-Tag: noindex` auf jeder Antwort 20
8 WebSocket 21
8.1 Verbindung 21
8.2 Parameter der Verbindung 21
8.3 Nachlieferung beim Verbindungsaufbau 21
8.4 Server → Client 22
8.5 Client → Server 22
SDS senden 23
Teilnehmer ändern (`issiedit`) 23
Gerät steuern (`control`) 23
Anhang A — Objektdefinitionen 25
A.1 `WSdev` — Funkgerät (`type: "radio"`) 25
A.2 `WSdevUpd` — Geräteänderung (`type: "devupd"`) 25
A.3 `WSstatus` — Statusmeldung (`type: "status"`) 26
A.4 `WScall` — Gespräch (`type: "call"`) 26
A.5 `WSsds` — SDS (`type: "sds"`) 26
A.6 `WSpos` — Position (`type: "pos"`) 27
A.7 `WScallout` — Alarmierung (`type: "callout"`) 27
A.8 `WSissi` — Teilnehmer (`type: "issi"`) 27
A.9 `WStse` — Sirene (`type: "tse"`) 28
A.10 `WStme` — Meldeempfänger (`type: "tme"`) 28
A.11 Nachschlagetabellen 28
A.12 Objekte zum Senden 28
Anhang B — Beispiele 30
B.1 Funkgerät (`radio`) 30

B.2 Gespräch (`call`) 30
B.3 Status (`status`) 30
B.4 Position (`pos`) 30
B.5 SDS, Standard (`sds`, Protokoll-ID 130) 31
B.6 SDS, Alarmierung (`sds`, Protokoll-ID 195) 31
B.7 Teilnehmer (`issi`) 31
B.8 Vollständige Anbindung im Browser 31
B.9 Senden per HTTP mit auswertbarem Statuscode 32
Anhang C — Hinweise für bestehende Anbindungen 33
C.1 Korrekturen gegenüber früheren Fassungen 33
C.2 Was eine Anbindung beachten sollte 33
C.3 Grenzen 33
Kontakt 34

1 Grundlagen

1.1 Web-Server einrichten

Der Web-Server wird über **Optionen → Netzwerk** aktiviert. Nach der Aktivierung sollte TETRAcontrol einmal neu gestartet werden.

Zusätzlich zum Klartext-HTTP-Server kann ein HTTPS-Server (TLS) betrieben werden. Dazu ist der Haken bei *SSL* zu setzen und ein Port einzutragen. Beide Server beantworten dieselben Adressen; wo sie sich unterscheiden, ist das im jeweiligen Abschnitt vermerkt.

TETRAcontrol liefert ein selbstsigniertes Zertifikat mit, das in aktuellen Browsern zu Warnungen führt. Ein eigenes Zertifikat wird über `Config.txt` eingebunden:

```
SSLServerport=2244
SSLServerCertFile=Dateiname.pfx
SSLServerCertPW=PASSWORT
```

Das Zertifikat muss im PKCS#12-Format vorliegen (Endung `.pfx` oder `.p12`). Ist die Datei nicht passwortgeschützt, bleibt `SSLServerCertPW=` leer stehen — die Zeile selbst ist trotzdem anzugeben.

> **Hinweis für Browser-Anwendungen:** Eine Installation als Web-App („Zum Startbildschirm hinzufügen“, Vollbild ohne Adressleiste) verlangt einen fehlerfreien sicheren Kontext. Das mitgelieferte Zertifikat genügt dafür nicht: Es trägt keinen SAN-Eintrag und einen vom Hostnamen abweichenden Namen.

1.2 Authentifizierung

Die Anmeldung gegenüber dem Web-Server ist auf fünf Wegen möglich. Sie werden in dieser Reihenfolge ausgewertet; schlägt einer fehl, wird der nächste versucht:

#	Weg	Form
1	Web-Schlüssel als Parameter	`?userkey=<Schlüssel>`
2	Benutzername/Passwort als Parameter	`?user=<Name>&pw=<Passwort>`
3	Web-Schlüssel als Cookie	`Cookie: UserKey=<Schlüssel>`
4	HTTP Basic Authentication	`Authorization: Basic <base64>`
5	Bearer-Token (entspricht dem Web-Schlüssel)	`Authorization: Bearer <Schlüssel>`

> **Serverstände vor dem 20.09.2026 werten `Authorization: Bearer` nur auf dem Klartext-Server aus**; über TLS führte er dort zu 401. Wer ältere Stände bedienen muss, weicht auf das Cookie oder Basic aus.

Jede erfolgreiche Anmeldung setzt das Cookie

```
Set-Cookie: UserKey=<60 Zeichen>; Path=/; HttpOnly; SameSite=Lax; Max-Age=2592000
```

so dass Folgeaufrufe ohne Zugangsdaten in der URL auskommen.

Zugangsdaten gehören nicht in die URL: Parameter landen im Klartext in `Logs\HTTP_<Datum>.log` und in Zwischenprotokollen. Für einen Browser ist das Cookie der richtige Weg, für eine eigene Anwendung `Authorization: Bearer` — beides hält den Schlüssel aus der Adresszeile.

Für WebSocket-Verbindungen aus dem Browser gibt es keine Alternative zum Cookie — ein WebSocket-Handshake kann keinen `Authorization`-Kopf setzen. Ein vorheriger Seitenaufruf muss das Cookie also gesetzt haben.

Ohne Anmeldung erreichbar

Pfad	Klartext-Server	TLS-Server

`login.htm`, `login.now`, `logout.now`	ja	ja
`favicon.ico`	ja	ja
`robots.txt`	ja	ja
`manifest.json` (auch `/ui/manifest.json`)	ja	ja
alle `*.png` außer den Kartenkacheln	ja	ja
`*.css`	ja	ja
`*.js`	**ja**	nein

Antwort ohne gültige Anmeldung

Der Server unterscheidet anhand des `Accept`-Kopfes:

- enthält er `text/html` (Seitenaufruf eines Browsers) → **302** auf `/login.htm`
- sonst (API-Client, WebSocket) → **401** mit `WWW-Authenticate: Basic realm="TETRAcontrol"`

Auf dieses Verhalten stützen sich vorhandene Integrationen; ein API-Client sollte `Accept: application/json` senden, um verlässlich den 401 zu erhalten.

Abmelden: `GET /logout.now` verwirft das Cookie und leitet auf die Anmeldeseite.

1.3 Berechtigungen

Benutzer und ihre Rechte werden unter **Optionen → Netzwerk → Benutzer** verwaltet. `AccessLevel` ist eine Bitmaske:

Bit	Wert	Name	Bedeutung
0	1	`ReadCallSDS`	Gespräche, SDS, Status und Positionen sehen
1	2	`ReadDevice`	Gerätedaten sehen (Seriennummer, OPTA, Firmware)
2	4	`WriteTagebuch`	Bemerkungen zu Gesprächen und SDS schreiben
3	8	`SendSDS`	SDS, Status und LIP-Anforderungen senden
4	16	`SendRC`	SDS-Fernsteuerung und Aktionen auslösen
5	32	`SelectGSSI`	Gesprächsgruppen und TMO/DMO umschalten
6	64	`EditISSI`	Teilnehmerliste ändern
7	128	`EditConfig`	Programmeinstellungen, Lautstärke, Sprechtaste
8	256	`StatusClientFull`	Statusclient-Vollzugriff
9	512	`AudioStream`	Audioübertragung

Die eigene Maske liefert `/api/whoami.json` (Abschnitt 4.1). Geprüft wird stets mit UND-Verknüpfung, ein Benutzer mit `AccessLevel=3` hat also `ReadCallSDS` und `ReadDevice`.

Zusätzlich zu den Benutzerrechten wirken **Gerätesperren** (`Ger<N>SPERREN` in `Config.txt`). Sie unterbinden einzelne Eingriffe an einem bestimmten Funkgerät unabhängig davon, welche Rechte der Benutzer hat — siehe Abschnitt 5.

1.4 Benutzerspezifische Filterung

Ab der Lizenzstufe „Leitstelle PLUS“ lässt sich je HTTP-Benutzer festlegen, welche Status-, Positions- und SDS-Daten er erhält. Dazu wird im TETRAcontrol-Verzeichnis ein Unterordner `Filter` angelegt und dort je Filter eine einfache Textdatei abgelegt:

- **Status- und LIP-Filter:** eine ISSI je Zeile. Nur Status bzw. Positionen dieser ISSIs werden übertragen.
- **Textfilter:** ein Textstück je Zeile. Nur SDS und Alarmierungen, die eines dieser Textstücke enthalten, werden übertragen.

Ist bei einem Benutzer kein Filter hinterlegt, erhält er alle Daten. So kann ein Benutzer beispielsweise alle Statusmeldungen bekommen, aber nur ausgewählte Positionen.

Die Filter wirken auf den WebSocket `/live.json` und auf `/api/statusgps.json`.

1.5 Zeichenkodierung

Dieser Punkt ist die häufigste Fehlerquelle bei eigenen Anbindungen:

Kanal	Kodierung	Deklaration
REST-Antworten (JSON, Text, HTML)	**Windows-1252**	`charset=iso-8859-1`
WebSocket-Textframe	**UTF-8**	—
Statische Dateien	byte-genau, Erkennung je Datei	`charset=utf-8` oder `iso-8859-1`

Die REST-Antworten sind also **nicht** UTF-8. `Response.json()` im Browser und die meisten HTTP-Bibliotheken dekodieren jedoch nach UTF-8, wodurch Umlaute in Teilnehmer- und Statusnamen zerfallen. Richtig ist, die Antwort als Bytes zu lesen und ausdrücklich mit Windows-1252 zu dekodieren:

```
const roh  = await (await fetch(url)).arrayBuffer()
const text = new TextDecoder('windows-1252').decode(roh)
const daten = JSON.parse(text)
```

Die Pakete des WebSockets sind davon nicht betroffen und werden normal verarbeitet.

Beim Senden gilt: Ein `POST` mit `Content-Type: …; charset=utf-8` wird serverseitig nach Windows-1252 umgewandelt. Ohne diese Angabe wird der Rumpf als Windows-1252 gelesen.

1.6 Parameterübergabe

Parameter können als Query-Zeichenfolge oder im Rumpf eines `POST` (`application/x-www-form-urlencoded`) übergeben werden. Werte sind URL-kodiert anzugeben.

Bei `POST` mit zusätzlicher Query-Zeichenfolge in der URL werden beide ausgewertet; gleichnamige Parameter aus der Query haben Vorrang.

> **Mindestens zwei Parameter.** Die Endpunkte `/api/sds`, `/api/status`, `/api/lip`, `/api/result/*`, `/api/issiupd`, `/api/s2s` und `/NBX` werten ihre Parameter nur aus, wenn die Parameterzeichenfolge mindestens ein `&` enthält. Ein Aufruf mit nur einem Parameter bleibt **stillschweigend wirkungslos**. In der Praxis fällt das selten auf, weil `userkey` ohnehin mitgegeben wird; wer über Cookie anmeldet, muss darauf achten. Notfalls genügt ein unbenutzter Zusatzparameter.
>
> **Zeilenumbrüche in SDS-Texten** werden als `%0A` übergeben. Bei der SDS-Fernsteuerung (Protokoll-ID 224) müssen AT-Befehle mit einem Carriage Return abgeschlossen werden, also `%0D` am Ende.

1.7 Datums-, Farb- und Zahlenformate

Zeitstempel erscheinen im JSON als `"\/Date(1499155190641)\/"` — Millisekunden seit dem 1.1.1970 UTC. Ein nicht gesetzter Zeitstempel trägt den Vorgabewert **1.1.2000 00:00 Uhr**; alle Werte mit Jahr 2000 sind daher als „unbekannt“ zu lesen.

Farben werden von den Konfigurationsendpunkten als vorzeichenbehafteter ARGB-Int32 in einem Objekt geliefert:

```
"BackColor": { "Color": -16711936 }
```

`-16711936` entspricht `0xFF00FF00`, also volldeckendes Grün. Umrechnung:

```
const u = wert >>> 0
const a = (u >>> 24) & 0xff, r = (u >>> 16) & 0xff
const g = (u >>>  8) & 0xff, b =  u         & 0xff
// a === 0 bedeutet "nicht gesetzt", nicht "transparent"
```

Die Farben in `WScallout` und `WSdev` sind dagegen bereits Zeichenketten der Form `#RRGGBB`.

Statuscodes gibt es in zwei Ausprägungen: der TETRA-Status als Hexadezimalwert (`statusCode`, z. B. `8003`) und die FMS-Entsprechung als einzelnes Zeichen (`status`, `0`–`9` und `A`–`F`). Nicht jeder TETRA-Status hat eine FMS-Entsprechung. Die verbindliche Zuordnung samt Klartext und Farben liefert `/api/statustexte.json`.

1.8 Zwischenspeicherung

Die Antworten tragen `ETag` und `Cache-Control`. Ein Client sollte den ETag speichern und bei Folgeanfragen als `If-None-Match` mitsenden — die Konfigurationsendpunkte beantworten das mit **304** und leerem Rumpf.

Gruppe	Cache-Control
Konfiguration (Statustexte, Gruppen, Symbole, Kartenquellen)	`no-cache, private` + 304
Betriebsdaten (`issi`, `radio`, `statusgps`, KML)	`max-age=120, private`
Sirenen und Meldeempfänger (`tse`, `tme`, `costatus`)	`max-age=30, private`
`whoami.json`	`no-cache, private`
Sprachaufzeichnungen	`no-store`
Sendebefehle und Steuerung	`private, max-age=0, no-cache`

> Der HTTP-Handler beantwortet Anfragen auf dem Oberflächen-Thread von TETRAcontrol. Umfangreiche Abfragen und viele gleichzeitige Verbindungen belasten damit die Bedienoberfläche des Servers. Für Daueranbindungen ist der WebSocket vorzuziehen; Sammelabfragen sollten sparsam und mit engem Zeitfenster erfolgen.

1.9 CORS

Anfragen mit `Origin`-Kopf werden nur beantwortet, wenn die Herkunft freigegeben ist: entweder sie stimmt mit dem eigenen Host überein, oder sie steht in `CORSORIGINS` in der `Config.txt` (mehrere Einträge durch Komma getrennt).

```
CORSORIGINS=https://leitstelle.example.de,https://dashboard.example.de
```

Freigegebene Antworten tragen `Access-Control-Allow-Origin`, `Access-Control-Allow-Credentials: true` und `Vary: Origin`. Ein `OPTIONS`-Vorabflug wird mit 200 beantwortet.

2 Übersicht der Adressen

Pfade sind ohne Beachtung der Groß-/Kleinschreibung anzugeben; `/API/SDS` und `/api/sds` sind gleichwertig.

Adresse	Zweck	Erforderliches Recht	Abschnitt
`GET /login.now`, `GET /logout.now`	An- und Abmelden	—	1.2
`GET|POST /api/sds`	SDS oder Alarmierung senden	`SendSDS`	3.1
`GET|POST /api/status`	Status senden	`SendSDS`	3.1
`GET|POST /api/lip`	Positionsabfrage senden	`SendSDS`	3.1
`GET|POST /api/result/sds`	Senden mit Zustellquittung	`SendSDS` + Lizenz PRO	3.2
`GET|POST /api/divera247alarm`	Alarm von divera247 übernehmen	`SendSDS`	3.3
`GET|POST /api/control`	Gerät steuern	je Befehl	5.1
`GET|POST /api/issiupd`	Status/Position eines Teilnehmers setzen	`EditISSI`	6.1
`GET /api/whoami.json`	Angemeldeter Benutzer und Rechte	`AccessLevel > 0`	4.1
`GET /api/issi.json`	Teilnehmerliste	`AccessLevel > 0`	4.2
`GET /api/statusgps.json`	Teilnehmer mit Status und Position	`AccessLevel > 0`	4.3
`GET /api/radio.json`	Funkgeräte	`ReadDevice`	4.4
`GET /api/gssi.json`	Gruppen, Favoriten, Ordner, Scanlisten	`AccessLevel > 0`	4.5
`GET /api/statustexte.json`	Statustabelle mit Klartext und Farben	`AccessLevel > 0`	4.6
`GET /api/tse.json`	Sirenen	`AccessLevel > 0`	4.7
`GET /api/tsestatus.json`	Sirenen-Statustabelle	`AccessLevel > 0`	4.7
`GET /api/tme.json`	Meldeempfänger	`AccessLevel > 0`	4.8
`GET /api/costatus.json`	Alarmierungs- und Verfügbarkeitstabelle	`AccessLevel > 0`	4.8
`GET /api/getstatustxt`	Zustand der Funkgeräte (Text)	`AccessLevel > 0`	4.9
`GET /api/loc.kml`	Positionen als KML	`AccessLevel > 0`	7.2
`GET /api/locupd.kml`	KML-NetworkLink	`AccessLevel > 0`	7.2
`GET /api/issi.zip`	Teilnehmerliste als GZIP	`AccessLevel > 0`	7.1
`GET /api/gssi.zip`	Gruppenliste als GZIP	`AccessLevel > 0`	7.1
`GET /api/tse.zip`	Sirenenliste als GZIP	`AccessLevel > 0`	7.1
`GET /robots.txt`	Sperre fuer Suchmaschinen	*keine*	7.4
`WS /live.json`	Livedaten und Befehle	`AccessLevel > 0`	8

Alles, was keiner dieser Adressen entspricht, wird als statische Datei aus dem Ordner `html\` ausgeliefert.

2.1 Gemeinsame Antwortcodes

Code	Bedeutung
200	in Ordnung
231 / 232 / 233	SDS / Status / LIP an das Funkgerät übergeben (nur mit `ReqType`, Abschnitt 3.1)
302	Weiterleitung zur Anmeldeseite (Seitenaufruf ohne

	Anmeldung)
304	unverändert, Antwort aus dem Zwischenspeicher gültig
400	ungültiger Parameter
401	nicht angemeldet
403	angemeldet, aber ohne das erforderliche Recht
404	unbekannte Adresse oder Datei
500	Fehler bei der Verarbeitung
503	kein Funkgerät verfügbar oder Funktion nicht lizenziert

3 Senden: SDS, Status, Alarmierung, Positionsabfrage

3.1 `/api/sds`, `/api/status`, `/api/lip`

Sendet eine SDS, einen Status oder eine Positionsabfrage (LIP) über eines der angeschlossenen Funkgeräte.

```
GET  /api/sds?ziel=4672476&text=Testnachricht&typ=130&gerid=1
POST /api/sds
     ziel=4672476&text=Testnachricht&typ=130&gerid=1
```

Erforderliches Recht: **SendSDS** (8).

Die drei Adressen `/api/sds`, `/api/status` und `/api/lip` nehmen denselben Parametersatz entgegen und unterscheiden sich nur darin, was ausgesendet wird. Aus historischen Gründen sind zusätzlich `/sds` und `/status` ohne das Präfix `/api/` erreichbar.

Parameter

Parameter	Bedeutung
`ziel`	Ziel-ISSI oder -GSSI. Mehrere Ziele durch Komma getrennt; an jedes wird einzeln gesendet
`gerid`	Gerätenummer 1–6. Fehlt der Wert oder liegt er außerhalb, wählt TETRAcontrol das erste passende aktive Gerät
`text`	Nachrichtentext (SDS und Alarmierung)
`statushex`	TETRA-Status hexadezimal, z. B. `8003`. Nur `/api/status`, zusammen mit `ziel`
`statusfms`	Status im FMS-Format 1–8, wird intern umgesetzt. Nur `/api/status`
`typ`	Protokoll-ID der SDS, siehe Tabelle unten. Vorgabe 1
`flash`	`1` = Flash-SDS. Gleichbedeutend mit `typ=137`
`encr`	`0`/`1` — Verschlüsselung. Vorgabe aus den Programmeinstellungen
`best`	`1` = Empfangsbestätigung anfordern
`prio`	0–15, Priorität der Aussendung. Vorgabe 0
`modus`	`dmo` = Gerät im DMO-Betrieb bevorzugen (nur bei automatischer Gerätewahl)
`remark`	Bemerkung für das interne Protokoll, wird nicht ausgesendet
`ReqType`	`1` schaltet von der HTML-Antwort auf eine reine Textantwort mit sprechendem Statuscode um — siehe unten

Zusätzliche Parameter bei Alarmierung (`typ=195`)

Parameter	Bedeutung
`coprio`	1–15 Priorität/Severity der Alarmierung. 17 = Verfügbarkeitsabfrage, 18 = Rücknahme
`conum`	1–250 Vorfallnummer
`noreply`	`1` = keine Rückmeldung anfordern
`sub`	Subgruppen, durch Komma getrennt

Die Lizenzstufe wirkt auf `coprio` unterschiedlich — das ist beim Auswerten der Antwort zu beachten:

`coprio`	ohne passende Lizenz
2–15 (setzt **Leitstelle PRO** voraus)	wird **stillschweigend auf 1 gesetzt** und ganz normal

	gesendet; es gibt **keine** Fehlermeldung
16–18 (setzt **Leitstelle PLUS** voraus)	**503** „Nicht Lizenziert“, es wird nicht gesendet
über 18	**500**, unabhängig von der Lizenz

Der Parameter `sub` wird dem Text vorangestellt. Bei den „hessischen“ Subtypen im Format `&xx` können diese auch direkt im Text stehen. Bei Verwendung von `sub` findet **keine Längenprüfung** statt — die aufrufende Anwendung muss den Text gegebenenfalls um die Länge der Subgruppenangabe kürzen. Überschreitet die Nachricht die zulässige SDS-Länge, wird sie automatisch verkettet.

Protokoll-IDs (`typ`)

Wert	Bedeutung
0 / 2	Einfache SDS (Simple Text)
1 / 130	Standard-SDS
9	Simple Flash SDS
10	LIP — Positionsanforderung
137	Flash-SDS
138	Verkettete SDS
195	Alarmierung (CallOut)
204	Home Mode Display
224	SDS-Fernsteuerung (Motorola)

Antwort

Ohne `ReqType` antwortet der Server mit einer HTML-Seite, die den Vorgang im Klartext beschreibt — gedacht für den Aufruf aus einem Browser oder einer Aktionssteuerung heraus. Ist ein `Referer`-Kopf vorhanden, enthält die Seite zusätzlich eine Weiterleitung dorthin.

Mit **`ReqType=1`** liefert der Server stattdessen reinen Text und setzt den Statuscode sprechend:

Code	Bedeutung
231	SDS an das Funkgerät übergeben
232	Status an das Funkgerät übergeben
233	Positionsanforderung an das Funkgerät übergeben
403	Recht SendSDS fehlt
500	Fehler, z. B. unvollständige Parameter
503	kein Gerät verfügbar oder Funktion nicht lizenziert

Für maschinelle Anbindungen ist `ReqType=1` die empfohlene Betriebsart.

> **Wichtig:** Die Antwort bestätigt nur, dass der Auftrag an das Funkgerät übergeben wurde — nicht, dass die Nachricht zugestellt ist. Eine echte Zustellquittung liefert `/api/result/sds` (Abschnitt 3.2).

Beispiele

```
/api/sds?ziel=4672476&text=Bitte%20melden&typ=130&gerid=1&best=1&ReqType=1
/api/sds?ziel=2691121&text=&23Funktionsprobe&typ=195&coprio=3&conum=15
/api/status?ziel=4672476&statushex=8003&gerid=1&ReqType=1
/api/lip?ziel=4672476&gerid=1&ReqType=1
```

3.2 `/api/result/sds` — Senden mit Zustellquittung

Wie Abschnitt 3.1, aber die Antwort kommt erst, wenn das Funkgerät die Zustellung quittiert hat oder die Wartezeit abgelaufen ist.

Erforderliches Recht: **SendSDS** (8), zusätzlich Lizenzstufe **Leitstelle PRO**.

Parameter wie in Abschnitt 3.1. Die Adresse lautet `/api/result/sds`, `/api/result/status` bzw. `/api/result/lip`.

Die Antwort ist eine mit Semikolon getrennte Zeile:

```
<Ergebnis>;<Text>;<Status>;<Status2>;<Sendezeit>;<Quittungszeit>;;<MessageRef>
```

Feld	Bedeutung
Ergebnis	`1` = zugestellt, `0` = nicht zugestellt bzw. Fehler
Text	`OK`, `FEHLER` oder `TIMEOUT`
Status	Quittungsstatus des Funkgeräts
Status2	Quittungsstatus des zweiten Geräts bei doppelter Alarmierung
Sendezeit, Quittungszeit	`yyyyMMddHHmmss.fff`
MessageRef	Referenznummer der Aussendung

Beispiel einer Antwort:

```
1;OK;1;;20260916143012.418;20260916143014.902;;37
```

3.3 `/api/divera247alarm` — Alarmübernahme von divera247

TETRAcontrol kann Alarme von divera247 entgegennehmen und als Alarmierung (CallOut) aussenden.

Erforderliches Recht: **SendSDS** (8), Lizenzstufe mindestens **Leitstelle**.

Der Endpunkt nimmt zwei Formen entgegen:

- **`POST` mit `application/json`** — der von divera247 gesendete JSON-Rumpf
- **`GET`/`POST` mit Formularparametern**

Parameter	Bedeutung
`title`	Stichwort
`text`	Meldungstext
`address`	Einsatzadresse
`priority`	Priorität; wird dem Alarmtext als `S<Wert>*` vorangestellt
`pager[0]` … `pager[50]`	Ziel-ISSIs der Meldeempfänger

Der ausgesendete Alarmtext wird zusammengesetzt aus `S<priority>*<title>*<address>*<text>` und auf die konfigurierte Höchstlänge gekürzt. Gesendet wird über das in den Alarmierungseinstellungen festgelegte Gerät; ist keines konfiguriert, antwortet der Server mit „Kein Gerät Verfügbar“.

Einrichtung in divera247: Verwaltung → Schnittstellen → Request-Service, als Format `POST application/json`, als Inhalt `TETRAcontrol`, als Adresse der HTTPS-Server von TETRAcontrol. Die Anmeldung erfolgt über `userkey`.

4 Abfragen im JSON-Format

Alle Antworten dieses Kapitels sind JSON in Windows-1252 (Abschnitt 1.5) und tragen einen `ETag`.

4.1 `/api/whoami.json` — angemeldeter Benutzer

```
GET /api/whoami.json

{ "Username": "leitstelle", "AccessLevel": 255 }
```

`AccessLevel` ist die Bitmaske aus Abschnitt 1.3. Eine Anwendung sollte diesen Endpunkt einmal beim Start abfragen und Funktionen ausblenden, für die das Recht fehlt, statt sie anzubieten und im Fehler laufen zu lassen.

4.2 `/api/issi.json` — Teilnehmer

```
GET /api/issi.json?filter=DRK
GET /api/issi.json?issi=4672476
GET /api/issi.json?MaxAlter=86400
```

Parameter	Bedeutung
`issi`	genau diesen Teilnehmer liefern
`filter`	Textsuche in ISSI, Name, OPTA und Bemerkung. **Groß-/Kleinschreibung ist relevant**
`MaxAlter`	Höchstalter des letzten Status in Sekunden. Vorgabe 788400000 (25 Jahre)

Antwort: ein Objekt mit dem Feld `issis`, darin ein Feld von `WSissi` (Anhang A):

```
{ "issis": [ { "type": "issi", "ISSI": "4672476", "name": "GN 1-20-1", … } ] }
```

Mit `issi=` wird abweichend **ein einzelnes, nicht eingepacktes Objekt** geliefert. Ist der Teilnehmer unbekannt, ist der Rumpf leer.

> `MaxAlter` filtert gegen den Zeitstempel des letzten Status. Ein Teilnehmer, der noch nie einen Status gemeldet hat, ist über die Liste deshalb **nicht** erreichbar — wohl aber einzeln über `issi=`.

4.3 `/api/statusgps.json` — Teilnehmer mit Status oder Position

```
GET /api/statusgps.json?maxalterstatus=86400&maxalterpos=3600
```

Parameter	Bedeutung	Vorgabe
`maxalterstatus`	Höchstalter des Status in Sekunden	2592000 (30 Tage)
`maxalterpos`	Höchstalter der Position in Sekunden	86400 (1 Tag)

Beide Werte dürfen höchstens neun Stellen haben; längere Angaben werden ignoriert und die Vorgabe verwendet.

Geliefert werden nur Teilnehmer, bei denen Status **oder** Position gesetzt und jünger als das jeweilige Fenster sind. Die benutzerspezifischen Filter aus Abschnitt 1.4 wirken hier.

Antwortformat wie `/api/issi.json`.

4.4 `/api/radio.json` — Funkgeräte

```
GET /api/radio.json?gerid=1
GET /api/radio.json?gerid=1234
```

Erforderliches Recht: **`ReadDevice`** (2). Fehlt es, antwortet der Server mit **403**.

`gerid` ist eine **Ziffernmenge**, keine Zahl: `1234` bedeutet „Geräte 1 bis 4“. Eine einzelne Ziffer liefert ein einzelnes Objekt, mehrere Ziffern ein Feld. Ohne Angabe wird Gerät 1 geliefert. Nicht aktive Geräte entfallen.

Antwort: `WSdev` (Anhang A) mit Hersteller, Modell, Seriennummer, Firmware, ISSI, OPTA, TEI, MNI, Betriebsart, geschalteter Gruppe, Feldstärke, Funkzelle, Lautstärke, Scanzustand, Fähigkeiten und Gerätesperren.

4.5 `/api/gssi.json` — Gruppen, Favoriten, Ordner und Scanlisten

```
GET /api/gssi.json
GET /api/gssi.json?full=1
```

Parameter	Bedeutung
`full`	`1` liefert zusätzlich **alle** Gruppen im Feld `Groups`

Ohne `full` enthält die Antwort nur Top Ten, die drei Favoritenlisten, die Ordnerstruktur und die Scanlisten — je nach Konfiguration wenige Kilobyte. Mit `full=1` kommen alle konfigurierten Gruppen hinzu; das können mehrere hundert Kilobyte sein. Anwendungen sollten die vollständige Liste erst laden, wenn der Anwender tatsächlich sucht.

```
{
  "Full": false, "GroupCount": 1828,
  "TopTen":  [ { "GSSI": "2693315", "Name": "EG1", "TopTen": 5, … } ],
  "Fav1":    [ … ], "Fav2": [ … ], "Fav3": [ … ],
  "Folders": [ { "Id": 12, "Mode": 0, "Name": "HE_MKK", "Parent": "" } ],
  "ScanLists": [ { "Name": "MKK-BG", "PriGSSI": "2693301",
                   "ScanGroups": [ { "GSSI": "2693302", "Name": "BG RD",
                                     "Prio": "2", "PrioKlartext": "Hoch" } ] } ],
  "Groups":  [ ]
}
```

4.6 `/api/statustexte.json` — Statustabelle

```
GET /api/statustexte.json
```

Liefert die vollständige, kundenspezifisch konfigurierte Statustabelle (`StatusTexte.csv`). Damit lassen sich die Statuscodes aus dem WebSocket und aus `issi.json` in Klartext und Farbe übersetzen.

```
[ { "StatusHex": "8003", "StatusDec": 32771,
    "Text": "Status 1 - E-Bereit Funk", "StatusFMS": "1",
    "HomeModeDisplay": "E-Bereit Funk", "AntwortPID": 220,
    "ForeColor": { "Color": -16777216 }, "BackColor": { "Color": -16711936 },
    "Permanent": true, "LIPRequest": 0, "SoundFile": "" } ]
```

Das Feld **`Permanent`** unterscheidet Zustände von Mitteilungen: Nur ein dauerhafter Status (`true`) beschreibt den Zustand eines Fahrzeugs. Flüchtige Status wie „Sprechwunsch“ oder „Quittung“ gehören in eine Ereignisliste, sollten aber die Zustandsanzeige auf Karte oder Statuspanel nicht umfärben.

> **Ausnahme:** Der Notruf `0000` trägt `Permanent: false`, ist aber sehr wohl als Zustand anzuzeigen. Die Windows-Oberfläche behandelt ihn ausdrücklich als Sonderfall; eigene Anwendungen sollten das ebenso tun.

Kommt im Datenstrom ein Status vor, der nicht in der Tabelle steht, ist er als unbekannt darzustellen — die Tabelle ist nicht zwingend vollständig.

4.7 Sirenen: `/api/tse.json` und `/api/tsestatus.json`

```
GET /api/tse.json
GET /api/tsestatus.json
```

`tse.json` liefert alle konfigurierten Sirenen als Feld von `WStse` (Anhang A): Kennung, Name, Standort, Position, letzter Status mit Zeitstempel, letzte Auslösung, Symbolindex und Farben.

`tsestatus.json` liefert die zugehörige Statustabelle (`Sirenen\TSE_Status.csv`) mit Klartext und Farben. Beide Endpunkte werden gemeinsam gebraucht.

Bei Sirenen steckt der Zustand zusätzlich im **Symbolindex**: `100 + Grundsymbol + Ampelstufe`, wobei sich die Ampelstufe als `(IconIdx - 100) % 5` ergibt.

> Eine Sirene ohne Statusmeldung liefert ein leeres `StatusText` und im Feld `TextLetzterStatus` den Platzhalter „Status Unbekannt“. Der Zeitstempel trägt dann den 1.1.2000 (Abschnitt 1.7).

4.8 Meldeempfänger: `/api/tme.json` und `/api/costatus.json`

```
GET /api/tme.json
GET /api/costatus.json
```

`tme.json` liefert die TETRA-Meldeempfänger (`CallOut\TME_ISSI.csv`) als Feld von `WStme` mit Kennung, Namen, Verfügbarkeit und letzter Alarmierungsantwort.

Parameter	Bedeutung

`costatus.json` liefert in **einer** Antwort beide Nachschlagetabellen, die zur Darstellung nötig sind:

```
{ "status": [ { "Status": "1", "Text": "komme", … } ],
  "ta":     [ { "TA": "1", "Text": "verfügbar", … } ] }
```

`status` übersetzt die Alarmierungsantwort (`CO_Status.csv`), `ta` die Verfügbarkeit (`TA.csv`).

> Die Verfügbarkeit der Meldeempfänger wird **nicht** über den WebSocket nachgeführt. Eine Anwendung, die sie anzeigt, muss `tme.json` in Abständen neu abrufen.

4.9 `/api/getstatustxt` — Zustand der Funkgeräte

```
GET /api/getstatustxt
```

Antwortet als reiner Text, nicht als JSON. Die erste Zeile ist das Gesamtergebnis, danach folgt je aktivem Gerät eine Zeile:

```
OK
1;OK;COM3;Wed, 16 Sep 2026 12:04:21 GMT
2;OK;COM4;Wed, 16 Sep 2026 12:04:19 GMT
```

Feld	Bedeutung
1	Gerätenummer
2	`OK` oder `FEHLER`
3	Schnittstelle: COM-Port bzw. Adresse des gekoppelten Servers
4	Zeitpunkt der letzten gültigen Daten (UTC)

Als `FEHLER` gilt ein Gerät, von dem seit über 121 Sekunden keine gültigen Daten kamen (gekoppelte Server: 181 Sekunden) oder dessen Zeitüberwachung angeschlagen hat. Der Endpunkt eignet sich als Prüfpunkt für eine Betriebsüberwachung.

5 Gerätesteuerung

5.1 `/api/control`

Steuert eines der angeschlossenen Funkgeräte.

```
GET /api/control?cmd=tgs&gerid=1&param1=2693324
GET /api/control?cmd=vol&gerid=1&int1=8
GET /api/control?cmd=ptt&gerid=1&int1=1
```

Parameter	Bedeutung
`cmd`	Befehl, siehe Tabelle
`gerid`	Gerätenummer 1–6, Vorgabe 1
`param1`	erster Parameter, Bedeutung je Befehl
`param2`	zweiter Parameter (derzeit ungenutzt)
`int1`	numerischer Parameter, Vorgabe 1

`cmd`	Wirkung	Parameter	Erforderliches Recht
`tgs`	Gesprächsgruppe schalten	`Param1` = GSSI	`Gruppe Schalten` (4)
`tgs`	Scan ein/aus	`Param2` = `1` bzw. `0`	`Scan Schalten` (8)
`tgs`	Scanliste schalten	`Int1` = 1–9, `Param1`/`Param2` leer	`Scan Schalten` (8)
`vol`	Lautstärke setzen	`int1` = Lautstärke	`EditConfig` (128)
`ptt`	Sprechtaste	`int1` = `1` drücken, `0` loslassen	`EditConfig` (128)

Die Höchstlautstärke ist geräteabhängig — bei Motorola 14, bei Sepura 9. Der verbindliche Wert steht als `volumeMax` in `/api/radio.json`. Werte kleiner als 1 werden auf 1 angehoben; **nach oben begrenzt dieser Endpunkt nicht**.

Antwort

`/api/control` antwortet **immer mit 200** und einer HTML-Seite, die das Ergebnis im Klartext nennt:

Text	Bedeutung
`Befehl gesendet`	an das Gerät übergeben
`NICHT AUTHORISIERT!`	erforderliches Recht fehlt
`Kein Gerät verfügbar`	Gerät nicht aktiv, nicht vorhanden oder gekoppelt
`Ungültige Gruppe`	`param1` liegt außerhalb des zulässigen Bereichs
`FEHLER`	unbekannter Befehl

> Anders als bei `/api/sds` wird der Parameter `ReqType` hier **nicht** ausgewertet. Eine maschinelle Anbindung muss den Antworttext auswerten oder den WebSocket-Befehl `control` verwenden (Abschnitt 8.5), der denselben Funktionsumfang und darüber hinaus Scan, Direktruf und Auflegen bietet.

Gerätesperren

Unabhängig vom Benutzerrecht kann ein Eingriff an einem Gerät über `Ger<N>SPERREN` in der `Config.txt` gesperrt sein. Welche Sperren gesetzt sind, verrät das Feld `restrictions` in `/api/radio.json` als Bitmaske:

Bit	Wert	Sperre
0	1	DMO schalten
1	2	TMO schalten

2	4	Gruppe schalten
3	8	Scan schalten
4	16	Standard-SDS
5	32	Alarmierung
6	64	LIP
7	128	SDS-Fernsteuerung
8	256	Status
9	512	Lautstärke
10	1024	Sprechtaste

> **Wichtiger Unterschied zum WebSocket:** `/api/control` wertet die Gerätesperren **nicht** aus und begrenzt die Lautstärke **nicht** nach oben. Beides prüft nur der WebSocket-Befehl `control` (Abschnitt 8.5). Wer sich auf die Sperren verlassen will, muss den WebSocket verwenden.

Beispiele

```
/api/control?cmd=tgs&gerid=1&param1=2493302&userkey=…
/api/control?cmd=ptt&gerid=1&int1=1&userkey=…
```

6 Daten aktualisieren

6.1 `/api/issiupd` — Status und Position eines Teilnehmers setzen

Trägt Status und/oder Position für einen Teilnehmer ein, ohne dass eine Funkmeldung vorliegt. Gedacht für die Anbindung fremder Quellen — etwa einer Einsatzleitsoftware oder eines GPS-Trackers.

```
GET /api/issiupd?issi=1234567&status=8004&lat=50.1234&lon=9.4567
```

Erforderliches Recht: **`EditISSI`** (64).

Parameter	Bedeutung
`issi`	zu aktualisierender Teilnehmer. **Pflicht**, muss numerisch sein (`ziel` wird als Synonym akzeptiert)
`status`	TETRA-Status hexadezimal, z. B. `8004`
`lat`, `lon`	Position in Dezimalgrad WGS84. Dezimaltrenner ist der **Punkt**
`show`	`1` = Aktualisierung in der Ereignisliste anzeigen, `0` = nur übernehmen
`remark`	Bemerkung, wird an die Clients mitgesendet

Mindestens `status` oder `lat` **und** `lon` müssen angegeben sein; beides zugleich ist möglich und erzeugt dann zwei Einträge.

Antwort: HTML mit `OK`, `NO PERMISSION` oder `INVALID ISSI`. Ist ein `Referer`-Kopf vorhanden, enthält die Seite eine Weiterleitung dorthin.

Mit `show=0` wird der Teilnehmer aktualisiert, der Vorgang aber nicht in der Ereignisliste angezeigt; die Meldung geht dann nur an Clients mit dem Recht `EditISSI`. Ob dabei protokolliert wird, steuern die Programmeinstellungen.

```
/api/issiupd?issi=1234567&status=8004&lat=50.1234&lon=9.4567&userkey=…
```

7 Dateien und Listen

7.1 Listen als GZIP: `/api/issi.zip`, `/api/gssi.zip`, `/api/tse.zip`

```
GET /api/issi.zip
GET /api/gssi.zip
GET /api/tse.zip
```

Liefern die Konfigurationsdateien `ISSI.csv` (Teilnehmer), `GSSI.csv` (Gruppen) und `Sirenen\TSE_ISSI.csv` (Sirenen) **gzip-komprimiert** mit `Content-Type: application/octet-stream`. Trotz der Endung `.zip` handelt es sich um einen GZIP-Strom, nicht um ein ZIP-Archiv.

Bei `/api/issi.zip` wird die Teilnehmerliste zuvor sortiert auf die Platte geschrieben; der Abruf ist also nicht folgenlos und sollte nicht getaktet erfolgen.

> In früheren Fassungen dieser Anleitung war die Adresse als `/API/ISSP.zip` angegeben. Richtig ist `/api/issi.zip`.

7.2 Positionen als KML: `/api/loc.kml` und `/api/locupd.kml`

```
GET /api/loc.kml
GET /api/locupd.kml
```

`loc.kml` liefert die aktuellen Positionen als KML-Dokument zur Anzeige in Google Earth und vergleichbaren Anwendungen.

`locupd.kml` liefert ein KML mit einem `NetworkLink` auf `loc.kml`, so dass sich die Anzeige selbsttätig auffrischt. Die Zieladresse wird aus dem `Host`-Kopf der Anfrage gebildet — der Aufruf muss also unter der Adresse erfolgen, unter der der Server auch für die Kartenanwendung erreichbar ist.

Beide Antworten tragen einen ETag und beantworten `If-None-Match` mit **304**.

7.3 Statische Dateien

Jede Adresse, die keinem der beschriebenen Endpunkte entspricht, wird als Datei aus dem Ordner `html\` ausgeliefert. Endet der Pfad auf `/`, ergänzt der Server `index.html`.

Die Auslieferung erfolgt byte-genau mit passendem MIME-Typ; die Zeichenkodierung wird je Datei erkannt (UTF-8 oder Windows-1252). `.html` wird mit `no-cache` ausgeliefert und über den ETag revalidiert, alle übrigen Dateien mit `max-age=7776000, private`.

> Der TLS-Server ersetzt in ausgelieferten `.js`-Dateien die Zeichenfolge `ws:/` durch `wss:/`. Eigene Skripte sollten das Protokoll zur Laufzeit aus `location.protocol` ableiten und die Zeichenfolge `ws:/` nicht enthalten.

7.4 `/robots.txt` — Suchmaschinen aussperren

```
GET /robots.txt
```

Beide Server beantworten diese Adresse **ohne Anmeldung**. Es gibt zwei Quellen, und die erste gewinnt:

1. **Eine eigene Datei `robots.txt` im Ordner `html`.** Wer sie dort ablegt, bestimmt den Inhalt selbst; sie wird wie jede statische Datei ausgeliefert, mit eigenem `ETag` und `Cache-Control: max-age=7776000, private`.
2. **Fehlt die Datei**, antwortet der Server selbst mit einer Sperre für alle Crawler — `Content-Type: text/plain`, `Cache-Control: max-age=86400, private`:

```
User-agent: *
Disallow: /
```

Die Adresse wird ohne Beachtung der Groß-/Kleinschreibung erkannt, eine Query-Zeichenfolge wird ignoriert. Belegt ist nur der Pfad im Wurzelverzeichnis; `/ui/robots.txt` wird nicht gesondert behandelt, denn ein Crawler fragt robots.txt laut RFC 9309 ausschließlich dort ab.

Ohne Anmeldung gilt für beide Quellen — auch eine selbst abgelegte Datei wird ohne Zugangsdaten ausgeliefert. Das ist Absicht: Ein Crawler meldet sich nicht an und bekäme sonst **401** statt der Sperre.

`X-Robots-Tag: noindex` auf jeder Antwort

Unabhängig davon trägt **jede** HTTP-Antwort beider Server den Kopf

```
X-Robots-Tag: noindex
```

Das ergänzt die robots.txt um das, was sie nicht leisten kann: Sie regelt das **Abholen**, der Kopf das **Aufnehmen in den Index**. Eine Seite, die eine Suchmaschine trotz robots.txt erreicht — etwa über einen Verweis von außerhalb — wird damit nicht in den Index übernommen.

Der Kopf steht auf allen Antworten, auch auf 401, 302 und 404 sowie auf der Anmeldeseite. Nicht betroffen ist der WebSocket-Wechsel (101), der über einen eigenen Weg läuft.

> Beides zusammen ist eine **Bitte an gutwillige Suchmaschinen**, keine Zugangssperre. Der eigentliche Schutz bleibt die Anmeldung.

8 WebSocket

8.1 Verbindung

```
ws://<host>:<port>/live.json?MaxAlter=3600&MaxAnz=200
wss://<host>:<port>/live.json?MaxAlter=3600&MaxAnz=200
```

Der Upgrade wird nur abgeschlossen, wenn die Anfrage bereits authentifiziert ist (`AccessLevel > 0`). Browser können beim WebSocket-Handshake keinen `Authorization`-Kopf setzen — dort läuft die Anmeldung zwingend über das `UserKey`-Cookie aus einem vorherigen Seitenaufruf. Andernfalls antwortet der Server mit **401** und trennt.

Eigene Anwendungen außerhalb des Browsers senden im Handshake:

```
GET /live.json?MaxAlter=3600&MaxAnz=200 HTTP/1.1
Host: leitstelle.example.de
Cookie: UserKey=<Schlüssel>
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: <16 zufällige Byte, base64>
Connection: keep-alive, Upgrade
Upgrade: websocket
```

> **Die `Host`-Kopfzeile ist Pflicht** (RFC 7230). Der Server selbst wertet sie nicht aus, ein vorgelagerter Reverse-Proxy oder Tunnel weist die Anfrage ohne sie aber mit **400** ab, bevor sie TETRAcontrol erreicht.
>
> Die Prüfung der Antwort auf `Sec-WebSocket-Accept` muss **ohne Beachtung der Groß-/Kleinschreibung** erfolgen: Kopfzeilennamen sind laut RFC 7230 nicht schreibungsabhängig, und Proxys normalisieren sie (`Sec-Websocket-Accept`).

Der Server beantwortet Ping-Pakete mit einem Pong. Eine Verbindung, über die länger als 185 Sekunden keine Daten laufen, wird getrennt; ein Keepalive-Ping in kürzerem Abstand hält sie offen.

8.2 Parameter der Verbindung

Parameter	Bedeutung	Vorgabe
`MaxAlter`	Höchstalter der beim Verbindungsaufbau nachgelieferten Daten, in Sekunden	3600
`MaxAnz`	Höchstzahl nachgelieferter Ereignisse (Status, SDS, Gespräche)	1000
`MaxAnzGPS`	Höchstzahl nachgelieferter Positionsmeldungen	Wert von `MaxAnz`
`dev`	Ziffernmenge der Geräte, deren Daten übertragen werden; `13` = Geräte 1 und 3	alle
`userkey`	Web-Schlüssel, falls kein Cookie gesetzt werden kann	—

`MaxAnz` wird auf 2000 begrenzt, `MaxAnzGPS` auf 1000; zusätzlich greifen die serverseitigen Obergrenzen aus der `Config.txt` (`HTTPMAXANZ`, `HTTPMAXANZGPS`, Vorgaben 1000 und 500).

Das getrennte Kontingent für Positionen ist wichtig: Positionsmeldungen sind die mit Abstand häufigste Datenart. Ohne eigenes Kontingent verdrängen sie in der Nachlieferung genau die Status- und SDS-Meldungen, auf die es ankommt. Ein sinnvoller Einstieg ist `MaxAnz=200&MaxAnzGPS=100`.

8.3 Nachlieferung beim Verbindungsaufbau

Unmittelbar nach dem Upgrade sendet der Server unaufgefordert:

1. je aktivem Funkgerät ein `radio`-Paket,

2. die Historie aus dem Ereignisspeicher, neueste zuerst, im Paket der Kontingente.

Die Historie wird nach Art abgebildet:

interne Art	Pakete
Gespräch	`call`
SDS mit Protokoll-ID 10	`pos`
sonstige SDS	`sds`
Status	`status`

Anschließend laufen laufende Ereignisse als einzelne Pakete ein.

8.4 Server → Client

Jeder Textframe ist ein JSON-Objekt mit dem Feld `type`. Die vollständigen Felddefinitionen stehen in Anhang A.

type	Inhalt
`radio`	Stammdaten eines Funkgeräts (`WSdev`)
`devupd`	Änderung an einem Funkgerät: Gruppe, Feldstärke, Lautstärke, Scan, Ruf (`WSdevUpd`)
`status`	Statusmeldung (`WSstatus`)
`call`	Gespräch, Beginn und Ende (`WScall`)
`sds`	SDS, Alarmierung, Rückmeldung (`WSsds`)
`pos`	Positionsmeldung (`WSpos`)
`callout`	Alarmierung mit Rückmeldezählern (`WScallout`)
`issi`	geänderter Teilnehmerdatensatz (`WSissi`)
`tse`	Sirenenmeldung (`WStse`)

Gespräche werden **zweimal** gesendet: beim Beginn ohne `tsEnd`, beim Ende mit. Beide Pakete tragen dieselbe `ID` und sind darüber zusammenzuführen. Dasselbe gilt für SDS, die später eine Empfangsbestätigung erhalten.

Unbekannte `type`-Werte sind stillschweigend zu verwerfen, nicht als Fehler zu behandeln: andere Betriebsarten des Servers kennen weitere Arten, und künftige Versionen können neue hinzufügen.

Binärpakete (Opcode 2) enthalten zlib-komprimiertes Roh-Audio. Eine Anwendung, die keine Sprache verarbeitet, ignoriert sie.

8.5 Client → Server

Der Client sendet Textpakete mit demselben Aufbau. Der Server erkennt den Befehl am Feld `type`; sowohl ein einzelnes Objekt als auch ein Feld von Objekten wird angenommen.

type	Wirkung	Erforderliches Recht
`sds`	SDS, Alarmierung oder LIP-Anforderung senden (`WSsds`)	`SendSDS`
`sendsds`	wie `sds`, mit mehr Feldern (`WSsendsds`)	`SendSDS`
`sendstatus`	Status senden (`WSsendstatus`)	`SendSDS`
`control`	Gerät steuern (`WScmd`), siehe unten	je Befehl
`issiedit`	einzelne Felder eines Teilnehmers ändern (`WSissiEdit`)	`EditISSI`
`call`	Bemerkung zu einem Gespräch schreiben	`WriteTagebuch`
`status`	Bemerkung zu einer Statusmeldung	`WriteTagebuch`

	schreiben	
pos	Bemerkung zu einer Positionsmeldung schreiben	WriteTagebuch

SDS senden

```
{ "type": "sds", "destSSI": "4672476", "text": "Testnachricht",
  "ProtocolID": 130, "radioID": 1 }
```

Für eine Positionsabfrage ist `ProtocolID` auf `10` und `text` auf `LOCATION REPORT REQUEST` zu setzen.

Die reichere Form `sendsds` unterstützt zusätzlich `encrypt`, `receipt`, `prio`, `COprio`, `COvorfall`, `COmin`, `COnoreply` und `PrioQueue` — sie ist für Alarmierungen vorzuziehen.

> Der Server sendet **keine Quittung** über den WebSocket. Dass der Auftrag angenommen wurde, zeigt sich erst daran, dass die Nachricht als eigenes `sds`-Paket zurückkommt. Ausgehende SDS trägt der Server dabei mit dem Präfix `Gesendet:` im Text ein — beim Abgleich ist es zu ignorieren.
>
> Fehlt das Recht `SendSDS`, bleibt der Befehl **wirkungslos und unbeantwortet**. Eine Anwendung sollte das Recht vorab über `/api/whoami.json` prüfen.

Teilnehmer ändern (`issiedit`)

```
{ "type": "issiedit", "ISSI": "4672476", "Bemerkung": "neuer Text" }
```

Es werden **ausschließlich die übermittelten Felder** geändert; nicht gesendete Felder bleiben unangetastet. Änderbar sind `Name`, `Bemerkung`, `OPTA`, `Favorites`, `IconIdx`, `Status` und `StatusTyp`.

Ein leeres Feld gilt als „nicht gesendet“. Ein einmal gesetzter manueller Status lässt sich darüber also **nicht** wieder entfernen.

Nach der Änderung sendet der Server den vollständigen Datensatz als `issi`-Paket an alle Clients.

Gerät steuern (`control`)

```
{ "type": "control", "cmd": "tgs", "Index": 1, "Param1": "2693324",
  "Param2": "", "Int1": 0 }
```

`Index` ist die Gerätenummer 1–6.

cmd	Wirkung	Parameter	Erforderliches Recht
`tgs`	Gruppe schalten	`Param1` = GSSI	`SelectGSSI` (32)
`tgs`	Scan ein/aus	`Param2` = `1` bzw. `0`	`SelectGSSI` (32)
`tgs`	Scanliste schalten	`Int1` = 1–9, `Param1`/`Param2` leer	`SelectGSSI` (32)
`vol`	Lautstärke	`Int1` = Wert, wird auf `volumeMax` begrenzt	`EditConfig` (128)
`ptt`	Sprechtaste	`Int1` = `1`/`0`	`EditConfig` (128)
`direct`	Direktruf aufbauen	`Param1` = Ziel-ISSI	`EditConfig` (128)
`hangup`	Ruf beenden	—	`EditConfig` (128)
`calls`	Gesprächshistorie anfordern	`Param1`, `Param2` = Zeitraum	`ReadCallSDS` (1)
`wav`	Aufzeichnung als Binärpaket anfordern	`Param1` = Gesprächs-ID	`ReadCallSDS` (1)

Jeder `tgs`-Befehl setzt die anstehende Scangruppe zurück. Wer nur die Gruppe wechseln will, sendet `Param2` leer und `Int1` auf `0`.

Ist das angesprochene Gerät nicht lokal, sondern über eine Kopplung angebunden, reicht TETRAcontrol den Befehl an den gekoppelten Server weiter.

Die Rückmeldung eines Befehls kommt als `devupd`-Paket, sobald das Gerät den neuen Zustand meldet — **bei Lautstärke und Gruppenwechsel können dazwischen bis zu 20 Sekunden liegen**. Eine Bedienoberfläche sollte das anzeigen, statt den Wert sofort als übernommen darzustellen.

Anhang A — Objektdefinitionen

Alle Objekte werden als JSON übertragen. Zeitstempel im Format `"\/Date(ms)\/"` (Abschnitt 1.7), Farben je nach Objekt als `#RRGGBB` oder als ARGB-Int32.

A.1 `WSdev` — Funkgerät (`type: "radio"`)

Feld	Typ	Bedeutung
`radioID`	Zahl	Gerätenummer 1–6
`radioName`	Text	Name des Geräts, z. B. `F4`
`active`	Zahl	1 = aktiv
`vendor`, `model`, `serial`, `swVersion`	Text	Hersteller, Modell, Seriennummer, Firmware
`ISSI`, `OPTA`, `TEI`, `MNI`	Text	Kennungen des Geräts
`mode`	Zahl	0 = TMO, 1 = DMO, -1 = unbekannt
`GSSI`, `GSSIname`	Text	geschaltete Gruppe, Kennung und Klartext
`RSSI`	Zahl	Feldstärke in dBm; **99 = kein Signal oder DMO**
`cellID`, `cellName`	Zahl/Text	Funkzelle
`volume`, `volumeMax`	Zahl	Lautstärke und zulässiger Höchstwert
`ts`	Zeit	letzte gültige Daten vom Gerät
`enaLIP`	Ja/Nein	Positionsmeldungen aktiv (Protokoll-ID 10)
`enaSDSRC`	Ja/Nein	SDS-Fernsteuerung aktiv (224)
`sdsRCsupported`	Ja/Nein	SDS-Fernsteuerung wird unterstützt, unabhängig von `enaSDSRC`
`enaHMD`	Ja/Nein	Home Mode Display aktiv (220)
`enaCallOut`	Ja/Nein	Alarmierung aktiv (195)
`enaREP`, `enaGATE`	Ja/Nein	DMO-Repeater bzw. -Gateway möglich
`scanStatus`	Zahl	1 = Scan aktiv, 0 = aus, -2 = Gerät kann kein Scanning
`scanGroups`	Feld	Scangruppen mit `GSSI`, `Name`, `Prio`, `PrioKlartext`
`callActive`, `callStatus`	Ja/Nein, Zahl	laufender Ruf
`restrictions`	Zahl	Bitmaske der Gerätesperren, siehe Abschnitt 5.1
`folderName`	Text	Ordner der geschalteten Gruppe
`colForeground`, `colBackground`	`#RRGGBB`	Farben des Geräts in der Oberfläche

A.2 `WSdevUpd` — Geräteänderung (`type: "devupd"`)

Feld	Typ	Bedeutung
`Index`	Zahl	Gerätenummer
`CallActive`, `CallStatus`	Ja/Nein, Zahl	laufender Ruf
`Mode`	Zahl	0 = TMO, 1 = DMO
`GSSI`, `GSSIname`	Text	geschaltete Gruppe mit Klartext
`CID`	**Text**	Funkzellenkennung — hier abweichend von `WSdev.cellID` eine Zeichenkette
`RSSI`, `Volume`	Zahl	Feldstärke, Lautstärke

`ScanStatusIST`	Zahl	Scanzustand
`scanGroups`	Feld	Scangruppen mit Klarnamen
`ScanGroupColl`	Feld	dieselbe Liste ohne Klarnamen (Kopplung)
`ts`	Zeit	Zeitpunkt der letzten gültigen Daten

A.3 `WSstatus` — Statusmeldung (`type: "status"`)

Feld	Typ	Bedeutung
`status`	Text	FMS-Zeichen `0`–`9`, `A`–`F`; leer, wenn es keine Entsprechung gibt
`statusCode`	Text	TETRA-Status hexadezimal, z. B. `8003`
`statusText`	Text	Klartext
`srcSSI`, `srcName`	Text	Absender
`destSSI`, `destName`	Text	Empfänger
`ts`	Zeit	Zeitstempel
`radioID`, `radioName`	Zahl/Text	empfangendes Funkgerät
`remark`	Text	Bemerkung

A.4 `WScall` — Gespräch (`type: "call"`)

Feld	Typ	Bedeutung
`ID`	Text	eindeutig; Beginn und Ende tragen dieselbe ID
`srcSSI`, `srcName`, `srcOPTA`	Text	Sprecher
`destSSI`, `destName`	Text	Ziel, bei Gruppenruf die Gruppe
`tsStart`, `tsEnd`	Zeit	Beginn und Ende
`callType`	Zahl	0 = Einzelruf, 1 = Gruppenruf, 3 = Broadcast, 4 = DMO-Einzelruf
`priority`	Zahl	Priorität
`radioID`, `radioName`	Zahl/Text	empfangendes Funkgerät
`wav`	Ja/Nein	zu diesem Gespräch liegt eine Aufzeichnung auf dem Server vor
`remark`	Text	Bemerkung

> In der Praxis liefern viele Anlagen durchgängig `callType: -1`. Ein unbekannter Wert sollte weggelassen und nicht als Einzelruf dargestellt werden.

A.5 `WSsds` — SDS (`type: "sds"`)

Feld	Typ	Bedeutung
`ID`	Text	eindeutig; eine spätere Empfangsbestätigung trägt dieselbe ID
`ProtocolID`	Zahl	2, 9, 10, 130, 137, 195, 204, 224 — siehe Abschnitt 3.1
`text`	Text	Nachrichteninhalt
`srcSSI`, `srcName`, `destSSI`, `destName`	Text	Absender und Empfänger
`ts`	Zeit	Zeitstempel
`radioID`, `radioName`	Zahl/Text	Funkgerät

`receipt`	Zahl	-1 = keine Bestätigung, 0 = nicht zugestellt, 1 = Empfangs-, 2 = Lesebestätigung
`remark`	Text	Bemerkung. Bei Alarmierung: `Typ;Severity;Vorfall;Status` mit Typ -1 = Alarmierung, 3 = Rückmeldung, 5 = Verfügbarkeit; im Feld Status stehen bei einer Alarmierung die Subgruppen

Eine SDS mit `ProtocolID: 195` ist eine **Alarmierung**, auch wenn sie als `sds`-Paket kommt. Anwendungen mit einer Ereignisliste sollten sie entsprechend einordnen — das eigene `callout`-Paket entsteht nur bei von dieser Instanz ausgelösten Alarmierungen und wird beim Verbindungsaufbau **nicht** nachgeliefert.

A.6 `WSpos` — Position (`type: "pos"`)

Feld	Typ	Bedeutung
`srcSSI`, `srcName`	Text	Absender
`Lat`, `Lon`	Gleitkomma	WGS84; Nord und Ost positiv
`Alt`	Gleitkomma	Höhe
`FixQual`	Zahl	0 = keine Position, 1 = schlechter als 100 m, 2 = besser als 100 m
`ts`	Zeit	Zeitstempel

A.7 `WScallout` — Alarmierung (`type: "callout"`)

Feld	Typ	Bedeutung
`ID`	Text	eindeutig
`text`	Text	Alarmtext
`severityNum`, `severityText`	Zahl/Text	Dringlichkeit
`responseYes`, `responseNO`	Zahl	Zahl der Rückmeldungen
`foreColor`, `backColor`	#RRGGBB	Darstellungsfarben
`srcSSI`, `srcName`, `destName`	Text	Absender und Ziel
`ts`	Zeit	Zeitstempel
`radioID`, `radioName`	Zahl/Text	Funkgerät

A.8 `WSissi` — Teilnehmer (`type: "issi"`)

Feld	Typ	Bedeutung
`ISSI`	Text	Kennung, numerisch
`name`, `OPTA`, `remark`	Text	Rufname, OPTA, Bemerkung
`favorites`	Text	Favoritenlisten; `13` = Liste 1 und 3
`lGSSI`, `lGSSIname`, `lGSSITS`	Text/Zeit	zuletzt aktive Gruppe
`Lat`, `Lon`, `posTS`	Gleitkomma/Zeit	letzte Position
`status`, `statusText`, `statusTS`	Text/Zeit	letzter Status
`IconIdx`	Zahl	Index des Kartensymbols

A.9 `WStse` — Sirene (`type: "tse"`)

Feld	Typ	Bedeutung
`ISSI`, `Name`, `Bemerkung`	Text	Kennung und Bezeichnung
`GSSI`, `SubGr`	Text	Alarmierungsgruppe und Subgruppe
`Lat`, `Lon`	Gleitkomma	Standort
`StatusHex`, `StatusText`, `ts`	Text/Zeit	ausgewerteter Gesamtzustand
`ForeColor`, `BackColor`	ARGB-Int32	Farben des Zustands
`IconIdx`	Zahl	Symbol samt Ampelstufe, `100 + Grundsymbol + Ampel`
`HatStatusFehler`, `HatUnterFehler`	Ja/Nein	Sammelfehler
`TextLetzterStatus`	Text	Klartext der letzten Meldung
`TextAusloesung`, `TS_Ausloesung`	Text/Zeit	letzte Auslösung
`AusloeseStatus`	Zahl	Zustand der Auslösung
`StatusSirene`, `StatusSabotage`, `StatusStromnetz`, `StatusBatterie`, `StatusTemperatur`, `StatusTimeout`	Zahl	Einzelzustände: -1 unbekannt, 0 Fehler, 1 in Ordnung
`TimeoutMin`	Zahl	vereinbarter Meldeabstand in Minuten
`TS_…OK`, `TS_…Fehler`	Zeit	Zeitpunkt der jeweils letzten Meldung je Einzelzustand

A.10 `WStme` — Meldeempfänger (`type: "tme"`)

Feld	Typ	Bedeutung
`ISSI`, `Name`, `NameLang`, `Bemerkung`	Text	Kennung und Bezeichnung
`TA`, `TAts`	Zahl/Zeit	Verfügbarkeit, Schlüssel zu `WSta.TA`
`status`, `statusTS`	Text/Zeit	letzte Alarmierungsantwort, Schlüssel zu `WScostatus.StatusHex`
`Lat`, `Lon`, `posTS`	Gleitkomma/Zeit	letzte Position
`IconIdx`	Zahl	Kartensymbol

A.11 Nachschlagetabellen

`TETRAstatus` (`/api/statustexte.json`): `StatusHex`, `StatusDec`, `Text`, `StatusFMS`, `HomeModeDisplay`, `AntwortPID`, `LStTextZuStatus`, `Permanent`, `LIPRequest`, `SoundFile`, `ForeColor`, `BackColor`.

`WStsestatus` (`/api/tsestatus.json`): `StatusHex`, `StatusDec`, `TypNr`, `Text`, `ForeColor`, `BackColor`.

`WScostatus` (`/api/costatus.json`, Feld `status`): `StatusHex`, `StatusDec`, `Text`, `OptMeldung`, `Available` (-1 unbekannt, 0 nicht verfügbar, 1 verfügbar), `ForeColor`, `BackColor`.

`WSta` (`/api/costatus.json`, Feld `ta`): `TA`, `Text`, `TextKurz`, `ZeitMinuten` (-1 = unbefristet), `ForeColor`, `BackColor`.

A.12 Objekte zum Senden

`WSsendsds` — SDS oder Alarmierung senden:

Feld	Typ	Bedeutung
`destSSI`	Text	Ziel
`ProtocolID`	Zahl	Protokoll-ID
`text`	Text	Inhalt
`encrypt`, `receipt`	Ja/Nein	Verschlüsselung, Empfangsbestätigung

prio	Zahl	Priorität 0–15
COprio	Zahl	Dringlichkeit der Alarmierung, Vorgabe 3
COvorfall	Zahl	Vorfallnummer
COmin	Zahl	Mindeststärke der Rückmeldung
COnoreply	Zahl	-1 Vorgabe, 0 Antwort anfordern, 1 keine Antwort
PrioQueue	Zahl	Vorrang in der Sendewarteschlange
radioID	Zahl	Funkgerät
remark	Text	Bemerkung

WSsendstatus — Status senden: destSSI, statusCode, radioID.

WScmd — Gerät steuern: cmd, Index, Param1, Param2, ParamC, Int1.

WSissiEdit — Teilnehmer ändern: ISSI (Pflicht), Name, Bemerkung, OPTA, Favorites, Status, StatusTyp, IconIdx. Nicht gesendete Felder bleiben unverändert.

Anhang B — Beispiele

Die JSON-Pakete werden ohne Zeilenumbrüche übertragen; hier sind sie zur Lesbarkeit umbrochen.

B.1 Funkgerät (`radio`)

```
{
  "type": "radio", "radioID": 1, "radioName": "F4", "active": 1,
  "vendor": "MOTOROLA", "model": "MTM800FuG", "serial": "938TQG1234",
  "swVersion": "R27.220.9952", "ISSI": "4671234",
  "OPTA": "HEFW.XYZ10100.........1.", "TEI": "000148270542600",
  "MNI": "2621001", "mode": 0, "GSSI": "2691121", "GSSIname": "BG FW",
  "RSSI": 30, "cellID": 12844, "cellName": "Frankfurter Str. 51",
  "volume": 12, "volumeMax": 14, "ts": "\/Date(1499155190641)\/",
  "enaLIP": true, "enaSDSRC": true, "sdsRCsupported": true,
  "enaHMD": false, "enaCallOut": true, "enaREP": false, "enaGATE": false,
  "scanStatus": 1,
  "scanGroups": [ { "GSSI": "2691353", "Name": "MKK GPS",
                    "Prio": "4", "PrioKlartext": "Mittel" } ],
  "callActive": false, "callStatus": 5, "restrictions": 0,
  "folderName": "HE_MKK",
  "colForeground": "#FFFFFF", "colBackground": "#FF0000"
}
```

B.2 Gespräch (`call`)

```
{
  "type": "call", "ID": "20170704091447804412344O2",
  "destSSI": "2691102", "destName": "BG RD",
  "srcSSI": "4123440", "srcName": "DRK 1-83-1",
  "srcOPTA": "HEDRKXYZ1..83RTW......1.",
  "tsStart": "\/Date(1499152487839)\/", "tsEnd": "\/Date(1499152494057)\/",
  "radioID": 2, "radioName": "F5", "callType": -1, "priority": 0,
  "wav": true, "remark": ""
}
```

B.3 Status (`status`)

```
{
  "type": "status", "status": "1", "statusCode": "8003",
  "statusText": "Status 1 - E-Bereit Funk",
  "destSSI": "2669900", "destName": "Daten",
  "srcSSI": "4671234", "srcName": "GG 2-42-1",
  "ts": "\/Date(1499155242655)\/",
  "radioID": 1, "radioName": "Extern", "remark": ""
}
```

B.4 Position (`pos`)

```
{
  "type": "pos", "srcSSI": "4671234", "srcName": "GG 2-42-1",
  "Lat": 50.238576507568359, "Lon": 8.05744457244873,
  "Alt": 0, "FixQual": 2, "ts": "\/Date(1499155243155)\/"
}
```

B.5 SDS, Standard (`sds`, Protokoll-ID 130)

```
{
  "type": "sds", "destSSI": "4561234", "destName": "GG 2-42-1",
  "srcSSI": "5639876", "srcName": "Testgerät", "ProtocolID": 130,
  "text": "Dies ist ein Beispieltext", "ts": "\/Date(1614867506603)\/",
  "radioID": 1, "radioName": "F4", "receipt": -1, "remark": ""
}
```

B.6 SDS, Alarmierung (`sds`, Protokoll-ID 195)

```
{
  "type": "sds", "destSSI": "2691111", "destName": "RD XYZ",
  "srcSSI": "4671234", "srcName": "Leitstelle", "ProtocolID": 195,
  "text": "&20&23Funktionsprobe", "ts": "\/Date(1614868411689)\/",
  "radioID": 4, "radioName": "P8GR", "remark": "-1;3;15;&20 &23"
}
```

B.7 Teilnehmer (`issi`)

```
{
  "type": "issi", "ISSI": "4672476", "name": "GN 1-20-1",
  "OPTA": "HEFW.GN 10100.........1.", "remark": "", "favorites": "13",
  "lGSSI": "2691121", "lGSSIname": "BG FW",
  "lGSSITS": "\/Date(1757000000000)\/",
  "Lat": 50.199, "Lon": 9.186, "posTS": "\/Date(1757000100000)\/",
  "status": "8003", "statusText": "Status 1 - E-Bereit Funk",
  "statusTS": "\/Date(1757000200000)\/", "IconIdx": 12
}
```

B.8 Vollständige Anbindung im Browser

```
// 1. Rechte holen (Windows-1252 beachten!)
async function json(pfad) {
  const antwort = await fetch(pfad, { credentials: 'same-origin' })
  if (!antwort.ok) throw new Error(antwort.status)
  const roh = await antwort.arrayBuffer()
  return JSON.parse(new TextDecoder('windows-1252').decode(roh))
}
const ich = await json('/api/whoami.json')
const darfSenden = (ich.AccessLevel & 8) === 8

// 2. Statustabelle einmal laden
const status = await json('/api/statustexte.json')

// 3. Livedaten - genau EINE Verbindung je Anwendung
const schema = location.protocol === 'https:' ? 'wss' : 'ws'
const port   = location.port ? ':' + location.port : ''
const ws = new WebSocket(`${schema}://${location.hostname}${port}` +
                         `/live.json?MaxAlter=3600&MaxAnz=200&MaxAnzGPS=100`)

ws.onmessage = (e) => {
  const m = JSON.parse(e.data)          // WebSocket ist UTF-8
  switch (m.type) {
    case 'status': /* … */ break
    case 'call':   /* Beginn und Ende ueber m.ID zusammenfuehren */ break
    case 'sds':    /* ProtocolID 195 = Alarmierung */ break
    case 'pos':    /* … */ break
    default:       /* unbekannte Arten verwerfen, nicht melden */
  }
}

// 4. SDS senden
```

```
if (darfSenden) {
  ws.send(JSON.stringify({ type: 'sds', destSSI: '4672476',
                           text: 'Testnachricht', ProtocolID: 130, radioID: 1 }))
}
```

B.9 Senden per HTTP mit auswertbarem Statuscode

```
curl -s -o /dev/null -w "%{http_code}\n" \
  -H "Authorization: Bearer $USERKEY" \
  "https://leitstelle.example.de/api/sds?ziel=4672476&text=Test&typ=130&gerid=1&ReqType=1"
# 231 = an das Funkgeraet uebergeben
```

Anhang C — Hinweise für bestehende Anbindungen

C.1 Korrekturen gegenüber früheren Fassungen

Frühere Angabe	Richtig
`/API/ISSP.zip`	`/api/issi.zip`
Parameter `Text`, `Ziel`, `GerID` groß geschrieben	Schreibung ist gleichgültig, intern wird kleingeschrieben verglichen
„Lautstärke: Maximum bei Motorola 14, bei Sepura 9“	gilt weiterhin, der verbindliche Wert steht als `volumeMax` in `/api/radio.json`
`/API/CONTROL` mit `cmd`-Tabelle	Parameterzuordnung war in der Tabelle verschoben; maßgeblich ist Abschnitt 5.1
Antwort von `/api/sds` nur als HTML	mit `ReqType=1` reiner Text und sprechender Statuscode

C.2 Was eine Anbindung beachten sollte

1. **Zeichenkodierung.** REST ist Windows-1252, WebSocket ist UTF-8 (Abschnitt 1.5). Das ist die häufigste Fehlerursache.
2. **Genau eine WebSocket-Verbindung je Anwendung.** Jede Verbindung kostet beim Aufbau die Nachlieferung der Historie, und diese läuft auf dem Oberflächen-Thread des Servers.
3. **Kein zusätzliches Polling**, wenn der WebSocket steht.
4. **Wiederverbinden mit wachsendem Abstand** (1 s bis 30 s, mit Streuung) und danach den Anfangszustand neu laden.
5. **`ETag`/`If-None-Match` verwenden**, besonders bei den Konfigurationslisten.
6. **Zugangsdaten nicht in die URL.** Sie stehen sonst im Klartext in `Logs\HTTP_*.log`.
7. **Rechte vorab prüfen** über `/api/whoami.json`, statt Befehle ins Leere laufen zu lassen.
8. **Unbekannte Nachrichtenarten verwerfen**, nicht als Fehler behandeln.

C.3 Grenzen

- Der Server bestätigt den **Versand**, nicht die **Zustellung** — außer über `/api/result/sds`.
- Ein SDS-Versand ohne das Recht `SendSDS` schlägt über den WebSocket **still** fehl.
- Für Gespräche und SDS gibt es keine Historien-Abfrage über HTTP; was über den WebSocket nachgeliefert wird, ist durch `MaxAlter` und `MaxAnz` begrenzt.
- Die Verfügbarkeit der Meldeempfänger wird nicht über den WebSocket nachgeführt.

Kontakt

Wir freuen uns über Ihre Anfragen.

Status 3 IT GmbH Dietrichsberg 33 63607 Wächtersbach

Fon: +49 (6051) 70076-00 Fax: +49 (6051) 70076-16 E-Mail: tetracontrol@status3.it Web: http://www.tetracontrol.de

HRB 94696, RG Hanau · USt-IdNr. DE293587360

TETRAcontrol® ist eine eingetragene Marke der Status 3 IT GmbH.