Prometheus Metrics and Monitoring Guide

Overview

OmniTAS exports comprehensive operational metrics in Prometheus format for monitoring, alerting, and observability. This guide covers all available metrics, their usage, troubleshooting, and monitoring best practices.

Metrics Endpoint

All metrics are exposed at: `http://<tas-ip>:8080/metrics`

Complete Metric Reference

Diameter Metrics

`diameter_response_duration_milliseconds`

Type: Histogram **Labels:** `application` (ro, sh), `command` (ccr, cca, etc), `result` (success, error, timeout) **Buckets:** 10, 50, 100, 250, 500, 1000, 2500, 5000, 10000 ms **Description:** Duration of Diameter requests in milliseconds

Usage:

```
# Average Diameter Response Time
rate(diameter_response_duration_milliseconds_sum[5m]) /
rate(diameter_response_duration_milliseconds_count[5m])

# P95 Diameter latency
histogram_quantile(0.95,
rate(diameter_response_duration_milliseconds_bucket[5m]))
```

Alert When:

- P95 > 1000ms - Slow Diameter responses

`diameter_requests_total`

Type: Counter **Labels:** `application` (ro, sh), `command` (ccr, udr, etc)
Description: Total number of Diameter requests sent

Usage:

```
# Request rate
rate(diameter_requests_total[5m])
```

`diameter_responses_total`

Type: Counter **Labels:** `application` (ro, sh), `command` (ccr, udr, etc), `result_code` (2001, 3002, 5xxx, etc) **Description:** Total number of Diameter responses received

Usage:

```
# Success rate
rate(diameter_responses_total{result_code="2001"}[5m]) /
rate(diameter_responses_total[5m]) * 100
```

`diameter_peer_state`

Type: Gauge **Labels:** `peer_host`, `peer_realm`, `application` (ro, sh)
Description: State of Diameter peers (1=up, 0=down) **Update interval:**

Every 10 seconds

Usage:

```
# Check for down peers
diameter_peer_state == 0
```

Alert When:

- Any peer down for > 1 minute

Dialplan Generation Metrics

1. HTTP Request Metrics

`http_dialplan_request_duration_milliseconds`

Type: Histogram **Labels:** `call_type` (mt, mo, emergency, unknown) **Description: End-to-end HTTP request duration** from when the dialplan HTTP request is received to when the response is sent. This includes all processing: parameter parsing, authorization, Diameter lookups (Sh/Ro), HLR lookups (SS7 MAP), and XML generation.

Usage:

```
# Average end-to-end HTTP request time
rate(http_dialplan_request_duration_milliseconds_sum[5m]) /
rate(http_dialplan_request_duration_milliseconds_count[5m])

# P95 by call type
histogram_quantile(0.95,
  rate(http_dialplan_request_duration_milliseconds_bucket[5m])
) by (call_type)

# Compare MT vs MO performance
histogram_quantile(0.95,

rate(http_dialplan_request_duration_milliseconds_bucket{call_type="mt
[5m])
)
vs
histogram_quantile(0.95,

rate(http_dialplan_request_duration_milliseconds_bucket{call_type="mo
[5m])
)
```

Alert When:

- P95 > 2000ms - Slow HTTP response times
- P95 > 3000ms - Critical performance issue
- P99 > 5000ms - Severe performance degradation
- Any requests showing `call_type="unknown"` - Call type detection failure

Insights:

- This is the **most important metric** for understanding user-facing latency
- Typical values: P50: 100-500ms, P95: 500-2000ms, P99: 1000-3000ms
- Includes all component timings (Sh + HLR + OCS + processing)
- If this is slow, drill down into component metrics (subscriber_data, hlr_data, ocs_authorization)
- Expected range: 100ms (fast local calls) to 5000ms (slow with retries/timeouts)

Important Notes:

- Replaces the older `dialplan_generation_duration_milliseconds` metric which only measured XML generation
- Accurately reflects what FreeSWITCH/SBC experiences
- Use this for SLA monitoring and capacity planning

2. Subscriber Data Metrics

`subscriber_data_duration_milliseconds`

Type: Histogram **Labels:** `result` (success, error) **Description:** Time taken to retrieve subscriber data from Sh interface (HSS)

Usage:

```
# Average Sh lookup time
rate(subscriber_data_duration_milliseconds_sum[5m]) /
rate(subscriber_data_duration_milliseconds_count[5m])

# 95th percentile Sh lookup time
histogram_quantile(0.95,
  rate(subscriber_data_duration_milliseconds_bucket[5m])
)
```

Alert When:

- P95 > 100ms - Slow HSS responses
- P95 > 500ms - Critical HSS performance issue

`subscriber_data_lookups_total`

Type: Counter **Labels:** `result` (success, error) **Description:** Total number of subscriber data lookups

Usage:

```
# Sh lookup rate
rate(subscriber_data_lookups_total[5m])

# Sh error rate
rate(subscriber_data_lookups_total{result="error"}[5m])

# Sh success rate percentage
(rate(subscriber_data_lookups_total{result="success"}[5m]) /
 rate(subscriber_data_lookups_total[5m])) * 100
```

Alert When:

- Error rate > 5% - HSS connectivity issues
- Error rate > 20% - Critical HSS failure

2. HLR Data Metrics

`hlr_data_duration_milliseconds`

Type: Histogram **Labels:** `result` (success, error) **Description:** Time taken to retrieve HLR data via SS7 MAP

Usage:

```
# Average HLR lookup time
rate(hlr_data_duration_milliseconds_sum[5m]) /
rate(hlr_data_duration_milliseconds_count[5m])

# 95th percentile HLR lookup time
histogram_quantile(0.95,
  rate(hlr_data_duration_milliseconds_bucket[5m])
)
```

Alert When:

- P95 > 500ms - Slow SS7 MAP responses
- P95 > 2000ms - Critical SS7 MAP issue

`hlr_lookups_total`

Type: Counter **Labels:** `result_type` (msrn, forwarding, error, unknown)
Description: Total HLR lookups by result type

Usage:

```
# HLR lookup rate by type
rate(hlr_lookups_total[5m])

# MSRN discovery rate (roaming subscribers)
rate(hlr_lookups_total{result_type="msrn"}[5m])

# Call forwarding discovery rate
rate(hlr_lookups_total{result_type="forwarding"}[5m])

# HLR error rate
rate(hlr_lookups_total{result_type="error"}[5m])
```

Alert When:

- Error rate > 10% - SS7 MAP issues
- Sudden drop in MSRN rate - Possible roaming issue

Insights:

- High MSRN rate indicates many roaming subscribers
- High forwarding rate indicates many forwarded calls
- Compare to call volume for roaming percentage

3. OCS Authorization Metrics

`ocs_authorization_duration_milliseconds`

Type: Histogram **Labels:** `result` (success, error) **Description:** Time taken for OCS authorization

Usage:

```
# Average OCS auth time
rate(ocs_authorization_duration_milliseconds_sum[5m]) /
rate(ocs_authorization_duration_milliseconds_count[5m])

# 95th percentile OCS auth time
histogram_quantile(0.95,
  rate(ocs_authorization_duration_milliseconds_bucket[5m])
)
```

Alert When:

- P95 > 1000ms - Slow OCS responses
- P95 > 5000ms - Critical OCS performance issue

`ocs_authorization_attempts_total`

Type: Counter **Labels:** `result` (success, error), `skipped` (yes, no)
Description: Total OCS authorization attempts

Usage:

```
# OCS authorization rate
rate(ocs_authorization_attempts_total{skipped="no"}[5m])

# OCS error rate
rate(ocs_authorization_attempts_total{result="error",skipped="no"}
[5m])

# OCS skip rate (emergency, voicemail, etc.)
rate(ocs_authorization_attempts_total{skipped="yes"}[5m])

# OCS success rate percentage
(rate(ocs_authorization_attempts_total{result="success",skipped="no"}
[5m]) /
 rate(ocs_authorization_attempts_total{skipped="no"}[5m])) * 100
```

Alert When:

- Error rate > 5% - OCS connectivity issues
- Success rate < 95% - OCS declining too many calls

Insights:

- High skip rate indicates many emergency/free calls
- Error rate spikes indicate OCS outages
- Compare success rate to business expectations

4. Call Processing Metrics

`call_param_errors_total`

Type: Counter **Labels:** `error_type` (parse_failed, missing_required_params) **Description:** Call parameter parsing errors

Usage:

```
# Parameter error rate
rate(call_param_errors_total[5m])

# Errors by type
rate(call_param_errors_total[5m]) by (error_type)
```

Alert When:

- Any errors > 0 - Indicates malformed call parameter requests
- Errors > 1% of call volume - Critical issue

`authorization_decisions_total`

Type: Counter **Labels:** `disposition` (mt, mo, emergency, unauthorized), `result` (success, error) **Description:** Authorization decisions by call type

Usage:

```
# Authorization rate by disposition
rate(authorization_decisions_total[5m]) by (disposition)

# MT call rate
rate(authorization_decisions_total{disposition="mt"}[5m])

# MO call rate
rate(authorization_decisions_total{disposition="mo"}[5m])

# Emergency call rate
rate(authorization_decisions_total{disposition="emergency"}[5m])

# Unauthorized call rate
rate(authorization_decisions_total{disposition="unauthorized"}
[5m])
```

Alert When:

- Unauthorized rate > 1% - Possible attack or misconfiguration
- Sudden spike in emergency calls - Possible emergency event
- Unexpected change in MT/MO ratio - Possible issue

Insights:

- MT/MO ratio indicates traffic patterns
- Emergency call rate indicates service usage
- Unauthorized rate indicates security posture

`freeswitch_variable_set_duration_milliseconds`

Type: Histogram **Labels:** `batch_size` (1, 5, 10, 25, 50, 100) **Description:** Time to set Dialplan Variables

Usage:

```
# Average variable set time
rate(freeswitch_variable_set_duration_milliseconds_sum[5m]) /
rate(freeswitch_variable_set_duration_milliseconds_count[5m])

# Variable set time by batch size
histogram_quantile(0.95,
  rate(freeswitch_variable_set_duration_milliseconds_bucket[5m])
) by (batch_size)
```

Alert When:

- P95 > 100ms - Slow variable set performance
- Growing trend - Possible system performance issue

5. Module Processing Metrics

`dialplan_module_duration_milliseconds`

Type: Histogram **Labels:** `module` (MT, MO, Emergency, CallParams, etc.), `call_type` **Description:** Processing time for each dialplan module

Usage:

```
# Processing time by module
histogram_quantile(0.95,
  rate(dialplan_module_duration_milliseconds_bucket[5m])
) by (module)

# MT module processing time
histogram_quantile(0.95,
  rate(dialplan_module_duration_milliseconds_bucket{module="MT"}
[5m])
)
```

Alert When:

- Any module P95 > 500ms - Performance issue
- Growing trend in any module - Potential leak or issue

Insights:

- Identify which module is slowest
- Optimize the slowest modules first
- Compare module times across call types

6. Call Volume Metrics

`call_attempts_total`

Type: Counter **Labels:** `call_type` (mt, mo, emergency, unauthorized), `result` (success, rejected) **Description:** Total call attempts

Usage:

```
# Call attempt rate
rate(call_attempts_total[5m])

# Success rate by call type
(rate(call_attempts_total{result="success"}[5m]) /
 rate(call_attempts_total[5m])) * 100 by (call_type)

# Rejected call rate
rate(call_attempts_total{result="rejected"}[5m])
```

Alert When:

- Rejected rate > 5% - Possible issue
- Sudden drop in call volume - Service outage
- Sudden spike in call volume - Possible attack

`active_calls`

Type: Gauge **Labels:** `call_type` (mt, mo, emergency) **Description:** Currently active calls

Usage:

```
# Current active calls
active_calls

# Active calls by type
active_calls by (call_type)

# Peak active calls (last hour)
max_over_time(active_calls[1h])
```

Alert When:

- Active calls > capacity - Overload
- Active calls = 0 for extended time - Service down

7. Simulation Metrics

`call_simulations_total`

Type: Counter **Labels:** `call_type` (mt, mo, emergency, unauthorized), `source` (web, api) **Description:** Call simulations run

Usage:

```
# Simulation rate
rate(call_simulations_total[5m])

# Simulations by type
rate(call_simulations_total[5m]) by (call_type)
```

Insights:

- Track diagnostic tool usage
- Identify heavy users
- Correlate with troubleshooting activity

8. SS7 MAP Metrics

`ss7_map_http_duration_milliseconds`

Type: Histogram **Labels:** `operation` (sri, prn), `result` (success, error, timeout) **Buckets:** 10, 50, 100, 250, 500, 1000, 2500, 5000, 10000 ms **Description:** Duration of SS7 MAP HTTP requests in milliseconds

Usage:

```
# SS7 MAP Error Rate
rate(ss7_map_operations_total{result="error"}[5m]) /
rate(ss7_map_operations_total[5m]) * 100
```

Alert When:

- P95 > 500ms - Slow SS7 MAP responses
- Error rate > 50% - Critical SS7 MAP issue

`ss7_map_operations_total`

Type: Counter **Labels:** `operation` (sri, prn), `result` (success, error) **Description:** Total number of SS7 MAP operations

9. Online Charging Metrics

`online_charging_events_total`

Type: Counter **Labels:** `event_type` (authorize, answer, reauth, hangup), `result` (success, nocredit, error, timeout) **Description:** Total number of online charging events

Usage:

```
# OCS Credit Failures
rate(online_charging_events_total{result="nocredit"}[5m])
```

Alert When:

- High rate of credit failures

10. System State Metrics

`tracked_registrations`

Type: Gauge **Description:** Number of currently active SIP registrations (from FreeSWITCH Sofia registration database) **Update interval:** Every 10 seconds

Notes:

- Automatically decrements when registrations expire (FreeSWITCH manages expiration)

`tracked_call_sessions`

Type: Gauge **Description:** Number of currently tracked call sessions in ETS **Update interval:** Every 10 seconds

11. HTTP Request Metrics

`http_requests_total`

Type: Counter **Labels:** `endpoint` (dialplan, call_event, directory, voicemail, sms_ccr, metrics), `status_code` (200, 400, 500, etc) **Description:** Total number of HTTP requests by endpoint

Usage:

```
# HTTP Error Rate
rate(http_requests_total{status_code=~"5.."}[5m]) /
rate(http_requests_total[5m]) * 100
```

Alert When:

- HTTP 5xx error rate > 10%

12. Call Rejection Metrics

`call_rejections_total`

Type: Counter **Labels:** `call_type` (mo, mt, emergency, unknown), `reason` (nocredit, unauthorized, parse_failed, missing_params, hlr_error, etc) **Description:** Total number of call rejections by reason

Usage:

```
# Call Rejection Rate by Reason
sum by (reason) (rate(call_rejections_total[5m]))
```

Alert When:

- Rejection rate > 1/sec - Investigation needed

13. Event Socket Connection Metrics

`event_socket_connected`

Type: Gauge **Labels:** `connection_type` (main, log_listener) **Description:** Event Socket connection state (1=connected, 0=disconnected) **Update interval:** Real-time on connection state changes

Usage:

```
# Event Socket Connection Status
event_socket_connected
```

Alert When:

- Connection down for > 30 seconds

`event_socket_reconnections_total`

Type: Counter **Labels:** `connection_type` (main, log_listener), `result` (attempting, success, failed) **Description:** Total number of Event Socket reconnection attempts

Grafana Dashboard Integration

The metrics can be visualized in Grafana using the Prometheus data source. Recommended panels:

Dashboard 1: Call Volume

- Active calls gauge
- Call attempts rate by type (MO/MT/Emergency)
- Call rejection rate

Dashboard 2: Diameter Performance

- Response time heatmap
- Request/response rates
- Peer status table
- Error rate by result code

Dashboard 3: Online Charging Health

- Credit authorization success rate
- "No credit" event rate
- OCS timeout rate

Dashboard 4: System Performance

- Dialplan generation latency (P50/P95/P99)
- SS7 MAP response times
- Overall system availability

Recommended Grafana Dashboard Layout

Row 1: Call Volume

- Call attempts rate (by type)

- Active calls gauge
- Success rate percentage

Row 2: Performance

- P95 HTTP dialplan request time (by call type) - **PRIMARY METRIC**
- P95 Sh lookup time
- P95 HLR lookup time
- P95 OCS authorization time
- P95 dialplan module processing time (by module)

Row 3: Success Rates

- Sh lookup success rate
- HLR lookup success rate
- OCS authorization success rate
- Call attempt success rate

Row 4: Module Performance

- P95 processing time by module
- Module call counts

Row 5: Errors

- Parameter errors
- Unauthorized attempts
- Sh errors
- HLR errors
- OCS errors

Critical Alerts

Priority 1 (Page immediately):

```
# Dialplan completely down
rate(call_attempts_total[5m]) == 0

# HSS completely down
rate(subscriber_data_lookups_total{result="error"}[5m]) /
rate(subscriber_data_lookups_total[5m]) > 0.9

# OCS completely down
rate(ocs_authorization_attempts_total{result="error"}[5m]) /
rate(ocs_authorization_attempts_total[5m]) > 0.9
```

Priority 2 (Alert):

```
# Slow dialplan generation
histogram_quantile(0.95,
  rate(dialplan_generation_duration_milliseconds_bucket[5m])
) > 1000

# High HSS error rate
rate(subscriber_data_lookups_total{result="error"}[5m]) /
rate(subscriber_data_lookups_total[5m]) > 0.2

# High OCS error rate
rate(ocs_authorization_attempts_total{result="error"}[5m]) /
rate(ocs_authorization_attempts_total[5m]) > 0.1
```

Priority 3 (Warning):

```
# Elevated HSS latency
histogram_quantile(0.95,
  rate(subscriber_data_duration_milliseconds_bucket[5m])
) > 100

# Elevated OCS latency
histogram_quantile(0.95,
  rate(ocs_authorization_duration_milliseconds_bucket[5m])
) > 1000

# Moderate error rate
rate(call_attempts_total{result="rejected"}[5m]) /
rate(call_attempts_total[5m]) > 0.05
```

Alerting Examples

Diameter Peer Down

```
alert: DiameterPeerDown
expr: diameter_peer_state == 0
for: 1m
annotations:
  summary: "Diameter peer {{ $labels.peer_host }} is down"
```

High Diameter Latency

```
alert: HighDiameterLatency
expr: histogram_quantile(0.95,
rate(diameter_response_duration_milliseconds_bucket[5m])) > 1000
for: 5m
annotations:
  summary: "Diameter P95 latency above 1s"
```

OCS Credit Failures

```
alert: HighOCSCreditFailures
expr: rate(online_charging_events_total{result="nocredit"}[5m]) >
0.1
for: 2m
annotations:
  summary: "High rate of OCS credit failures"
```

SS7 MAP Gateway Errors

```
alert: SS7MapErrors
expr: rate(ss7_map_operations_total{result="error"}[5m]) /
rate(ss7_map_operations_total[5m]) > 0.5
for: 3m
annotations:
  summary: "SS7 MAP error rate above 50%"
```

Event Socket Disconnected

```
alert: EventSocketDown
expr: event_socket_connected == 0
for: 30s
annotations:
  summary: "Event Socket {{ $labels.connection_type }}
disconnected"
```

High Call Rejection Rate

```
alert: HighCallRejectionRate
expr: rate(call_rejections_total[5m]) > 1
for: 2m
annotations:
  summary: "High call rejection rate: {{ $value }} rejections/sec"
```

HTTP Error Rate High

```
alert: HighHTTPErrorRate
expr: rate(http_requests_total{status_code=~"5.."}[5m]) /
rate(http_requests_total[5m]) > 0.1
for: 3m
annotations:
  summary: "HTTP 5xx error rate above 10%"
```

Troubleshooting with Metrics

Problem: Call type showing as "unknown"

Symptoms:

- All metrics show `call_type="unknown"` instead of `mt`, `mo`, or `emergency`
- Cannot differentiate performance between call types

Root Cause: The call type extraction is failing or not being properly passed through the processing pipeline.

Investigation:

1. Check logs for "HTTP dialplan request" messages - they should show the correct call type
2. Review system logs for call type processing errors

Resolution: Contact support if call type detection continues to fail.

Problem: Calls are slow

Investigation:

1. Check `http_dialplan_request_duration_milliseconds` P95 - **START HERE**
2. If high, check component timings:
    - Check `subscriber_data_duration_milliseconds` for Sh delays

- Check `hlr_data_duration_milliseconds` for HLR delays
- Check `ocs_authorization_duration_milliseconds` for OCS delays
- Check `dialplan_module_duration_milliseconds` for module-specific delays

3. Check if `call_type="unknown"` - indicates call type detection failure
4. Compare MT vs MO vs Emergency processing times
5. Correlate with system logs for detailed error messages

Resolution: Optimize the slowest component

Problem: Calls are failing

Investigation:

1. Check `call_attempts_total{result="rejected"}` rate
2. Check `subscriber_data_lookups_total{result="error"}` for Sh issues
3. Check `hlr_lookups_total{result_type="error"}` for HLR issues
4. Check `ocs_authorization_attempts_total{result="error"}` for OCS issues
5. Check `authorization_decisions_total{disposition="unauthorized"}` for auth issues

Resolution: Fix the failing component

Problem: High load

Investigation:

1. Check `active_calls` current value
2. Check `call_attempts_total` rate
3. Check if rate matches expected traffic
4. Compare MT vs MO ratio
5. Check for unusual patterns (spikes, steady growth)

Resolution: Scale up or investigate unusual traffic

Problem: Roaming issues

Investigation:

1. Check `hlr_lookups_total{result_type="msrn"}` rate
2. Check `hlr_data_duration_milliseconds` for delays
3. Use HLR Lookup tool for specific subscribers
4. Check if MSRN is being retrieved correctly

Resolution: Fix HLR connectivity or configuration

Performance Baselines

Typical Values (Well-Tuned System)

- **HTTP dialplan request (end-to-end):** P50: 100-500ms, P95: 500-2000ms, P99: 1000-3000ms
- **Sh lookup time:** P50: 15ms, P95: 50ms, P99: 100ms
- **HLR lookup time:** P50: 100ms, P95: 300ms, P99: 800ms
- **OCS auth time:** P50: 150ms, P95: 500ms, P99: 1500ms
- **Dialplan module processing:** P50: 1-5ms, P95: 10-25ms, P99: 50ms
- **Sh success rate:** > 99%
- **HLR success rate:** > 95% (lower is normal due to offline subscribers)
- **OCS success rate:** > 98%
- **Call success rate:** > 99%

Note: HTTP dialplan request time is the sum of all component times plus overhead. It should roughly equal: Sh lookup + HLR lookup + OCS auth + dialplan module processing + network/parsing overhead. Minimum expected time is ~100ms (when only Sh lookup is needed), maximum typical time is ~2000ms (with all lookups and retries).

Capacity Planning

Monitor these trends:

- Growth in `call_attempts_total` rate
- Growth in `active_calls` peak
- Stable or improving P95 latencies
- Stable or improving success rates

Plan for scaling when:

- Active calls approaching 80% of capacity
- P95 latencies growing despite stable load
- Success rates declining despite stable external systems

Integration with Logging

Correlate metrics with logs:

1. High error rate in metrics → Search logs for ERROR messages
2. Slow response times → Search logs for WARNING messages about timeouts
3. Specific call issues → Search logs by call ID or phone number
4. Use simulation tool to reproduce and debug

Best Practices

1. **Set up dashboards before issues occur**
2. **Define alert thresholds based on your baseline**
3. **Test alerts by using Call Simulator**
4. **Review metrics weekly to identify trends**
5. **Correlate metrics with business events** (campaigns, outages, etc.)
6. **Use metrics to justify infrastructure investments**
7. **Share dashboards with operations team**
8. **Document your alert response procedures**

Configuration

Metrics collection is automatically enabled when the application starts. The metrics endpoint is exposed on the same port as the API (default: 8080).

To configure Prometheus to scrape metrics, add this job to your `prometheus.yml`:

```
scrape_configs:
  - job_name: 'omnitas'
    static_configs:
      - targets: ['<tas-ip>:8080']
    metrics_path: '/metrics'
    scrape_interval: 10s
```

Metric Cardinality

The metrics are designed with controlled cardinality to avoid overwhelming Prometheus:

- **Peer labels:** Limited to configured peers only
- **Call types:** Fixed set (mo, mt, emergency, unauthorized)
- **Result codes:** Limited to actual Diameter/OCS result codes received
- **Operations:** Fixed set per interface (sri/prn for MAP, ccr/cca for Diameter)

Total estimated time series: ~200-500 depending on number of configured peers and active result codes.

Metric Retention

Recommended retention periods:

- **Raw metrics:** 30 days (high resolution)
- **5-minute aggregates:** 90 days
- **1-hour aggregates:** 1 year

- **Daily aggregates:** 5 years

This supports:

- Real-time troubleshooting (raw metrics)
- Weekly/monthly analysis (5-min/1-hour aggregates)
- Capacity planning (daily aggregates)
- Historical comparison (yearly aggregates)

HLR Lookup and Call Simulator - User Guide

Overview

Diagnostic tools to help operations staff troubleshoot call routing. The **HLR Lookup** and **Call Simulator** are read-only and do not affect live traffic. The **Echo Test Call** is different — it places a real MT call (see its own section).

HLR Lookup Tool

Purpose

The HLR Lookup tool queries the Home Location Register (HLR) via SS7 MAP protocol to retrieve real-time subscriber routing information.

Access

Navigate to `/hlr` or click "HLR" in the navigation menu.

What It Shows

For any phone number, the HLR Lookup displays:

1. **MSRN (Mobile Station Roaming Number)**
    - Temporary routing number assigned when subscriber roams to 2G/3G network
    - Only present if subscriber is currently roaming
    - Used by the dialplan to route calls to roaming subscriber's current location
2. **Call Forwarding Settings**

- Real-time call forwarding configuration from HLR
- Types: Unconditional, Busy, No-Reply, Not-Reachable
- Shows forwarding destination number
- Shows if notification is enabled

3. **Dialplan Variables**

- Exactly which channel variables will be set
- Variables match those used in actual call processing
- Shows how HLR data overrides Sh data

Use Cases

Diagnosing Roaming Issues

Scenario: Incoming call to roaming subscriber fails or routes incorrectly

Steps:

1. Open HLR Lookup page
2. Enter the subscriber's phone number
3. Click "Lookup HLR Data"
4. Check for MSRN in results
5. If MSRN present: Subscriber is roaming, verify MSRN is valid
6. If no MSRN: Subscriber may be in LTE/VoLTE (no MSRN needed)

Verifying Call Forwarding

Scenario: Call forwarding not working as expected

Steps:

1. Open HLR Lookup page
2. Enter the subscriber's phone number
3. Click "Lookup HLR Data"
4. Look for "Call Forwarding" in results
5. Verify forwarding type (Unconditional, Busy, etc.)

6. Verify forwarding destination number
7. Note: HLR data overrides any Sh/HSS data

Testing HLR Connectivity

Scenario: Verify SS7 MAP gateway is working

Steps:

1. Open HLR Lookup page
2. Enter any known subscriber number
3. Click "Lookup HLR Data"
4. Check for "Error" in results
5. If error: Check SS7 MAP gateway connectivity
6. Common errors:
    - "SS7 MAP is disabled" - Check configuration
    - "Timeout" - Network issue to HLR
    - "No VLR Number" - Subscriber offline or doesn't exist

Information Box

The HLR Lookup page includes educational information explaining:

- What MSRN is and when it's used
- How call forwarding works in HLR
- How this integrates with call processing
- SS7 MAP protocol basics

Call Simulator Tool

Purpose

The Call Simulator allows you to simulate complete call routing without actually placing a call or affecting live traffic.

Access

Navigate to `/simulator` or click "Simulator" in the navigation menu.

Features

Input Parameters

1. **Source Number (Caller)**
    - Phone number of calling party
    - For MT calls: Can be any number
    - For MO calls: Must be provisioned subscriber
2. **Destination Number (Called Party)**
    - Phone number of called party
    - For MT calls: Must be provisioned subscriber
    - For MO calls: Can be any number
    - For Emergency: Use "urn:service:sos" or similar
3. **Source IP Address**
    - IP address of SIP signaling source
    - Must be in `allowed_sbc_source_ips` (for MT) or `allowed_cscf_ips` (for MO)
    - Determines call disposition (MT vs MO)
4. **Force Disposition**
    - Auto: Determine from IP address (normal behavior)
    - MT: Force Mobile Terminating (incoming)
    - MO: Force Mobile Originating (outgoing)
    - Emergency: Force emergency call processing
5. **Options**
    - Skip OCS Authorization: Bypass online charging (faster simulation)
    - Skip HLR Lookup: Bypass SS7 MAP query (faster simulation)

Output

The simulator shows comprehensive results:

1. **Call Type Banner**
    - MT, MO, or Emergency
    - Color-coded for quick identification
    - Shows source and destination numbers
2. **Processing Steps** (Left Column)
    - **Subscriber Data**: Results from Sh interface (HSS)
    - **HLR Data**: Results from SS7 MAP lookup (MT only)
    - **OCS Authorization**: Results from online charging (MO only)
    - **On-Net Status**: Whether destination is on your network (MO only)
3. **Dialplan Variables** (Right Column)
    - Every variable that would be set on the channel
    - Sorted alphabetically for easy reading
    - Color-coded values (green for normal, red for errors)
4. **Processing Notes**
    - Step-by-step explanation of what happened
    - Describes data flow and decision points
    - Helps understand why certain variables were set

Use Cases

Pre-Flight Testing

Scenario: Testing configuration change before deploying to production

Steps:

1. Make configuration change in dev/test environment
2. Open Call Simulator

3. Test multiple scenarios:
    - MT call from your SBC
    - MO call from your CSCF
    - Emergency call
    - On-net destination
    - Off-net destination
4. Verify all variables are correct
5. Check processing notes for any issues
6. Deploy to production with confidence

Debugging MT Call Issues

Scenario: Incoming calls to subscriber failing

Steps:

1. Open Call Simulator
2. Enter destination as the problem subscriber
3. Enter source as test number
4. Set source IP to your SBC IP
5. Leave Force Disposition as "Auto"
6. Click "Simulate Call"
7. Check Subscriber Data section for Sh lookup success
8. Check HLR Data section for MSRN or forwarding
9. Check Final Variables for `hangup_case`
10. If `hangup_case` is "UNALLOCATED_NUMBER": Subscriber not provisioned
11. If variables look correct: Issue may be in dialplan template

Debugging MO Call Issues

Scenario: Outgoing calls from subscriber failing

Steps:

1. Open Call Simulator
2. Enter source as the problem subscriber

3. Enter destination as test number
4. Set source IP to your CSCF IP
5. Uncheck "Skip OCS Authorization" if testing charging
6. Click "Simulate Call"
7. Check Caller Data section for Sh lookup success
8. Check OCS Authorization section for success/failure
9. Check On-Net Status to verify correct routing
10. Check Final Variables for `allocated_time` or `hangup_case`
11. If `hangup_case` is "OUTGOING_CALL_BARRED": OCS denied the call

Testing Emergency Call Handling

Scenario: Verify emergency calls work correctly

Steps:

1. Open Call Simulator
2. Enter source as test subscriber
3. Enter destination as "urn:service:sos"
4. Set any source IP (emergency calls bypass IP auth)
5. Click "Simulate Call"
6. Verify Call Type shows "Emergency (SOS)"
7. Verify `hangup_case` is "none" (emergency calls always proceed)
8. Check that OCS and HLR were bypassed
9. Verify caller data was retrieved for location info

Training Staff

Scenario: Teaching operations staff how call routing works

Steps:

1. Open Call Simulator
2. Run various scenarios and explain each section:
    - Show MT call and explain Sh + HLR lookups
    - Show MO call and explain OCS authorization

    - Show Emergency call and explain bypass behavior
    - Show unauthorized IP and explain rejection
3. Have staff try different combinations
4. Use Processing Notes to explain each decision
5. Compare variables between different scenarios

Comparing Sh vs HLR Data

Scenario: Understanding how HLR overrides Sh data

Steps:

1. Open Call Simulator for MT call
2. Uncheck "Skip HLR Lookup"
3. Click "Simulate Call"
4. Compare Subscriber Data variables vs HLR Data variables
5. Check Final Variables to see which values won
6. Note: HLR data always takes precedence for:
    - MSRN
    - `call_forward_all_destination`
    - `call_forward_not_reachable_destination`

Tips

- Use "Skip OCS Authorization" and "Skip HLR Lookup" for faster simulations when testing other aspects
- Copy/paste phone numbers from logs into simulator for quick testing
- Use "Force Disposition" to test specific call types regardless of IP
- Check Processing Notes if you're unsure why certain variables were set
- Run simulation multiple times to verify consistency
- Compare simulation results to actual call logs

Limitations

The simulator:

- Does NOT actually place calls
- Does NOT affect the call routing system
- Does NOT consume OCS quota (even if OCS is queried)
- Does NOT generate CDRs
- Is safe to use on production systems

The simulator DOES:

- Query actual Sh interface (HSS) if not skipped
- Query actual HLR via SS7 MAP if not skipped
- Query actual OCS if not skipped
- Show exactly what would happen in real call
- Use real configuration values

Echo Test Call (real MT call)

> Unlike the Call Simulator above, this **places a real call**. It rings the destination subscriber and runs them through the full terminating path (Sh lookup, OCS, CDR, media).

Purpose

Start a real **Mobile Terminating** call to a subscriber that connects them to a FreeSWITCH **echo** application, so they hear their own audio back. Useful for end-to-end verification of the MT path and the media/codec leg without a real upstream caller.

API

`POST /simulator/echo_call`

Field	Required	Description
`destination`	Yes	MSISDN to ring — the **terminating subscriber** (must be provisioned/registered). Digits, optional leading `+`.
`source`	No	MSISDN shown as the caller ID. Digits, optional leading `+`.

```
curl -k -X POST https://<tas>:8444/simulator/echo_call \
  -H 'content-type: application/json' \
  -d '{"destination":"99988037866","source":"61400000001"}'
```

Response (issued via `bgapi`, returns immediately):

```
{
  "ok": true,
  "command": "bgapi originate
{tas_call_reason=echo_test,ignore_early_media=true,origination_caller
&echo",
  "result": "+OK Job-UUID: <uuid>"
}
```

How it works

It issues a FreeSWITCH `originate` that **loops the call back into the local internal profile** (`sofia/internal/<destination>@<tas_ip>`). Because the INVITE then arrives from the TAS's own IP — which is in `allowed_sbc_source_ips` — it is classified as **MT (as if delivered by the upstream SBC)** and runs the real MT dialplan, routing out to the subscriber via the IMS core. The originate's `&echo` answers, so the rung subscriber hears their own audio.

The loopback target is taken **server-side** from the Diameter listen IP (the local box); it is not a request parameter.

What it does / doesn't do

Unlike the dry-run simulator, the echo call **DOES**:

- Place a real call and ring the destination subscriber
- Run Sh lookup, OCS authorization, and generate a CDR / Ro CCRs
- Exercise the real MT dialplan and media path

It is **safe against MO / toll fraud**:

- Always **MT, never MO** — the loopback arrives from the local IP, which can never be a CSCF source IP (the only thing that yields MO).
- It **cannot dial an arbitrary outbound number** — the MT bridge target comes from the Sh subscriber lookup (`NokiaCSCF/+<msisdn>`), not the raw `destination`; an unknown destination has no MSISDN and the bridge fails. It can only ring **provisioned on-net subscribers**.
- The loopback IP is **server-side only**, so the call can never be aimed at an external gateway. `source`/`destination` are validated as phone numbers before reaching the ESL command (no command injection).

Caller ID (`source`) is shown to the rung subscriber as provided — keep the endpoint behind your normal API access control, as it does place real calls.

Integration with Monitoring

Both tools integrate with Prometheus metrics:

- HLR lookups via the tool are counted in `hlr_lookups_total`
- Call simulations are counted in `call_simulations_total{call_type, source}`
- Processing times are tracked in respective duration metrics

This helps:

- Track diagnostic tool usage
- Monitor performance of diagnostic queries
- Identify heavy users of diagnostic tools

For complete metrics documentation: See metrics.md for all available metrics, query examples, and monitoring setup.

Best Practices

1. **Use Call Simulator First**
    - Before making configuration changes
    - When troubleshooting subscriber-specific issues
    - To understand call flow for training
2. **Use HLR Lookup For**
    - Quick check of roaming status
    - Verifying call forwarding from HLR
    - Testing SS7 MAP connectivity
3. **Document Findings**
    - Take screenshots of simulator results
    - Note any unexpected behavior
    - Share results with team for analysis
4. **Compare to Logs**
    - Run simulation with same parameters as failed call
    - Compare simulator variables to actual call logs
    - Identify discrepancies
5. **Regular Testing**
    - Weekly spot checks with simulator
    - Test each call type (MT/MO/Emergency)
    - Verify OCS and HLR integration

Troubleshooting the Tools

HLR Lookup Issues

Tool shows "SS7 MAP is disabled"

- Check `config/runtime.exs` for `ss7_map.enabled`
- Restart application after config change

Tool shows timeout errors

- Check SS7 MAP gateway is reachable
- Check network connectivity to HLR
- Check `ss7_map.timeout_ms` in configuration

Tool shows "No VLR Number"

- Subscriber is offline or doesn't exist in HLR
- Normal for subscribers who are powered off
- Normal for non-existent numbers

Call Simulator Issues

Simulator shows "No Sh data"

- Subscriber not provisioned in HSS
- HSS is unreachable
- Check `diameter.sh_application` configuration

Simulator shows "Source IP is not authorized"

- IP not in `allowed_sbc_source_ips` or `allowed_cscf_ips`
- Use "Force Disposition" to override IP-based auth

Simulator shows "Missing required parameters"

- All fields are required except options
- Enter valid phone numbers

- Enter valid IP address

Simulator takes too long

- Uncheck "Skip OCS Authorization" if not testing OCS
- Uncheck "Skip HLR Lookup" if not testing HLR
- Check actual system performance (Sh/HLR/OCS response times)

Support

For issues with these tools:

1. Check application logs for errors
2. Verify configuration (Sh, HLR, OCS)
3. Test connectivity to external systems
4. Contact support team with screenshots and error messages

IMS Conference Server - User Guide

Overview

The IMS Conference Server provides multi-party conferencing capabilities compliant with the 3GPP IMS Conference Framework (RFC 4579, RFC 4575, TS 24.147). It enables subscribers to create and manage audio/video conferences through the IMS Application Server.

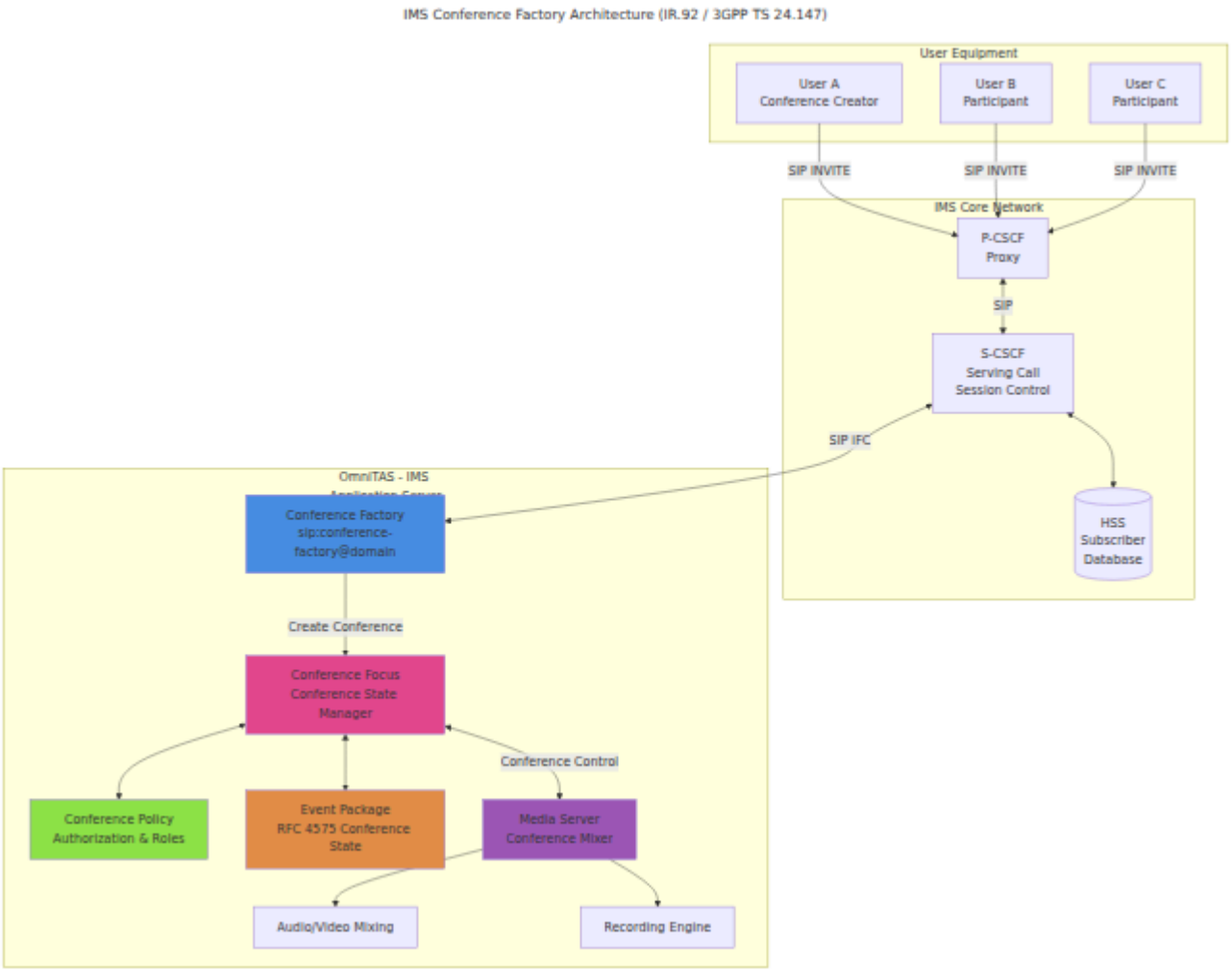
Architecture

The IMS Conference Server is an integrated component of OmniTAS that provides:

- **Conference Factory URI**: SIP URI for creating new conferences
- **Conference Focus**: Manages conference state and participants
- **Conference Policy Control**: Enforces participant roles and permissions
- **Media Mixing**: Handles audio/video mixing for conference participants

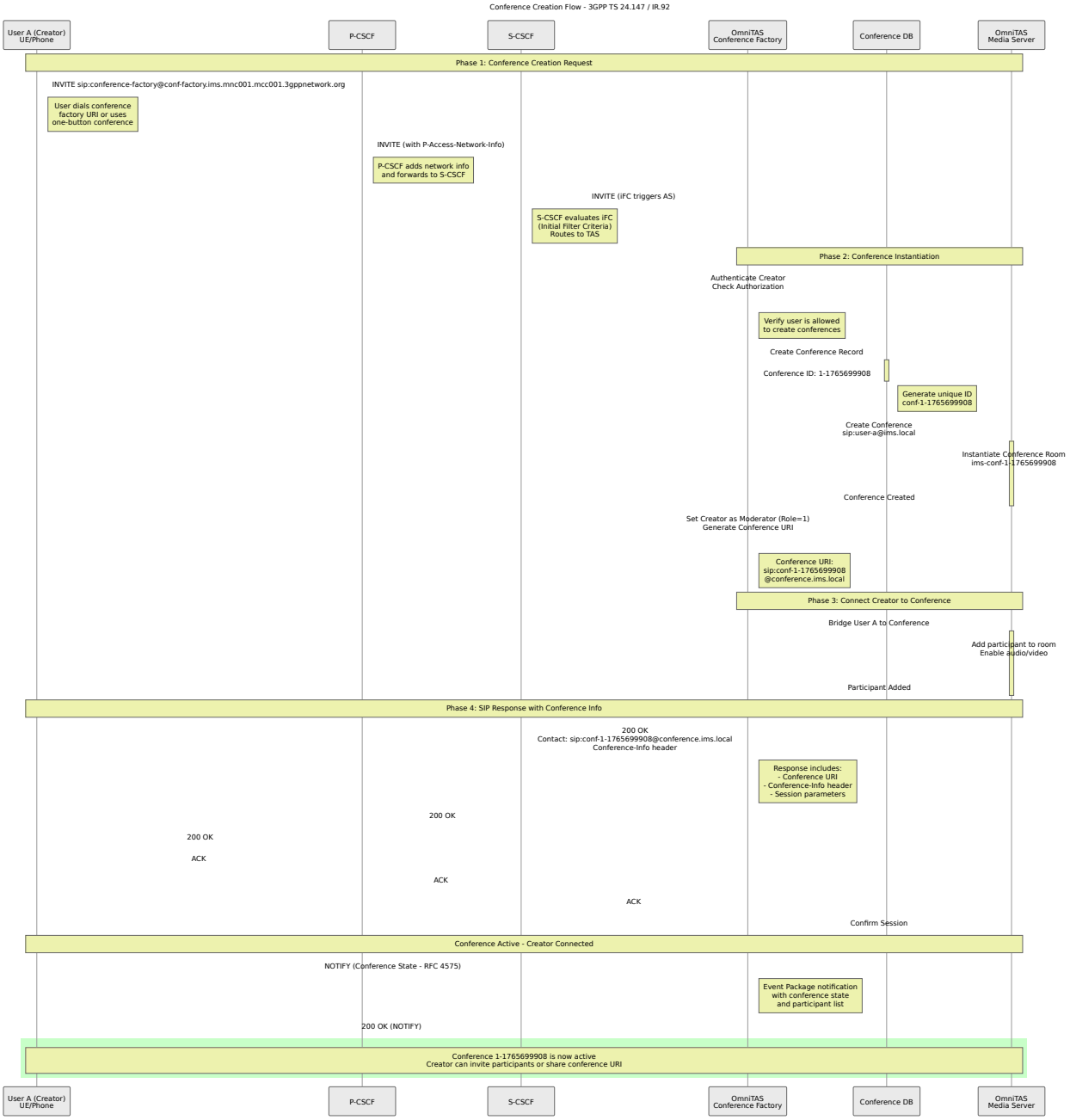
IMS Conference Factory Architecture

The TAS implements the 3GPP Conference Factory pattern as defined in TS 24.147 and RFC 4579:



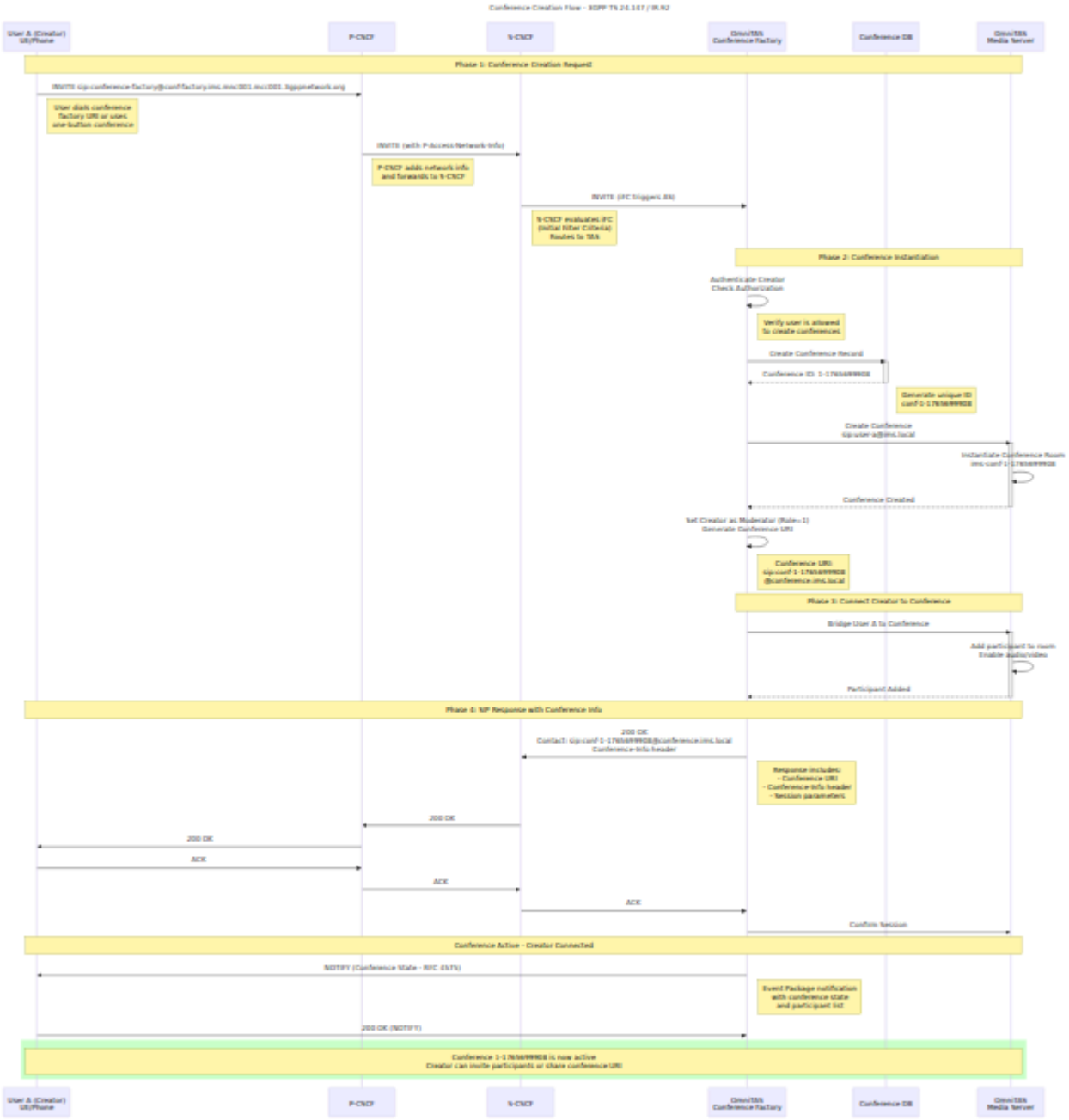
Conference Creation Flow (RFC 4579 Factory Pattern)

This diagram shows how a user creates a new conference through the Conference Factory URI:



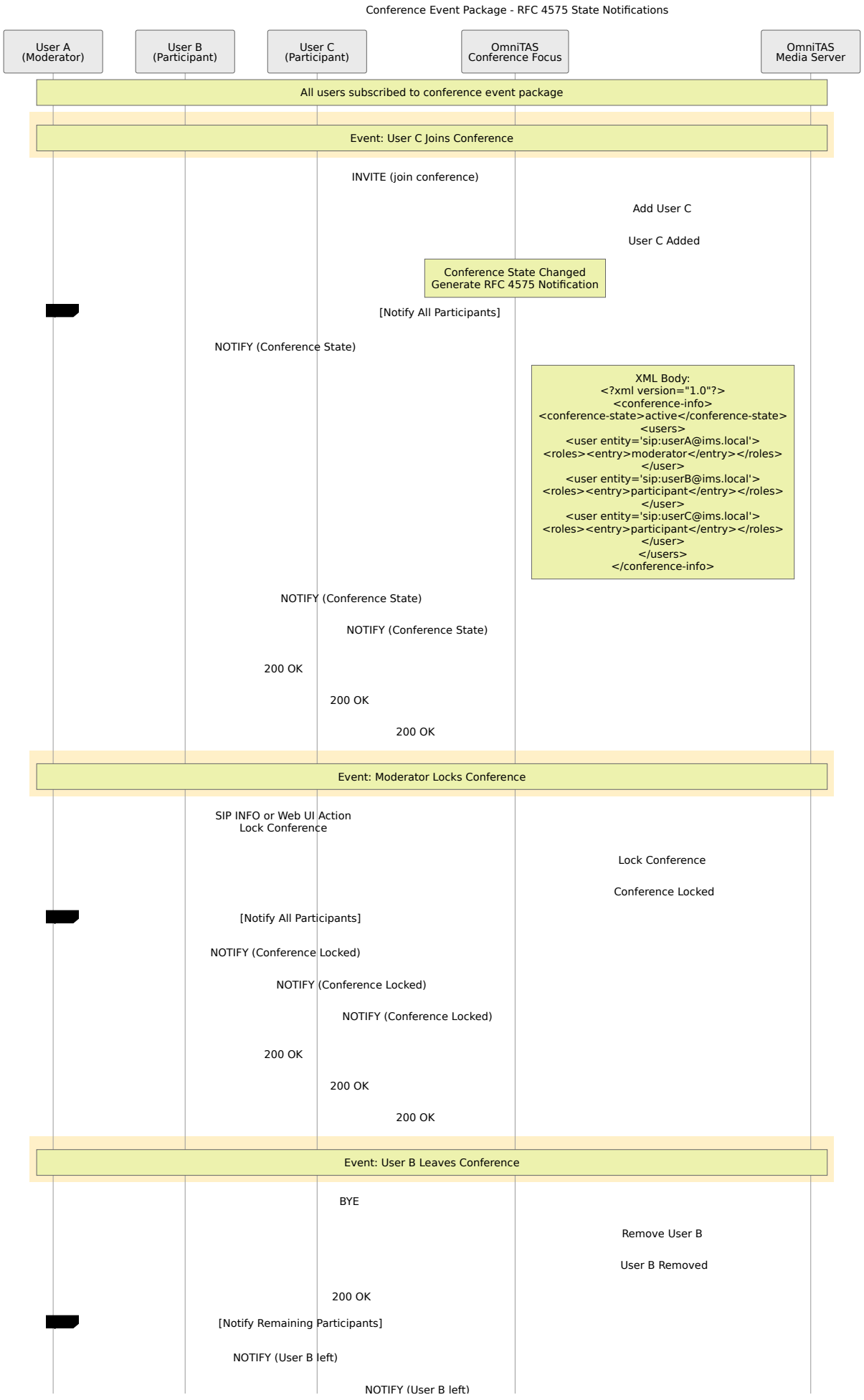
Participant Join Flow

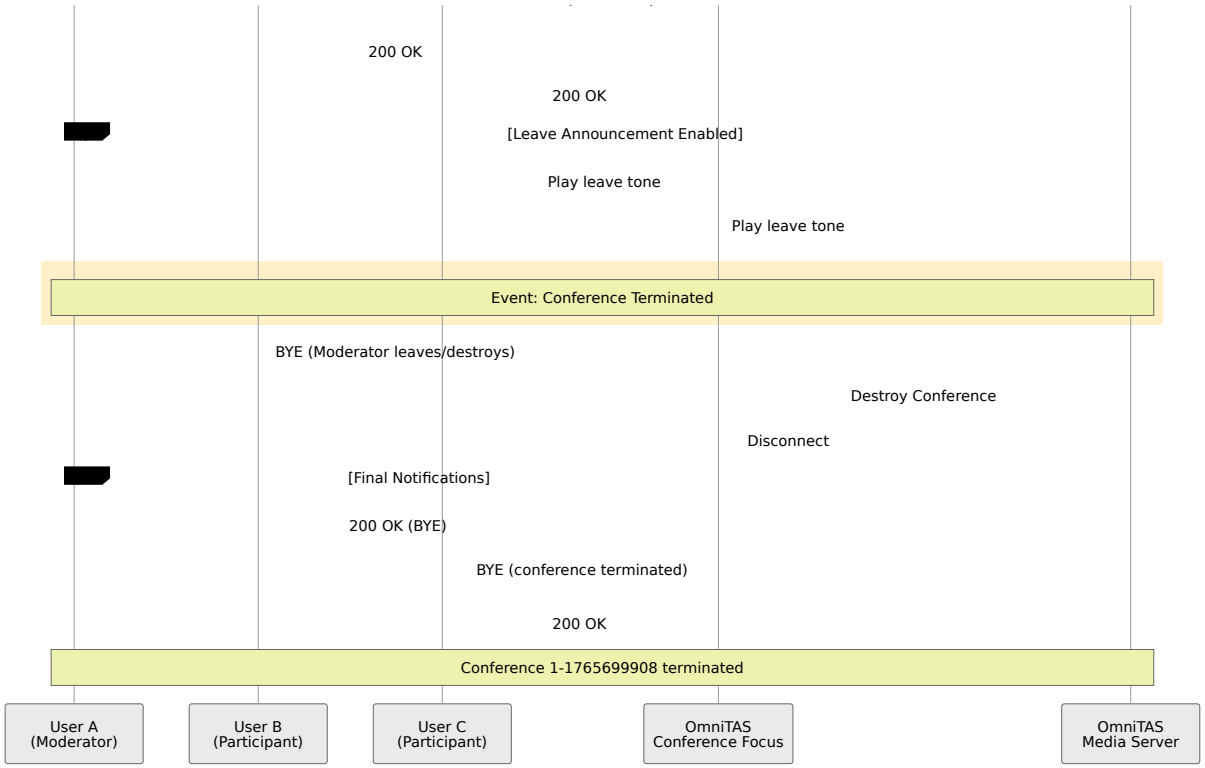
This diagram shows how additional participants join an existing conference:



Conference Event Package (RFC 4575)

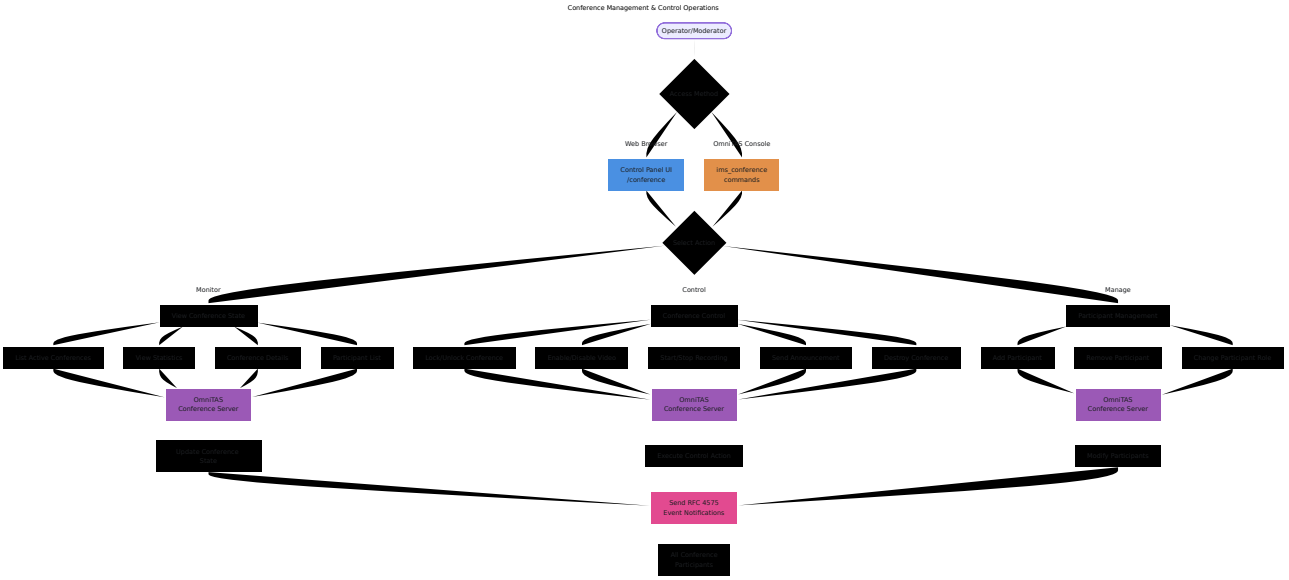
The conference server sends conference state notifications to all participants:


Conference Event Package - RFC 4575 State Notifications
User A (Moderator)
User B (Participant)
User C (Participant)
OmniTAS Conference Focus
OmniTAS Media Server
All users subscribed to conference event package
Event: User C Joins Conference
INVITE (join conference)
Add User C
User C Added
Conference State Changed
Generate RFC 4575 Notification
[Notify All Participants]
NOTIFY (Conference State)
XML Body:
<?xml version="1.0"?>
<conference-info>
<conference-state>active</conference-state>
<users>
<user entity='sip:userA@ims.local'>
<roles><entry>moderator</entry></roles>
</user>
<user entity='sip:userB@ims.local'>
<roles><entry>participant</entry></roles>
</user>
<user entity='sip:userC@ims.local'>
<roles><entry>participant</entry></roles>
</user>
</users>
</conference-info>
NOTIFY (Conference State)
NOTIFY (Conference State)
200 OK
200 OK
200 OK
Event: Moderator Locks Conference
SIP INFO or Web UI Action
Lock Conference
Lock Conference
Conference Locked
[Notify All Participants]
NOTIFY (Conference Locked)
NOTIFY (Conference Locked)
NOTIFY (Conference Locked)
200 OK
200 OK
200 OK
Event: User B Leaves Conference
BYE
Remove User B
User B Removed
200 OK
[Notify Remaining Participants]
NOTIFY (User B left)
NOTIFY (User B left)



Conference Management Operations

Operations performed through Web UI or OmniTAS Console:



Access

Web Interface

Navigate to `/conference` or click "Conference" in the navigation menu to access the Conference Management interface.

OmniTAS Console

Access the conference server from the OmniTAS console using the `ims_conference` command.

Features

Conference Management Interface

The web interface provides real-time monitoring and management of active IMS conferences:

Statistics Dashboard

Displays high-level conference server statistics:

- **Active Conferences**: Total number of ongoing conferences
- **Total Participants**: Combined participant count across all conferences
- **Video Conferences**: Number of conferences with video enabled
- **Locked Conferences**: Number of conferences locked to new participants

The dashboard also shows server configuration:

- **Domain**: Conference server domain (e.g., conference.ims.local)
- **Factory URI**: SIP URI for conference creation requests
- **MNC/MCC**: Mobile Network Code and Country Code
- **Access Network**: Network type (e.g., 3GPP-E-UTRAN-FDD)
- **Default Max Participants**: Maximum participants per conference

- **Video by Default**: Whether video is enabled by default
- **Recording Enabled**: Whether conference recording is available

Conference List

Shows all active conferences with:

- **Conference ID**: Unique identifier for the conference
- **URI**: SIP URI of the conference
- **Participants**: Current number of participants
- **Creator**: Phone number/URI of the conference creator

Click on any conference to expand and view detailed information.

Conference Details

Expanding a conference shows:

Conference Information:

- ID and URI
- Room name
- Creator identity
- Conference state
- Participant count (current/max and minimum)
- Video status (Enabled/Disabled)
- Lock status (Locked/Unlocked)
- Recording status (Active/Inactive)

Participant List:

- SIP URI of each participant
- Session UUID
- Participant state
- Role (0 = participant, 1 = moderator)
- Video status

Conference Actions:

- Lock/Unlock conference
- Enable/Disable video
- (Additional actions available via CLI)

Auto-Refresh

The interface automatically refreshes every 5 seconds to show real-time conference status. You can toggle auto-refresh on/off or manually refresh using the "Refresh" button.

OmniTAS Console Commands

All conference management operations are available through the `ims_conference` command in the OmniTAS console.

Command Syntax

```
ims_conference <command> [arguments]
```

Available Commands

list

Lists all active IMS conferences.

```
omnitas@server> ims_conference list
IMS Conferences:
Conference ID        Conference URI                          Partici
Creator
=============================================================================
1-1765699908         sip:conf-1-1765699908@conference.ims.local 3
12025550151

Total: 1 conferences
```

info

Shows detailed information about a specific conference.

Syntax: `ims_conference info <conf_id>`

Important: Use the Conference ID (e.g., `1-1765699908`), not the conference name with prefix.

```
omnitas@server> ims_conference info 1-1765699908
Conference Information:
  ID: 1-1765699908
  URI: sip:conf-1-1765699908@conference.ims.local
  Room: ims-conf-1-1765699908
  Creator: 12025550151
  State: 1
  Participants: 3/10 (min: 2)
  Video: Enabled
  Locked: No
  Recording: Inactive

Participants:
  - sip:1235;phone-
context=ims.mnc001.mcc001.3gppnetwork.org@ims.mnc001.mcc001.3gppnetwo
(342d50e0-9f67-4cc5-9179-4acae6f65f34)
      State: 3, Role: 0, Video: On
  - sip:1235;phone-
context=ims.mnc001.mcc001.3gppnetwork.org@ims.mnc001.mcc001.3gppnetwo
(bd98ca37-64fd-4618-b2db-aaba108c73e2)
      State: 3, Role: 0, Video: On
  - 12025550151 (6270da85-9b94-4285-8130-8769b11d0aa2)
      State: 3, Role: 1, Video: On
```

stats

Displays overall conference server statistics and configuration.

```
omnitas@server> ims_conference stats
IMS Conference Server Statistics:
================================
Active conferences: 1
Total participants: 3
Video conferences: 1
Locked conferences: 0

Configuration:
  Domain: conference.ims.local
  Factory URI: sip:conference-factory@conf-
factory.ims.mnc001.mcc001.3gppnetwork.org
  MNC/MCC: 001/001
  Access Network: 3GPP-E-UTRAN-FDD
  Default max participants: 10
  Allow anonymous: Yes
  Video by default: Yes
  Recording enabled: Yes
  Announcements: Join=On, Leave=On, Count=On
```

create

Creates a new conference.

Syntax: `ims_conference create <creator_uri>`

```
omnitas@server> ims_conference create sip:12025550151@ims.local
Conference created: 1-1765699909
Conference URI: sip:conf-1-1765699909@conference.ims.local
```

destroy

Terminates a conference and disconnects all participants.

Syntax: `ims_conference destroy <conf_id>`

```
omnitas@server> ims_conference destroy 1-1765699908
Conference 1-1765699908 destroyed
```

add

Adds a participant to an existing conference.

Syntax: `ims_conference add <conf_id> <sip_uri>`

```
omnitas@server> ims_conference add 1-1765699908
sip:12025550152@ims.local
Adding participant sip:12025550152@ims.local to conference 1-
1765699908
```

remove

Removes a participant from a conference.

Syntax: `ims_conference remove <conf_id> <uuid>`

Note: Use the participant's session UUID from the `info` command output.

```
omnitas@server> ims_conference remove 1-1765699908 342d50e0-9f67-
4cc5-9179-4acae6f65f34
Removed participant from conference 1-1765699908
```

lock

Locks a conference to prevent new participants from joining.

Syntax: `ims_conference lock <conf_id>`

```
omnitas@server> ims_conference lock 1-1765699908
Conference 1-1765699908 locked
```

unlock

Unlocks a conference to allow new participants.

Syntax: `ims_conference unlock <conf_id>`

```
omnitas@server> ims_conference unlock 1-1765699908
Conference 1-1765699908 unlocked
```

video

Controls video for a conference.

Syntax: `ims_conference video <conf_id> on|off`

```
omnitas@server> ims_conference video 1-1765699908 on
Video enabled for conference 1-1765699908

omnitas@server> ims_conference video 1-1765699908 off
Video disabled for conference 1-1765699908
```

record

Controls conference recording.

Syntax: `ims_conference record <conf_id> start|stop`

```
omnitas@server> ims_conference record 1-1765699908 start
Recording started for conference 1-1765699908

omnitas@server> ims_conference record 1-1765699908 stop
Recording stopped for conference 1-1765699908
```

announce

Plays an announcement to all conference participants.

Syntax: `ims_conference announce <conf_id> <message>`

```
omnitas@server> ims_conference announce 1-1765699908 "This
conference will end in 5 minutes"
Announcement sent to conference 1-1765699908
```

subscribers

Lists all subscribers currently in a conference (alternative view to `info`).

Syntax: `ims_conference subscribers <conf_id>`

```
omnitas@server> ims_conference subscribers 1-1765699908
Subscribers in conference 1-1765699908:
  - sip:1235;phone-
context=ims.mnc001.mcc001.3gppnetwork.org@ims.mnc001.mcc001.3gppnetwo
  - 12025550151
```

Conference States

Conferences and participants have numeric state values:

Conference States

- **0**: Initializing
- **1**: Active
- **2**: Terminating
- **3**: Terminated

Participant States

- **0**: Invited
- **1**: Dialing
- **2**: Alerting
- **3**: Connected
- **4**: Disconnecting
- **5**: Disconnected

Participant Roles

- **0**: Regular participant
- **1**: Moderator/Creator

Use Cases

Monitoring Active Conferences

Scenario: Operations team needs to see how many conferences are active

Steps:

1. Open Conference Management interface (`/conference`)
2. View the Statistics Dashboard for high-level metrics
3. Review the conference list for specific conferences
4. Use auto-refresh to monitor in real-time

CLI Alternative:

```
omnitas@server> ims_conference stats
omnitas@server> ims_conference list
```

Troubleshooting Conference Issues

Scenario: User reports they cannot join a conference

Steps:

1. Get the conference ID from the user
2. Run `ims_conference info <conf_id>` to check conference state
3. Check if conference is locked (Locked: Yes)
4. Check current participant count vs. maximum
5. Review participant list for any connection issues
6. Check OmniTAS logs for SIP invite failures

Common Issues:

- Conference locked: `ims_conference unlock <conf_id>`
- Maximum participants reached: Check default_max_participants config
- Network issues: Check SIP connectivity and firewall rules

Managing Conference Bandwidth

Scenario: Need to reduce bandwidth usage during network congestion

Steps:

1. Identify conferences with video enabled
2. For non-critical conferences, disable video:

```
ims_conference video <conf_id> off
```

3. Monitor bandwidth usage
4. Re-enable video when congestion clears

Handling Disruptive Participants

Scenario: A participant is being disruptive in a conference

Steps:

1. Get the conference ID and participant's session UUID
2. Remove the participant:

```
ims_conference remove <conf_id> <participant_uuid>
```

3. Lock the conference to prevent them from rejoining:

```
ims_conference lock <conf_id>
```

4. Add legitimate participants manually if needed:

```
ims_conference add <conf_id> <sip_uri>
```

Recording Important Conferences

Scenario: Need to record a conference for compliance or documentation

Steps:

1. Identify the conference ID
2. Start recording:

```
ims_conference record <conf_id> start
```

3. Monitor that recording is active (Recording: Active in `info` output)
4. Stop recording when complete:

```
ims_conference record <conf_id> stop
```

5. Recording files are stored in the OmniTAS recordings directory

Emergency Conference Termination

Scenario: Need to immediately terminate a conference

Steps:

1. Optionally announce to participants:

```
ims_conference announce <conf_id> "This conference is being
terminated"
```

2. Wait a few seconds for announcement to play
3. Destroy the conference:

```
ims_conference destroy <conf_id>
```

4. All participants will be disconnected immediately

Integration with IMS Network

Conference Creation Flow

1. Subscriber sends SIP INVITE to conference factory URI
2. IMS Application Server receives request

3. Conference Server creates new conference instance
4. Conference ID and URI are generated
5. Conference policy is initialized based on creator
6. Creator is added as first participant with moderator role
7. Conference URI is returned to creator
8. Other participants can now join via the conference URI

Participant Roles

Moderator (Role: 1)

- Can lock/unlock conference
- Can remove other participants
- Can control video settings
- Receives conference notifications

Participant (Role: 0)

- Can join/leave conference
- Can speak and listen
- Can enable/disable own video
- Subject to conference policies

3GPP Compliance

The IMS Conference Server implements key 3GPP specifications:

- **TS 24.147**: Conferencing using IP Multimedia (IM) Core Network (CN) subsystem
- **RFC 4579**: Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents
- **RFC 4575**: A Session Initiation Protocol (SIP) Event Package for Conference State
- **RFC 5239**: A Framework for Centralized Conferencing

Network Elements Integration

- **P-CSCF**: Handles initial SIP signaling from UE
- **S-CSCF**: Routes conference requests to Application Server
- **OmniTAS**: Hosts the Conference Server functionality and provides media mixing
- **HSS**: Provides subscriber authentication and authorization

Configuration

Conference server configuration is managed through OmniTAS configuration files:

Key Parameters:

- `domain`: Conference server domain
- `factory_uri`: SIP URI for conference creation
- `mnc_mcc`: Mobile network identifiers
- `access_network`: Network access type
- `default_max_participants`: Default maximum participants per conference
- `allow_anonymous`: Whether to allow anonymous participants
- `video_by_default`: Default video setting for new conferences
- `recording_enabled`: Whether recording feature is available
- `announce_join`: Play tone when participant joins
- `announce_leave`: Play tone when participant leaves
- `announce_count`: Announce participant count

Best Practices

Capacity Planning

- Monitor active conference count and participant counts
- Plan for peak usage (e.g., business hours)
- Allocate sufficient CPU/memory for media mixing

- Consider video vs. audio-only for bandwidth management

Security

- Ensure conference URIs are not easily guessable
- Use conference locking for private conferences
- Monitor for unauthorized access attempts
- Implement maximum participant limits
- Review conference recordings access controls

Operational Monitoring

- Set up alerts for conference server errors
- Monitor conference creation/destruction rates
- Track average conference duration
- Review participant connection failures
- Monitor media quality metrics

For detailed metrics documentation: See metrics.md for:

- RTP/RTCP media quality metrics (Port 9093)
- Active call and session metrics (Port 9090)
- System and Erlang VM metrics (Port 8080)
- Prometheus query examples

Troubleshooting

- Check OmniTAS logs for conference-related errors
- Verify SIP connectivity between participants and conference server
- Monitor RTP media streams for packet loss
- Verify network bandwidth availability
- Check participant device compatibility

Limitations

- Maximum participants per conference: Configurable (default: 10)
- Maximum concurrent conferences: Limited by server resources
- Video quality: Depends on network bandwidth and participant devices
- Recording format: Determined by OmniTAS configuration
- Conference ID format: Auto-generated, cannot be customized via web interface

Support

For issues or questions about the IMS Conference Server:

1. Check OmniTAS logs for error messages
2. Verify conference server configuration
3. Review network connectivity and firewall rules
4. Contact Omnitouch support with conference ID and timestamps

ANSSI R226 Interception Compliance Documentation

Document Purpose: This document provides technical specifications required for ANSSI R226 authorization under Articles R226-3 and R226-7 of the French Penal Code for the OmniTAS IMS Application Server.

Classification: Regulatory Compliance Documentation

Target Authority: Agence nationale de la sécurité des systèmes d'information (ANSSI)

Regulation: R226 - Protection of Correspondence Privacy and Lawful Interception

---

1. DETAILED TECHNICAL SPECIFICATIONS

1.1 Commercial Technical Datasheet

Product Name: OmniTAS IMS Application Server **Product Type:** Telecommunications Application Server (TAS) **Primary Function:** IMS (IP Multimedia Subsystem) call processing and session management **Network Protocols:** SIP, Diameter, HTTP/HTTPS, SS7/MAP **Deployment Model:** On-premises server application

Core Capabilities

Call Processing:

- Session Initiation Protocol (SIP) proxy and B2BUA functionality
- IMS Initial Filter Criteria (iFC) processing
- Session routing and call control
- Emergency call handling (E.164 PSAP routing)
- Call Detail Record (CDR) generation

Network Interfaces:

- **Northbound:** IMS S-CSCF interface (SIP over TCP/UDP)
- **Southbound:** SBC/Gateway interface (SIP trunking)
- **Diameter:** Sh (subscriber data), Ro (online charging)
- **SS7:** MAP gateway interface for HLR/MSC interworking
- **HTTP/HTTPS:** External service integration (SMS, TTS, MAP gateway)

Storage and Processing:

- Real-time session state management
- CDR storage and retrieval
- Subscriber registration database (Sofia SIP)
- Configuration database (SQLite)

1.2 Interception Capabilities

1.2.1 Signal Acquisition

SIP Signaling Capture:

- The OmniTAS processes all SIP signaling messages between IMS subscribers and external networks
- Full access to SIP headers including:
  - Calling party identification (From, P-Asserted-Identity)
  - Called party identification (To, Request-URI)
  - Contact URIs and network location
  - Call routing information
  - Session description (SDP) including media codecs and endpoints

Call Metadata Acquisition:

- Complete Call Detail Records (CDR) stored in database with:
  - Timestamp (start, answer, end times)
  - Caller and callee identifiers (MSISDN, IMSI, SIP URI)
  - Call direction (mobile originating/terminating)
  - Call result (answered, busy, failed, etc.)
  - Duration and charging information
  - Network location data (cell tower information when available)

Session Recording Interface (SIPREC):

- SIPREC protocol support for lawful interception
- Capability to replicate SIP signaling to external recording servers
- Configurable session recording policies
- **Licensing Control:** SIPREC functionality requires explicit licensing authorization
- **Access Control:** SIPREC configuration restricted to authorized administrators

1.2.2 Media Processing Capabilities

Media Plane:

- B2BUA with RTP media relay capabilities
- RTP streams pass through the server
- Access to media flows for interception purposes
- SDP parsing for media endpoint and codec information

Signaling Plane:

- SIP message parsing and analysis
- Diameter message encoding/decoding (Sh, Ro interfaces)
- HTTP/HTTPS request/response processing

1.2.3 Analysis Capabilities

Real-Time Call Monitoring:

- Web UI dashboard showing active calls with:
  - Call state (trying, ringing, active, terminated)
  - Caller/callee information
  - Call duration
  - Media codec information
  - Network endpoints

Historical Analysis:

- CDR database queryable by:
  - Time range
  - Calling/called party number
  - Call type (voice, emergency, etc.)
  - Call result/disposition
  - Duration thresholds

Subscriber Tracking:

- Active registration monitoring
- Subscriber location tracking via:
  - IMS registration contact URI
  - P-Access-Network-Info header (cell tower identification)
  - IP address and port information
- Historical registration records

Network Analytics:

- Call volume metrics (Prometheus integration)
- Gateway status and connectivity
- Diameter peer connectivity
- System performance metrics

For comprehensive metrics documentation: See metrics.md for detailed monitoring, alerting, and observability configuration.

Location Intelligence:

- Cell tower database integration
- E.164 number to geographic location mapping (North American Numbering Plan)
- Emergency services routing (PSAP mapping)

1.3 Countermeasure Capabilities

1.3.1 Privacy Protection Mechanisms

Communication Confidentiality:

- Diameter TLS transport security
- HTTPS for web interfaces and APIs
- Database encryption at rest (configurable)

Access Control:

- Role-based access control (RBAC) for web UI
- Password hashing with SHA-512 and salt (65,532 iterations)

Audit Logging:

- Complete audit trail of administrative actions
- Configuration change logging
- Authentication event logging
- Tamper-evident log storage

1.3.2 Anti-Interception Features

Secure Communications:

- Mandatory TLS for external interfaces (configurable)
- Certificate-based authentication
- Perfect Forward Secrecy (PFS) cipher suites

Data Protection:

- Automatic CDR retention policies

- Secure data deletion capabilities
- Database access controls
- Network segmentation support (separate management/signaling/media networks)

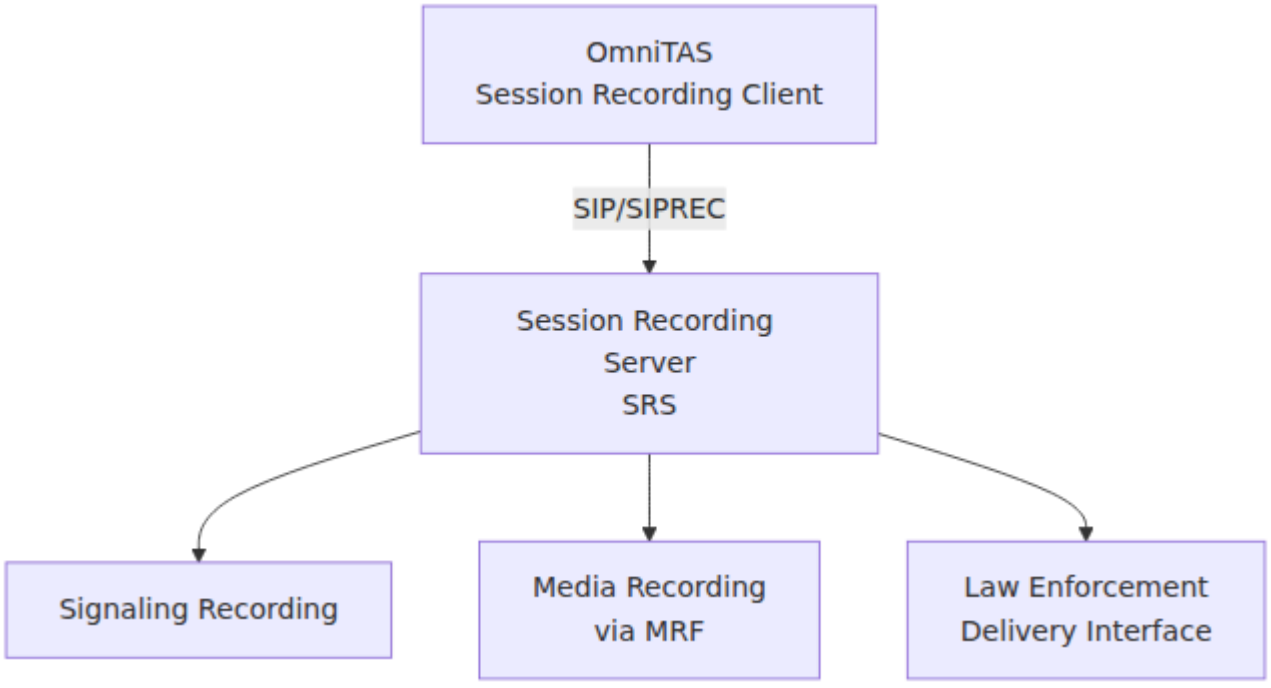
System Hardening:

- Boot parameter protection
- Integrity verification mechanisms
- Minimal attack surface (only required services enabled)

1.4 Technical Architecture for Lawful Interception

Lawful Interception Integration Points

1. SIPREC Interface (Session Recording Protocol - RFC 7866):



2. CDR Export Interface:

- CDR export to external systems
- Standard formats (CSV, JSON)
- Secure transfer (HTTPS)

3. Direct Database Access:

- Read-only database credentials for authorized systems
- SQL query access to CDR tables
- Subscriber registration data access
- Audit log access

4. API Integration:

- RESTful API for call monitoring
- Real-time active call queries
- Historical CDR retrieval
- Subscriber registration status

Interception Triggering Mechanisms

Target-Based Interception:

- Subscriber identifier matching (MSISDN, IMSI, SIP URI)
- Configurable interception rules in application logic
- SIPREC session forking based on caller/callee identity

Event-Based Interception:

- Emergency call detection and recording
- Specific destination number monitoring
- Geographic area-based triggering (cell tower location)

Time-Based Interception:

- Scheduled recording windows
- Retention period enforcement
- Automatic expiration of interception warrants

---

2. ENCRYPTION AND CRYPTANALYSIS CAPABILITIES

2.1 Cryptographic Capabilities Overview

The OmniTAS IMS Application Server implements cryptographic mechanisms for securing communications and protecting sensitive data. This section documents all cryptographic capabilities in accordance with ANSSI requirements.

2.2 Transport Layer Encryption

2.2.1 TLS/SSL Implementation

Supported Protocols:

- TLS 1.2 (RFC 5246)
- TLS 1.3 (RFC 8446)
- SSL 2.0/3.0: DISABLED (known vulnerabilities)
- TLS 1.0/1.1: DEPRECATED (configurable, disabled by default)

Cipher Suites (Configurable Priority List):

Preferred - TLS 1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256

Supported - TLS 1.2:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

Security Features:

- Perfect Forward Secrecy (PFS) required
- Strong Diffie-Hellman groups (2048-bit minimum)
- Elliptic Curve Cryptography: NIST P-256, P-384, P-521
- Server Name Indication (SNI) support
- OCSP stapling for certificate validation

Certificate Management:

- X.509 certificate support
- RSA key sizes: 2048-bit minimum, 4096-bit recommended
- ECDSA support (P-256, P-384)
- Certificate chain validation
- CRL and OCSP revocation checking
- Self-signed certificates (development only)
- External CA integration

Applications:

- HTTPS for web UI and API access
- Diameter over TLS

2.3 Data Encryption at Rest

2.3.1 Database Encryption

SQLite Encryption:

- SQLCipher integration support
- AES-256 encryption
- Encrypted storage for sensitive data (CDR, subscriber data)

2.3.2 File System Encryption

Sensitive Data Storage:

- CDR files: AES-256 encryption (optional)

- Configuration files: Encrypted storage for credentials
- Private keys: Encrypted keystores (PKCS#12, PEM with passphrase)
- Log files: Encryption support for archived logs

Key Storage:

- File-based keystores with passphrase protection
- Secure key rotation mechanisms

2.4 Authentication and Password Cryptography

2.4.1 Password Hashing

Algorithm: SHA-512 with salt **Configuration:**

- Randomly generated salt (128-bit minimum)
- 65,532 iteration rounds (configurable)
- Salt stored alongside hash
- Resistant to rainbow table attacks

Storage Format:

```
$6$rounds=65532$<salt>$<hash>
```

Applications:

- Web UI user authentication
- API token generation
- Administrator password storage
- Database user credentials

2.4.2 SSH Key Authentication

Supported Key Types:

- RSA: 1024-4096 bits (2048-bit minimum recommended)
- DSA: 1024-4096 bits (deprecated, RSA preferred)

- ECDSA: P-256, P-384, P-521 curves
- Ed25519: 256-bit (preferred for new deployments)

Key Management:

- External key generation support
- Public key import for client authentication
- Server host key management
- Individual key revocation
- Key rotation procedures

SSH Protocol:

- SSH-2 protocol only (SSH-1 disabled)
- Strong MAC algorithms (HMAC-SHA2-256, HMAC-SHA2-512)
- Key exchange: curve25519-sha256, ecdh-sha2-nistp256, diffie-hellman-group14-sha256

2.5 Diameter Protocol Security

2.5.1 Diameter Security Mechanisms

Transport Security:

- TLS over TCP for Diameter peer connections
- Mutual certificate authentication

Application-Level Security:

- Peer authentication via Origin-Host/Origin-Realm validation
- Shared secret configuration (legacy, deprecated)
- AVP (Attribute-Value Pair) encryption for sensitive data
- End-to-end security with CMS (Cryptographic Message Syntax)

2.6 SIP Identity Mechanisms

P-Asserted-Identity:

- Trusted network assertion
- Identity validation and translation
- Privacy header support

Note: Subscriber authentication is performed by the IMS Core (P-CSCF/S-CSCF), not by the TAS.

2.7 Cryptanalysis and Security Assessment Capabilities

2.7.1 Protocol Analysis Tools

Built-in Debugging Capabilities:

- SIP message tracing with full header/body capture
- Diameter message logging (AVP decoding)
- TLS handshake debugging
- Certificate chain validation logging

External Integration:

- Wireshark/tcpdump packet capture support
- SSLKEYLOGFILE export for TLS decryption (development only)
- PCAP export for offline analysis

2.7.2 Vulnerability Assessment Considerations

Known Cryptographic Weaknesses:

- Legacy MD5 support in SIP Digest (maintained for backward compatibility)
- Configurable weak cipher suites (disabled by default)
- Self-signed certificate support (development/testing only)

Security Testing:

- Regular security audits recommended
- Penetration testing support
- Cipher suite strength validation

- Certificate expiration monitoring

2.8 Key Management Infrastructure

2.8.1 Key Generation

Internal Key Generation:

- RSA key generation: OpenSSL library (FIPS 140-2 compliant algorithms)
- Random number generation: /dev/urandom (Linux kernel CSPRNG)
- Entropy sources: Hardware RNG, system entropy pool

2.8.2 Key Storage and Protection

Private Key Storage:

- File system with restricted permissions (0600)
- Encrypted PEM format with passphrase
- Secure deletion on key rotation

Key Backup:

- Encrypted backup procedures
- Split-key recovery mechanisms
- Secure key escrow (if required by regulation)

2.8.3 Key Distribution

Certificate Distribution:

- Manual import via web UI
- Automated provisioning via API
- ACME protocol support (Let's Encrypt, future enhancement)

Symmetric Key Distribution:

- Out-of-band key exchange for Diameter peers
- Diffie-Hellman key agreement in TLS
- No cleartext key transmission

2.9 Compliance and Standards

Cryptographic Standards Compliance:

- NIST SP 800-52: TLS guidelines
- NIST SP 800-131A: Cryptographic algorithm transitions
- RFC 7525: TLS recommendations
- ETSI TS 133 310: IMS network security
- 3GPP TS 33.203: IMS access security

French Cryptography Regulations:

- Cryptographic means declaration (if applicable)
- ANSSI cryptographic product certification (if required)
- No export-restricted cryptography (all standard algorithms)

2.10 Cryptanalysis Resistance

2.10.1 Design Principles

Defense Against Cryptanalysis:

- No custom/proprietary cryptographic algorithms
- Industry-standard, peer-reviewed algorithms only
- Regular security updates for cryptographic libraries
- Deprecation of weak algorithms

2.10.2 Operational Security

Key Rotation:

- TLS certificate renewal (annually recommended)
- Session key rotation (per-session for TLS)
- Password expiration policies (configurable)

Monitoring and Detection:

- Failed authentication attempt logging

- Certificate expiration alerts
- Cipher suite negotiation logging
- Anomaly detection for encryption failures

---

3. INTERCEPTION CONTROL AND AUTHORIZATION

3.1 Access Control for Lawful Interception

Administrative Authorization:

- Lawful interception features require administrator-level privileges
- SIPREC configuration access: Super-admin role only
- CDR access: Configurable role-based permissions
- Audit logging of all interception-related actions

Legal Framework Integration:

- Interception warrant tracking (external system integration)
- Target identifier authorization lists
- Time-limited interception activation
- Automatic deactivation on warrant expiration

3.2 Data Retention and Privacy

Retention Policies:

- CDR retention: Configurable (default 90 days, regulatory requirement 1 year)
- Registration logs: Configurable retention
- Audit logs: Minimum 1 year retention
- Automatic purging of expired data

Privacy Protections:

- Minimal data collection principle
- Purpose limitation (telecommunications service provision)
- Access logging and monitoring

3.3 Handover Interfaces for Law Enforcement

Standard Lawful Interception Interfaces:

- ETSI LI (Lawful Interception) interface support (via external mediation device)
- SIPREC to LI gateway integration
- X1, X2, X3 interface support (external system)

Delivery Formats:

- IRI (Intercept Related Information): CDR metadata
- CC (Content of Communication): SIP signaling + media (via MRF)
- Structured reporting: XML, JSON formats

---

4. SYSTEM SECURITY AND INTEGRITY

4.1 Boot Security

Secure Boot Mechanisms:

- Bootparameter protection (ANSSI R226 requirement)
- Configuration integrity verification
- Tamper detection on startup
- Secure configuration loading

4.2 Network Security

Network Security:

- Minimal exposed ports (SIP, Diameter, HTTPS only)
- Port-based access control
- IP whitelisting/blacklisting

4.3 Intrusion Detection

Monitoring Capabilities:

- Failed authentication monitoring
- Unusual call pattern detection
- Anomalous Diameter traffic detection
- Security event alerting (SIEM integration)

---

5. DOCUMENTATION REFERENCES

5.1 Technical Manuals

Available documentation in the project repository:

- **README.md:** System overview, architecture, and operational features
- **doc/deployment_guide.md:** Deployment instructions (if available)
- **doc/configuration.md:** Configuration reference (if available)

5.2 Security Certifications

- **Penetration Test Reports:** [To be provided upon request]
- **Security Audit Reports:** [To be provided upon request]
- **Cryptographic Module Validation:** OpenSSL FIPS 140-2 compliance

5.3 Compliance Documentation

- **ANSSI R226 Authorization Request:** This document
- **Lawful Interception Compliance:** As required by French telecommunications regulations

6. CONTACT INFORMATION

Vendor/Operator Information:

- Company Name: Omnitouch Network Services Pty Ltd
- Address: PO BOX 296, QUINNS ROCKS WA 6030, AUSTRALIA
- Contact Person: Compliance Team
- Email: compliance@omnitouch.com.au

Technical Security Contact:

- Name: Compliance Team
- Email: compliance@omnitouch.com.au

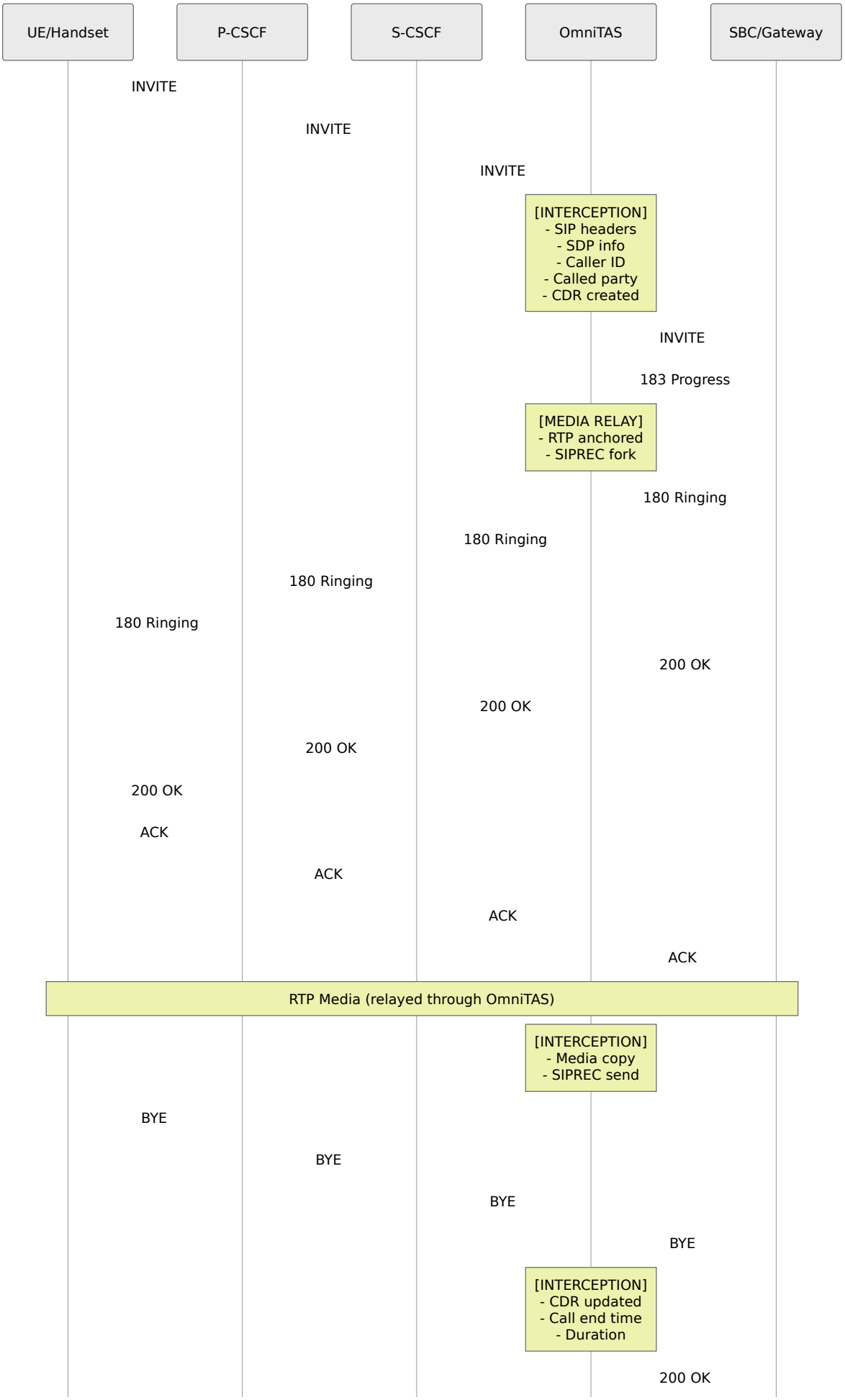
Legal/Compliance Contact:

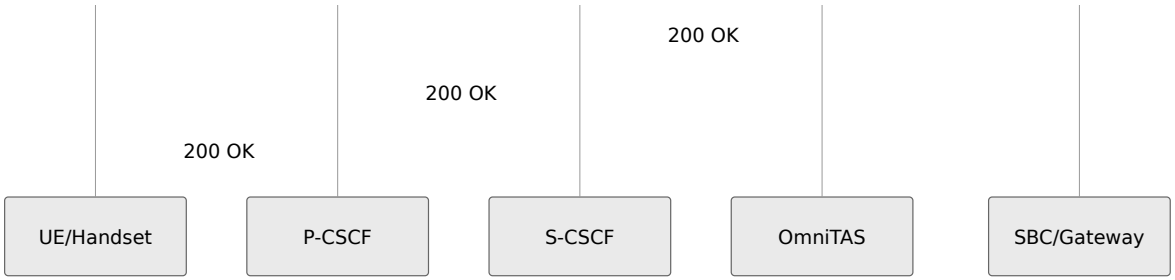
- Name: Compliance Team
- Email: compliance@omnitouch.com.au

APPENDICES

Appendix A: SIP Message Flow Examples

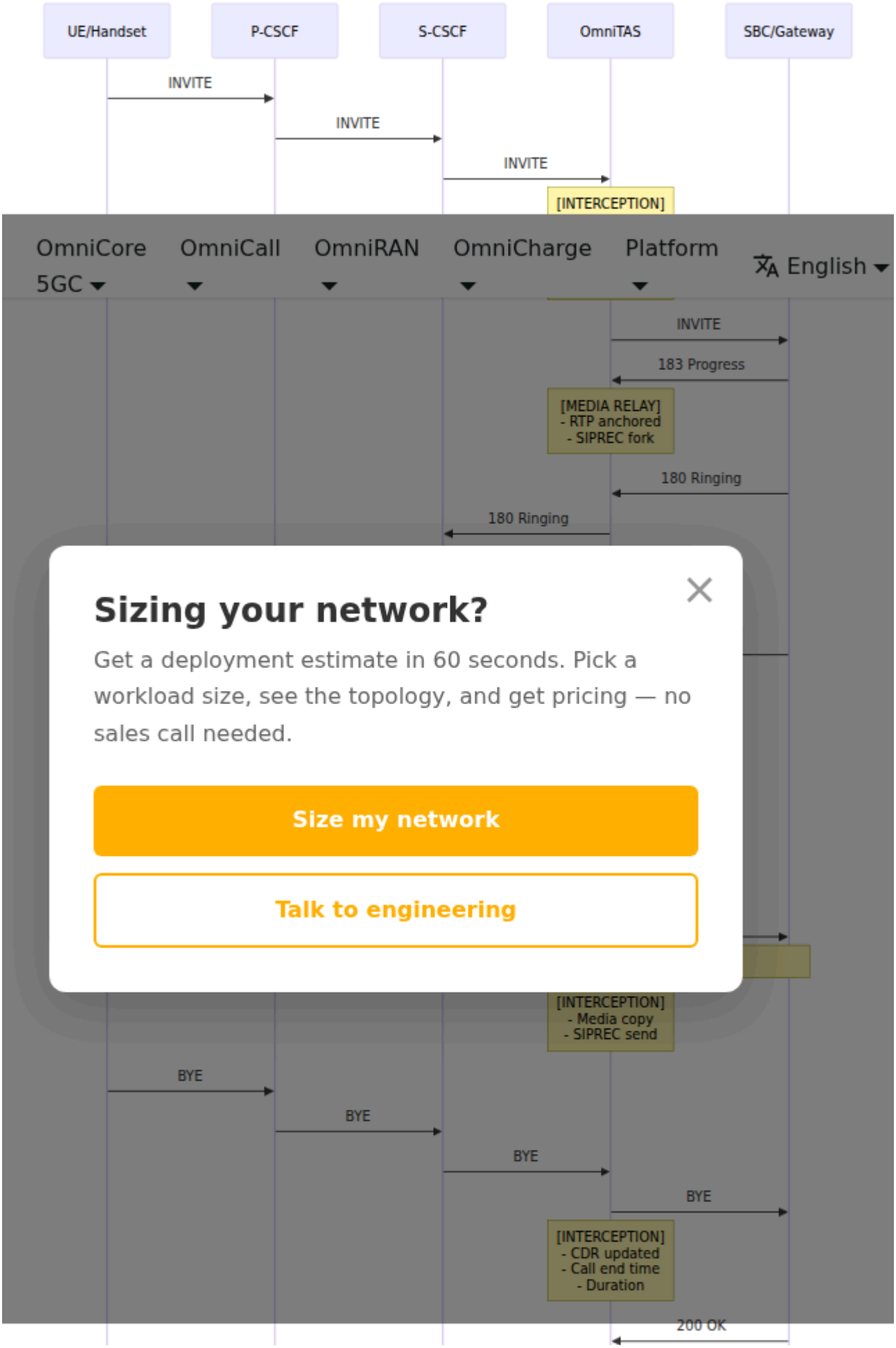
A.1 Mobile Originating Call Flow with Interception Points

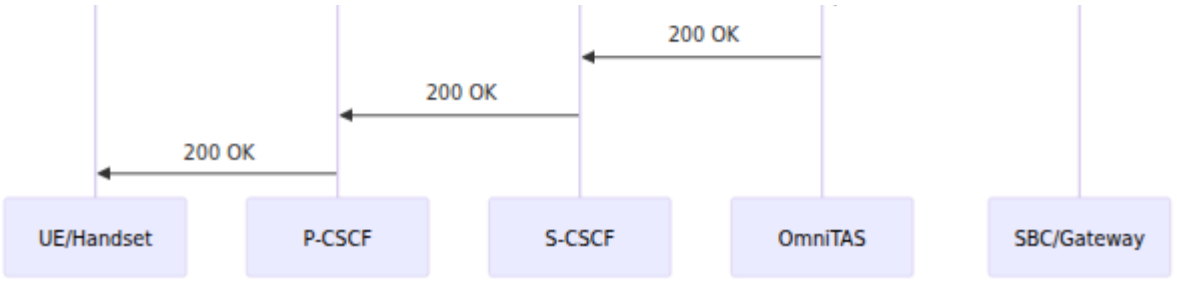

UE/Handset
P-CSCF
S-CSCF
OmniTAS
SBC/Gateway
INVITE
INVITE
INVITE
[INTERCEPTION]
- SIP headers
- SDP info
- Caller ID
- Called party
- CDR created
INVITE
183 Progress
[MEDIA RELAY]
- RTP anchored
- SIPREC fork
180 Ringing
180 Ringing
180 Ringing
180 Ringing
200 OK
200 OK
200 OK
200 OK
ACK
ACK
ACK
ACK
RTP Media (relayed through OmniTAS)
[INTERCEPTION]
- Media copy
- SIPREC send
BYE
BYE
BYE
BYE
[INTERCEPTION]
- CDR updated
- Call end time
- Duration
200 OK



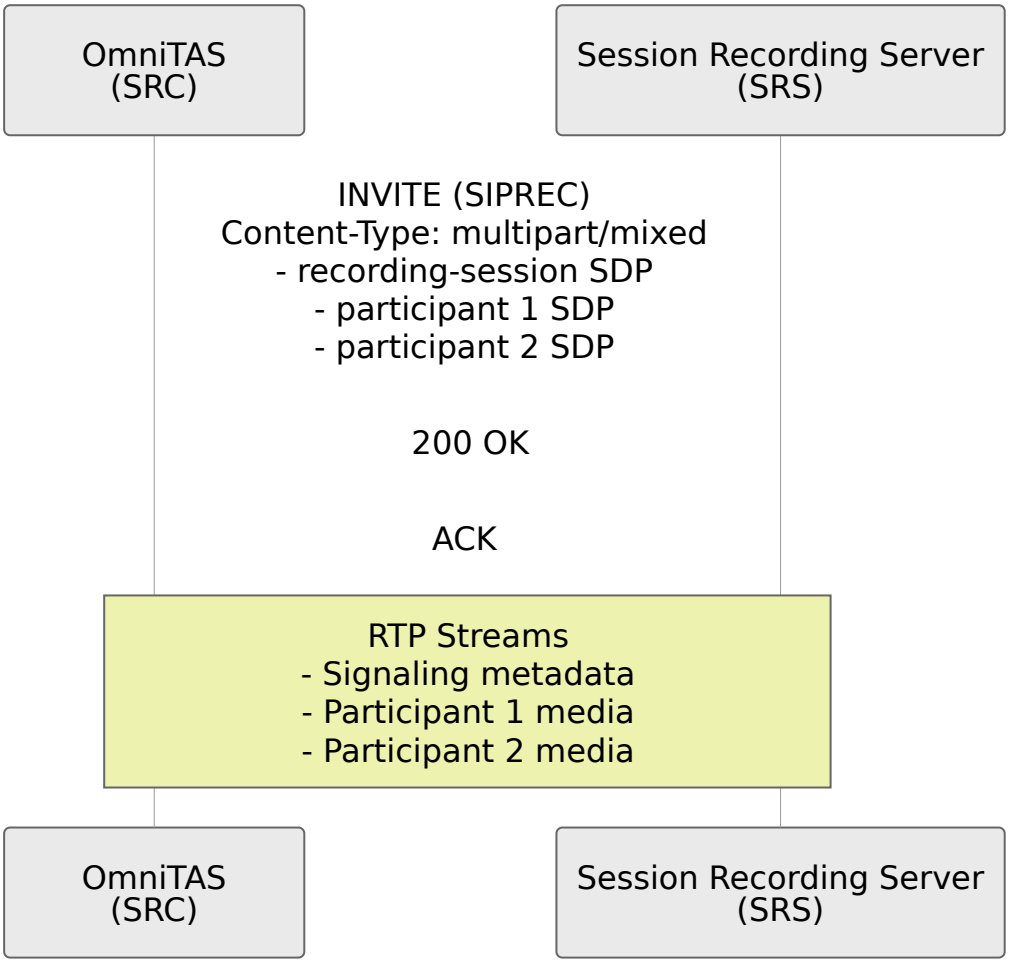
Legend: `[INTERCEPTION]` = Points where lawful interception data is captured

A.2 Emergency Call with Location Tracking


UE/Handset
P-CSCF
S-CSCF
OmniTAS
SBC/Gateway
INVITE
INVITE
INVITE
[INTERCEPTION]
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
English
INVITE
183 Progress
[MEDIA RELAY]
- RTP anchored
- SIPREC fork
180 Ringing
180 Ringing
Sizing your network?
Get a deployment estimate in 60 seconds. Pick a workload size, see the topology, and get pricing — no sales call needed.
Size my network
Talk to engineering
[INTERCEPTION]
- Media copy
- SIPREC send
BYE
BYE
BYE
BYE
[INTERCEPTION]
- CDR updated
- Call end time
- Duration
200 OK



A.3 SIPREC Recording Session Establishment



Appendix B: CDR Schema

The OmniTAS system stores Call Detail Records in a SQLite database (FreeSWITCH CDR format) located at `/etc/freeswitch/db/cdr.db`.

B.1 Key CDR Fields for Lawful Interception

Field Name	Type	Description	Interception Relevance
`uuid`	TEXT	Unique call identifier	Session correlation
`caller_id_number`	TEXT	Calling party number (MSISDN)	**Primary identifier for target tracking**
`caller_id_name`	TEXT	Calling party display name	Identity verification
`destination_number`	TEXT	Called party number	**Target destination tracking**
`start_stamp`	DATETIME	Call start timestamp	**Event timeline**
`answer_stamp`	DATETIME	Call answer timestamp	Call establishment time
`end_stamp`	DATETIME	Call end timestamp	**Session duration calculation**
`duration`	INTEGER	Total call duration (seconds)	Session length
`billsec`	INTEGER	Billable seconds (answered time)	Actual conversation duration

Field Name	Type	Description	Interception Relevance
`hangup_cause`	TEXT	Call termination reason	Call outcome analysis
`sip_hangup_disposition`	TEXT	SIP termination details	Protocol-level termination
`network_addr`	TEXT	Network IP address	**Source location tracking**
`sip_from_user`	TEXT	SIP From header user part	Original SIP identity
`sip_to_user`	TEXT	SIP To header user part	SIP destination
`sip_call_id`	TEXT	SIP Call-ID header	**SIP session correlation**

B.2 CDR Query Examples for Lawful Interception

Query calls by target number:

```
SELECT * FROM cdr
WHERE caller_id_number = '+33612345678'
   OR destination_number = '+33612345678'
ORDER BY start_stamp DESC;
```

Query calls within time window:

```
SELECT * FROM cdr
WHERE start_stamp BETWEEN '2025-11-01 00:00:00' AND '2025-11-30
23:59:59'
  AND (caller_id_number = '+33612345678' OR destination_number =
'+33612345678')
ORDER BY start_stamp DESC;
```

Export to CSV for law enforcement:

```
.mode csv
.output /tmp/interception_report.csv
SELECT caller_id_number, destination_number, start_stamp,
end_stamp, duration, hangup_cause
FROM cdr
WHERE caller_id_number = '+33612345678'
ORDER BY start_stamp DESC;
.output stdout
```

B.3 CDR Retention

- Default retention: Configurable (typically 90 days to 1 year)
- Automatic purging: Supported
- Manual export: Via Web UI at `/cdr` or API
- Format: SQLite database, exportable to CSV/JSON

Appendix C: SIPREC Configuration Examples

SIPREC (Session Initiation Protocol Recording) enables the OmniTAS to send both call signaling and media to external Session Recording Servers for lawful interception.

C.1 SIPREC Architecture

C.2 Triggering SIPREC Recording

Recording can be triggered based on:

Target-based:

- Caller phone number (caller_id_number)
- Called phone number (destination_number)
- SIP URI matching

Event-based:

- All emergency calls (911, 112, etc.)
- Calls to/from specific destinations
- Time-window based recording

Geographic:

- Cell tower location (via P-Access-Network-Info header)
- IP address ranges

C.3 SIPREC Session Content

The SIPREC session sends to the SRS:

Signaling Metadata:

- Complete SIP headers (From, To, P-Asserted-Identity)
- Call-ID and session identifiers
- Timestamps (start, answer, end)
- Caller/callee information

Media Streams:

- Participant 1 RTP stream (caller audio)
- Participant 2 RTP stream (callee audio)
- Codec information
- DTMF tones

C.4 Integration with Law Enforcement

The Session Recording Server provides:

- **X1 Interface:** Administrative function (warrant management)
- **X2 Interface:** Intercept Related Information (IRI) - call metadata
- **X3 Interface:** Content of Communication (CC) - actual media

The OmniTAS serves as the Session Recording Client (SRC) and delivers both IRI and CC to the SRS for handover to law enforcement via standardized interfaces.

Appendix D: Encryption Configuration Guide

D.1 Certificate Generation

Generate TLS Certificate:

```
# Generate private key
openssl genrsa -out server.key 4096

# Generate certificate signing request
openssl req -new -key server.key -out server.csr

# Self-signed certificate (for testing)
openssl x509 -req -days 365 -in server.csr -signkey server.key -
out server.crt

# Production: Obtain certificate from trusted CA
```

Note: SIP signaling to/from IMS does not use TLS. SIP communication is unencrypted TCP/UDP.

D.2 HTTPS Configuration for Web UI

API/Web Server TLS (config/runtime.exs):

```
config :api_ex,
  api: %{
    enable_tls: true,
    tls_cert_path: "priv/cert/server.crt",
    tls_key_path: "priv/cert/server.key",
    tls_versions: [:"tlsv1.2", :"tlsv1.3"],
    ciphers: [
      "ECDHE-RSA-AES256-GCM-SHA384",
      "ECDHE-RSA-AES128-GCM-SHA256",
      "TLS_AES_256_GCM_SHA384",
      "TLS_AES_128_GCM_SHA256"
    ]
  }
```

D.3 SIP Configuration

SIP interfaces use unencrypted TCP/UDP transport. No TLS configuration required.

FreeSWITCH SIP Profile:

```
<!-- SIP profile uses TCP/UDP only -->
<profile name="external">
  <settings>
    <param name="sip-port" value="5060"/>
    <param name="context" value="public"/>
  </settings>
</profile>
```

D.4 Diameter TLS Configuration

Diameter Peer TLS:

```
# Enable TLS for Diameter connections
config :diameter_ex,
  peers: [
    %{
      host: "dra.example.com",
      port: 3868,
      transport: :tls,
      tls_opts: [
        certfile: "priv/cert/diameter.crt",
        keyfile: "priv/cert/diameter.key",
        cacertfile: "priv/cert/ca.crt",
        verify: :verify_peer
      ]
    }
  ]
```

D.5 Database Encryption

SQLite Encryption with SQLCipher:

```
# config/runtime.exs
config :exqlite,
  encryption: true,
  encryption_key: System.get_env("DB_ENCRYPTION_KEY")
```

Note: Database encryption is optional. For lawful interception purposes, physical access controls and database access logging may be sufficient.

D.6 Password Security Configuration

Password hashing is automatically configured with SHA-512 and salt:

```
# Default password hashing configuration
config :pbkdf2_elixir,
  rounds: 65_532,
  salt_len: 16
```

No additional configuration required - secure by default.

Appendix E: Glossary

Regulatory and Standards Bodies

- **ANSSI:** Agence nationale de la sécurité des systèmes d'information - French National Cybersecurity Agency
- **ETSI:** European Telecommunications Standards Institute
- **3GPP:** 3rd Generation Partnership Project - Mobile telecommunications standards organization
- **IETF:** Internet Engineering Task Force - Internet standards body

IMS Network Components

- **IMS:** IP Multimedia Subsystem - All-IP network architecture for multimedia services
- **CSCF:** Call Session Control Function - SIP server in IMS core
  - **P-CSCF:** Proxy-CSCF - First contact point for UE, SIP proxy
  - **I-CSCF:** Interrogating-CSCF - Entry point to operator's network
  - **S-CSCF:** Serving-CSCF - Session control and service triggering
- **HSS:** Home Subscriber Server - Subscriber database
- **TAS:** Telephony/Telecommunications Application Server - Service logic execution

Protocols and Signaling

- **SIP:** Session Initiation Protocol (RFC 3261) - Signaling protocol for voice/video calls
- **SDP:** Session Description Protocol (RFC 4566) - Media session parameters
- **RTP:** Real-time Transport Protocol (RFC 3550) - Media stream transport
- **RTCP:** RTP Control Protocol - Quality monitoring for RTP
- **SRTP:** Secure RTP (RFC 3711) - Encrypted media streams
- **Diameter:** AAA protocol used in IMS (authentication, authorization, accounting)
  - **Sh:** Diameter interface for subscriber data access
  - **Ro:** Diameter interface for online charging

- **SIPREC:** Session Initiation Protocol Recording (RFC 7866) - Call recording protocol

Telecommunications Equipment

- **SBC:** Session Border Controller - Network edge security and media gateway
- **MRF:** Media Resource Function - Media processing (transcoding, mixing, recording)
- **UE:** User Equipment - Mobile handset or device
- **PSAP:** Public Safety Answering Point - Emergency services call center
- **DRA:** Diameter Routing Agent - Diameter message routing

Lawful Interception

- **LI:** Lawful Interception - Legal monitoring of telecommunications
- **IRI:** Intercept Related Information - Call metadata for law enforcement
- **CC:** Content of Communication - Actual voice/media content
- **SRC:** Session Recording Client - SIPREC client (OmniTAS role)
- **SRS:** Session Recording Server - SIPREC server for recording storage
- **X1 Interface:** LI administrative interface (warrant provisioning)
- **X2 Interface:** LI interface for IRI delivery
- **X3 Interface:** LI interface for CC delivery
- **R226:** Articles R226-3 and R226-7 of French Penal Code governing interception equipment

Call Processing

- **CDR:** Call Detail Record - Billing and logging record for each call
- **B2BUA:** Back-to-Back User Agent - SIP element that acts as both client and server
- **DTMF:** Dual-Tone Multi-Frequency - Touch-tone signals
- **MSISDN:** Mobile Station International Subscriber Directory Number - Phone number
- **IMSI:** International Mobile Subscriber Identity - Unique subscriber identifier
- **E.164:** International numbering plan for telephone numbers

Security and Encryption

- **TLS:** Transport Layer Security (RFC 5246, RFC 8446) - Encryption protocol
- **PFS:** Perfect Forward Secrecy - Cryptographic property ensuring session key security
- **SHA-512:** Secure Hash Algorithm with 512-bit output
- **AES:** Advanced Encryption Standard
- **RSA:** Rivest-Shamir-Adleman - Public key cryptography algorithm
- **ECDSA:** Elliptic Curve Digital Signature Algorithm
- **PKI:** Public Key Infrastructure - Certificate management system
- **CA:** Certificate Authority - Issues digital certificates
- **CRL:** Certificate Revocation List
- **OCSP:** Online Certificate Status Protocol

Network and Location

- **MAP:** Mobile Application Part - SS7 protocol for mobile networks
- **HLR:** Home Location Register - Subscriber location database (legacy)
- **SS7:** Signaling System No. 7 - Legacy telephony signaling
- **NANP:** North American Numbering Plan
- **Cell Tower/Cell ID:** Mobile network base station identifier for location tracking

Data Formats and Storage

- **SQLite:** Embedded relational database
- **SQLCipher:** SQLite extension with encryption support
- **CSV:** Comma-Separated Values - Export format
- **JSON:** JavaScript Object Notation - Data interchange format
- **XML:** eXtensible Markup Language - Structured data format

Application Components

- **API:** Application Programming Interface - Programmatic access
- **UI:** User Interface - Web-based control panel
- **RBAC:** Role-Based Access Control - Permission system
- **UUID:** Universally Unique Identifier - Session tracking

Document Version: 1.0 **Date:** 2025-11-29 **Prepared for:** ANSSI R226 Authorization Application **Document Classification:** Regulatory Compliance - Confidential

Configuration Guide

Back to Main Documentation

This document provides comprehensive configuration reference for the TAS Application Server.

Related Documentation

Core Configuration

- **Main README** - Overview and quick start
- **Operations Guide** - Monitoring and operational tasks
- **Metrics Reference** - Prometheus metrics and monitoring

Integration Interfaces

- **Sh Interface** - Subscriber data retrieval from HSS/Repository
- **Online Charging (Ro)** - OCS integration and credit control
- **SS7 MAP** - HLR queries for roaming and call forwarding

Call Processing

- **Dialplan Configuration** - XML dialplan and call routing logic
- **Number Translation** - E.164 normalization rules
- ⚙ **Supplementary Services** - Call forwarding, CLI blocking, emergency

Value-Added Services

- **Voicemail** - Voicemail service with SMS notifications
- **TTS Prompts** - Text-to-Speech prompt configuration
- **IMS Conference Server** - Multi-party conferencing

Testing & Compliance

- **HLR & Call Simulator** - Testing tools
- **ANSSI R226 Compliance** - French market compliance

Config

The Application Server needs:

- To connect to SIP Trunks / SBCs for calls to/from off-net
- Connect to the DRA or HSS to get the `Sh`
- Optionally connect to DRA or OCS for `Ro` online charging
- Dialplan Config
- Configuration around the dialing rules / number translation
- Voicemail config
- Prompts
- Tests
- Metrics (Prometheus)

Event Socket Configuration

The Event Socket is used for call control, monitoring active calls, and interacting with the telephony engine. This connection allows the TAS to control call routing, retrieve channel variables, and manage active sessions.

Configuration Location: `config/runtime.exs`

```
config :tas,
  fs_event_socket: %{
    host: "127.0.0.1",
    port: 8021,
    secret: "YourSecretPassword"
  }
```

Configuration Parameters:

- **`host`** (string, required): Hostname or IP address of the Event Socket server
  - Default: `"127.0.0.1"` (localhost)
  - Use localhost if the telephony engine runs on the same server as TAS
  - Use remote IP for distributed deployments
  - Example: `"10.8.82.60"` for remote connection
- **`port`** (integer, required): TCP port for Event Socket connections
  - Default: `8021`
  - Standard Event Socket port is 8021
  - Must match the Event Socket configuration in your telephony engine
  - Example: `8021`
- **`secret`** (string, required): Authentication password for Event Socket
  - Must match the password configured in your telephony engine
  - Used for authenticating ESL connections
  - **Security Note:** Use a strong random password and keep it secure
  - Example: `"cd463RZ8qMk9AHMMDGT3V"`

Use Cases:

- Real-time call control and routing
- Retrieving active call information for the `/calls` view in Control Panel
- Executing dialplan applications programmatically
- Monitoring call state changes and events
- Managing conference calls

Connection Behavior:

- TAS establishes persistent connections to the Event Socket
- Automatically reconnects on connection failure
- Used for both inbound (receiving events) and outbound (controlling calls) modes
- Connection timeouts and retry logic are built-in

Security Considerations:

- Always use a strong, unique password for the `secret` parameter
- If using remote connections, ensure firewall rules allow only trusted TAS servers
- Consider using localhost-only connections when TAS and telephony engine are co-located
- Do not expose Event Socket port to public networks

Troubleshooting:

- **Connection Refused:** Verify the telephony engine is running and Event Socket is enabled
- **Authentication Failed:** Check that `secret` matches the telephony engine configuration
- **Timeout Errors:** Verify network connectivity and firewall rules
- **Cannot Control Calls:** Ensure TAS has connected successfully (check logs)

Control Panel Configuration

The Control Panel provides a web-based interface for monitoring and managing the TAS system. This includes viewing subscribers, CDRs, active calls, Diameter peers, gateways, and system configuration.

Configuration Location: `config/runtime.exs`

```
config :control_panel,
  page_order: ["/application", "/configuration"]

config :control_panel, ControlPanelWeb.Endpoint,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 443,
    keyfile: "priv/cert/server.key",
    certfile: "priv/cert/server.crt"
  ]
```

Configuration Parameters:

Page Order Configuration

- `page_order` (list of strings): Controls the display order of configuration pages in the Control Panel
  - Specifies which pages appear in navigation and their order
  - Example: `["/application", "/configuration"]`
  - Default: If not set, pages appear in default alphabetical order

Web Endpoint Configuration

- `url` (map): Public URL configuration for the Control Panel
  - `host`: Hostname for generating URLs (e.g., `"tas.example.com"` or `"0.0.0.0"`)
  - `path`: Base path for all Control Panel routes (default: `"/"`)
  - Used for generating absolute URLs in redirects and links
- `https` (map): HTTPS/TLS configuration for secure access
  - `port` (integer): HTTPS port number (standard is `443`)
  - `keyfile` (string): Path to TLS private key file (PEM format)
  - `certfile` (string): Path to TLS certificate file (PEM format)
  - Both files must be readable by the TAS application

Certificate Management:

The Control Panel requires valid TLS certificates for HTTPS access:

1. **Self-Signed Certificates** (Development/Testing):

```
openssl req -x509 -newkey rsa:4096 -keyout priv/cert/server.key
\
  -out priv/cert/server.crt -days 365 -nodes
```

2. **Production Certificates:**
   - Use certificates from a trusted Certificate Authority (CA)
   - Common providers: Let's Encrypt (free), commercial CAs

- Ensure certificates include full chain for browser trust
- Keep private keys secure with appropriate file permissions (`chmod 600`)

Access Control:

The Control Panel provides access to sensitive operational data:

- **Subscriber Information:** Registration details, call history, locations
- **Call Detail Records:** Complete call records with MSISDN data
- **System Configuration:** Diameter peers, gateways, routing
- **Active Calls:** Real-time monitoring of ongoing sessions

Recommended Security Measures:

- Deploy behind firewall or VPN for production environments
- Use strong TLS certificates from trusted CAs
- Implement network-level access controls (IP whitelisting)
- Consider additional authentication layers if exposing externally
- Regularly audit access logs
- Use HTTPS only - never serve over plain HTTP

Common Deployment Patterns:

1. **Internal-Only Access:**

```
url: [host: "10.8.82.60", path: "/"]  # Internal network only
```

2. **External Access with Domain:**

```
url: [host: "tas.operator.com", path: "/"]
https: [port: 443, ...]
```

3. **Behind Reverse Proxy:**

```
url: [host: "tas.internal", path: "/panel"]  # Nginx/Apache
forwards to this
```

Troubleshooting:

- **Certificate Errors:** Verify paths to `keyfile` and `certfile` are correct and files are readable
- **Port Already in Use:** Check if another service is using port 443, or change to another port
- **Cannot Access UI:** Verify firewall rules allow access to the configured HTTPS port
- **SSL Handshake Failures:** Ensure certificate and key match and are in PEM format

---

API Configuration

The TAS includes a REST API for programmatic access to system functions, subscriber management, and operational data. The API supports OpenAPI/Swagger documentation and is secured with TLS.

Configuration Location: `config/runtime.exs`

```
config :api_ex,
  api: %{
    port: 8444,
    listen_ip: "0.0.0.0",
    product_name: "OmniTAS",
    title: "API - OmniTAS",
    hostname: "localhost",
    enable_tls: true,
    tls_cert_path: "priv/cert/server.crt",
    tls_key_path: "priv/cert/server.key"
  }
```

Configuration Parameters:

- **`port`** (integer, required): TCP port for the API server
  - Default: `8444`
  - Choose a port that doesn't conflict with other services

    - Standard HTTPS port is 443, but custom ports are common for APIs
    - Example: `8444`, `8443`, `9443`
- **`listen_ip`** (string, required): IP address to bind the API server
    - `"0.0.0.0"`: Listen on all network interfaces (external access)
    - `"127.0.0.1"`: Listen only on localhost (internal access only)
    - Specific IP: Bind to a particular interface (e.g., `"10.8.82.60"`)
    - **Security:** Use `"127.0.0.1"` if API only needed internally
- **`product_name`** (string): Product identifier for API metadata
    - Used in API responses and documentation
    - Example: `"OmniTAS"`, `"MyOperator-IMS"`
- **`title`** (string): Human-readable title for API documentation
    - Displayed in OpenAPI/Swagger UI header
    - Example: `"API - OmniTAS"`, `"IMS Application Server API"`
- **`hostname`** (string): Hostname for API server in documentation
    - Used in OpenAPI spec for generating example URLs
    - Should match how clients access the API
    - Examples: `"localhost"`, `"api.operator.com"`, `"10.8.82.60"`
- **`enable_tls`** (boolean): Enable or disable TLS/HTTPS for API
    - `true`: Serve API over HTTPS (recommended for production)
    - `false`: Serve API over HTTP (only for testing/development)
    - **Security:** Always use `true` in production environments
- **`tls_cert_path`** (string): Path to TLS certificate file (PEM format)
    - Required when `enable_tls: true`
    - Must be readable by the TAS application
    - Example: `"priv/cert/server.crt"`
- **`tls_key_path`** (string): Path to TLS private key file (PEM format)

    - Required when `enable_tls: true`
    - Must be readable by the TAS application
    - **Security:** Protect with file permissions (`chmod 600`)
    - Example: `"priv/cert/server.key"`

API Features:

The REST API provides programmatic access to:

- Subscriber management and provisioning
- Call Detail Records (CDR) queries
- System status and health checks
- Diameter peer status
- Gateway status and statistics
- Active call monitoring
- Configuration management

OpenAPI/Swagger Documentation:

The API includes built-in OpenAPI (Swagger) documentation:

- Access Swagger UI at: `https://hostname:port/api/swaggerui`
- OpenAPI JSON spec at: `https://hostname:port/api/openapi`
- Interactive API testing directly from the browser
- Complete endpoint documentation with request/response schemas

Security Considerations:

- **Authentication:** Implement API authentication based on your security requirements
- **Network Access:** Use firewall rules to restrict API access to authorized clients
- **TLS Required:** Always enable TLS in production (`enable_tls: true`)
- **Certificate Validation:** Use trusted certificates for production APIs
- **Rate Limiting:** Consider implementing rate limiting for public-facing APIs
- **Access Logs:** Monitor API access logs for suspicious activity

Example Usage:

```
# Query API with curl (replace with actual endpoint)
curl -k https://localhost:8444/api/health

# Access Swagger documentation
https://localhost:8444/api/swaggerui
```

Common Deployment Scenarios:

1. **Internal API Only:**

   ```
   listen_ip: "127.0.0.1"  # Only accessible from localhost
   enable_tls: false       # HTTP for internal testing
   ```

2. **Production API with TLS:**

   ```
   listen_ip: "0.0.0.0"    # Accessible from network
   enable_tls: true        # HTTPS required
   hostname: "api.operator.com"
   ```

3. **Development/Testing:**

   ```
   listen_ip: "0.0.0.0"
   enable_tls: false       # HTTP for easier testing
   port: 8080              # Non-privileged port
   ```

Troubleshooting:

- **Port Binding Failed:** Verify port is not in use by another service, or run as root for ports < 1024
- **TLS Errors:** Check that certificate and key paths are correct and files are readable
- **Cannot Connect:** Verify firewall allows access to the configured port
- **Certificate Mismatch:** Ensure `hostname` matches the certificate Common Name (CN) or SAN

- **API Returns 404:** Check that the API application started successfully in logs

---

SIP Trunk Config

Ansible is responsible for creating the XML config for each outgoing gateway, visible in the `Gateways` tab, which are used for outgoing calls.

CSCF addresses and Gateway addresses have to be included in the that are visible in the runtime config, so we know what IPs to allow calls from, we do this in the `allowed_sbc_source_ips` for Gateways / SBCs (sources that will send MT traffic towards the network) and `allowed_cscf_ips` for CSCFs (sources that MO traffic will originate from).

Note - If you will route calls from your TAS to itself (ie a MO call to an on-net subscriber routes back into the MT dialplan) then your TAS IP must also be in the allowed source IPs list.

```
config :tas,
  allowed_sbc_source_ips: ["10.5.198.200", "103.26.174.36"],
  allowed_cscf_ips: ["10.8.3.34"],
```

From the Web UI we can see the state of each gateway, and:

- SIP Registration status (if register is enabled)
- SIP Realm
- SIP Proxy Address (if used)
- Username
- Ping Time (Average SIP OPTIONs response time (if SIP OPTIONs enabled))
- Uptime (Seconds since the profile was restarted or came up)
- Calls in / Calls Out / Failed Calls In / Failed Calls Out
- Last SIP OPTIONs ping time (Epoch)
- SIP OPTIONs ping frequency
- More info in the **detail** button

Gateway Configuration Reference

Gateways are configured in XML format. Each gateway represents a SIP trunk connection to an external SBC, carrier, or PSTN gateway.

Basic Gateway Example:

```
<include>
  <gateway name="carrier_trunk">
    <param name="proxy" value="203.0.113.50;transport=tcp"/>
    <param name="register" value="true"/>
    <param name="caller-id-in-from" value="true"/>
    <param name="username" value="trunk_user"/>
    <param name="password" value="secure_password"/>
    <param name="register-transport" value="tcp"/>
    <param name="retry-seconds" value="30"/>
    <param name="ping" value="25"/>
  </gateway>
</include>
```

Gateway without Registration:

```
<include>
  <gateway name="sbc_static">
    <param name="proxy" value="198.51.100.10"/>
    <param name="register" value="false"/>
    <param name="caller-id-in-from" value="true"/>
  </gateway>
</include>
```

Gateway Parameters

Required Parameters

`name` (gateway attribute)

- The unique name identifier for this gateway
- Used in dialplan to reference the gateway: `sofia/gateway/name/destination`
- Example: `<gateway name="my_trunk">`

`proxy`

- SIP proxy/gateway IP address or hostname
- Can include port and transport protocol
- Examples:
  - `value="203.0.113.50"` (default port 5060, UDP)
  - `value="203.0.113.50:5061"` (custom port)
  - `value="203.0.113.50;transport=tcp"` (TCP transport)
  - `value="203.0.113.50:5061;transport=tls"` (TLS on port 5061)

`register`

- Whether to send SIP REGISTER to the gateway
- Values: `true` | `false`
- Set to `true` if the trunk requires registration
- Set to `false` for static IP-based trunks

Authentication Parameters

`username`

- SIP authentication username
- Used in REGISTER and for digest authentication
- Required if `register="true"`
- Example: `value="trunk_account_123"`

`password`

- SIP authentication password
- Used for digest authentication challenges
- Required if `register="true"`
- Example: `value="MySecureP@ssw0rd"`

`realm`

- SIP realm for authentication
- Optional - usually auto-detected from challenge
- Example: `value="sip.carrier.com"`

`auth-username`

- Alternative username for authentication (if different from `username`)
- Rarely needed - only if carrier requires different user in auth vs From header
- Example: `value="auth_user_456"`

Registration Parameters

`register-transport`

- Transport protocol for REGISTER messages
- Values: `udp` | `tcp` | `tls`
- Must match transport specified in `proxy` parameter
- Example: `value="tcp"`

`register-proxy`

- Alternative proxy address for REGISTER (if different from call routing)
- Useful when registration server differs from call routing server
- Example: `value="register.carrier.com:5060"`

`retry-seconds`

- Seconds to wait before retrying failed registration
- Default: `30`
- Range: `5` to `3600`

- Example: `value="30"`

`expire-seconds`

- Registration expiry time in seconds
- Default: `3600` (1 hour)
- The gateway will re-register before expiry
- Example: `value="1800"` (30 minutes)

`caller-id-in-from`

- Include caller ID in SIP From header
- Values: `true` | `false`
- `true`: From header includes actual caller number (required by most carriers)
- `false`: From header uses gateway username
- **Recommendation:** Set to `true` for most deployments
- Example: `value="true"`

Monitoring Parameters

`ping`

- Send SIP OPTIONS ping every N seconds
- Monitors gateway availability and measures latency
- Disabled if not specified or set to `0`
- Typical values: `15` to `60` seconds
- Visible in Gateway Status UI as "Ping Time"
- Example: `value="25"`

`ping-max`

- Maximum time (seconds) to retry pings before marking gateway down
- Default: Calculated from `ping` interval
- Example: `value="3"`

Call Routing Parameters

`extension`

- Fixed destination number to always dial on this gateway
- Rarely used - usually destination comes from dialplan
- Example: `value="+12125551234"`

`extension-in-contact`

- Include extension in Contact header
- Values: `true` | `false`
- Default: `false`
- Example: `value="false"`

`contact-params`

- Additional parameters to append to Contact header
- Useful for carrier-specific requirements
- Example: `value="line=1;isup=true"`

Advanced Parameters

`from-user`

- Override username in From header
- Default: Uses calling number or gateway username
- Example: `value="trunk_pilot"`

`from-domain`

- Override domain in From header
- Default: Uses proxy domain
- Example: `value="my-domain.com"`

`outbound-proxy`

- Outbound proxy for all SIP messages
- Different from `proxy` - used as Route header target
- Example: `value="edge-proxy.carrier.com:5060"`

`context`

- Dialplan context for incoming calls from this gateway
- Default: `public`
- Allows different incoming call routing per gateway
- Example: `value="from-carrier"`

`channels`

- Maximum concurrent calls on this gateway
- Default: Unlimited
- Used for capacity management
- Example: `value="100"`

`dtmf-type`

- DTMF transmission method
- Values: `rfc2833` | `info` | `inband` | `auto`
- Default: `rfc2833` (recommended)
- `rfc2833`: RTP telephone events (most common)
- `info`: SIP INFO messages
- `inband`: Audio tones
- Example: `value="rfc2833"`

`codec-prefs`

- Preferred codec list for this gateway
- Comma-separated list in preference order
- Example: `value="PCMU,PCMA,G729"`
- Common codecs: `PCMU`, `PCMA`, `G729`, `AMR`, `AMR-WB`, `G722`, `OPUS`

`rtp-timeout-sec`

- Hangup call if no RTP received for N seconds
- Default: `0` (disabled)
- Useful for detecting dead calls
- Example: `value="120"`

`rtp-hold-timeout-sec`

- Timeout for calls on hold with no RTP
- Default: `0` (disabled)
- Example: `value="1800"` (30 minutes)

SIP Signaling Options

`sip-port`

- Local SIP port to use for this gateway
- Default: Profile's port
- Rarely needed
- Example: `value="5060"`

`rtp-ip`

- Local IP address for RTP media
- Default: Profile's RTP IP
- Example: `value="10.0.0.5"`

`register-proxy-port`

- Port for registration proxy
- Only needed if different from proxy port
- Example: `value="5061"`

`contact-host`

- Override host portion of Contact header
- Useful for NAT scenarios
- Example: `value="public-ip.example.com"`

`distinct-to`

- Use distinct To header (different from Request-URI)
- Values: `true` | `false`
- Carrier-specific requirement

- Example: `value="false"`

`cid-type`

- Caller ID type in Remote-Party-ID or P-Asserted-Identity headers
- Values: `rpid` | `pid` | `none`
- `rpid`: Remote-Party-ID header
- `pid`: P-Asserted-Identity header
- Example: `value="pid"`

`extension-in-contact`

- Add extension parameter to Contact URI
- Values: `true` | `false`
- Example: `value="true"`

Transport Security

`transport` (in proxy parameter)

- Transport protocol
- Values: `udp` | `tcp` | `tls` | `ws` | `wss`
- Specified as part of proxy value
- Example: `proxy="203.0.113.50;transport=tcp"`

For TLS connections, additional certificate configuration may be required in the SIP profile.

Complete Example with Common Options

```
<include>
  <gateway name="primary_carrier">
    <!-- Required: Basic connection -->
    <param name="proxy"
value="sbc.carrier.com:5060;transport=tcp"/>
    <param name="register" value="true"/>

    <!-- Authentication -->
    <param name="username" value="customer_trunk_01"/>
    <param name="password" value="SecurePassword123"/>

    <!-- Registration -->
    <param name="register-transport" value="tcp"/>
    <param name="expire-seconds" value="1800"/>
    <param name="retry-seconds" value="30"/>

    <!-- Caller ID -->
    <param name="caller-id-in-from" value="true"/>

    <!-- Monitoring -->
    <param name="ping" value="30"/>

    <!-- Media -->
    <param name="codec-prefs" value="PCMU,PCMA,G729"/>
    <param name="dtmf-type" value="rfc2833"/>

    <!-- Call limits -->
    <param name="channels" value="100"/>

    <!-- RTP timeouts -->
    <param name="rtp-timeout-sec" value="300"/>
  </gateway>
</include>
```

Gateway Usage in Dialplan

Reference gateways in your dialplan using the `sofia/gateway/name/destination` format:

```
<!-- Route to specific gateway -->
<action application="bridge" data="sofia/gateway/primary_carrier/+121

<!-- Route using variable -->
<action application="bridge" data="sofia/gateway/primary_carrier/${ta

<!-- Route with custom SIP headers -->
<action application="bridge" data="{sip_h_X-Custom=Value}sofia/gatewa

<!-- Failover between gateways -->
<action application="bridge"
data="sofia/gateway/primary_carrier/${tas_destination_number}|sofia/g
```

Troubleshooting Gateway Issues

Gateway Won't Register:

- Verify `username` and `password` are correct
- Check `proxy` address is reachable
- Confirm `register-transport` matches carrier requirements
- Review logs for authentication failures

Calls Fail:

- Check gateway status in Web UI (`/gw`)
- Verify `caller-id-in-from` setting matches carrier requirement
- Confirm codec compatibility with `codec-prefs`
- Check firewall allows SIP and RTP traffic

Poor Call Quality:

- Review `ping` times in Gateway Status
- Check `rtp-timeout-sec` isn't too aggressive
- Verify codec preferences match network capabilities
- Monitor network latency and packet loss

SBC Mode (Skipping Diameter Lookups)

Sometimes this node is deployed purely as a **Session Border Controller (SBC)**, passing calls through without acting as a full Telephony Application Server. In that role you do not want it talking Diameter at all - no HSS subscriber lookups, no OCS credit control.

The `skip_diameter_lookups` flag puts the node into this pass-through mode:

```
config :tas,
  skip_diameter_lookups: true
```

Behaviour when `true`:

- **Sh interface (HSS):** No User-Data-Request (UDR) is sent. MO callers and MT called parties resolve to the default subscriber-data shape (all fields `"none"`, `no_reply_timer` from `default_no_reply_timer`), so calls proceed with `hangup_case = "none"` instead of being rejected as `UNALLOCATED_NUMBER`.
- **On-net detection (MO):** Skipped - every destination is treated as off-net and routed out via the normal gateway/SBC path.
- **Ro interface (OCS):** No Credit-Control-Request (CCR-I / CCR-U / CCR-T) is sent. Authorization, answer, and hangup charging are all bypassed; calls are authorized with unlimited time. This overrides `online_charging.enabled` - when `skip_diameter_lookups` is `true`, Ro stays off regardless of the online charging config.

Defaults and compatibility:

- The flag is **optional** and defaults to `false` (full TAS behaviour) when omitted, so existing configs are unaffected and it is not a required key in config validation.
- You can leave the `diameter_ex` peer config and the `online_charging` block in place; with the flag set, neither is exercised. You do not need to define Diameter peers for an SBC-only node.

```
# Full TAS node (default)
config :tas,
  skip_diameter_lookups: false

# SBC-only node - no Diameter traffic
config :tas,
  skip_diameter_lookups: true
```

Sh Data-Reference Selection (`sh_data_references`)

Optional. Controls which Sh `Data-Reference` values the TAS requests when looking up subscriber data, **per side** of the call (`caller` = MO source, `called` = MT destination). When unset, the TAS requests the full default set for both sides — so this key only matters when an HSS rejects part of that set (e.g. answering `5012 UNABLE_TO_COMPLY` or `5101 OPERATION_NOT_ALLOWED` for certain references, which fails the whole UDR).

```
config :tas,
  # Per-side override; an omitted side falls back to the full
default set.
  sh_data_references: [
    caller: [10, 16, 17, 32],
    called: [10, 16, 17, 32]
  ]

# ...or a single bare list applied to both sides:
config :tas, sh_data_references: [10, 16, 17, 32]
```

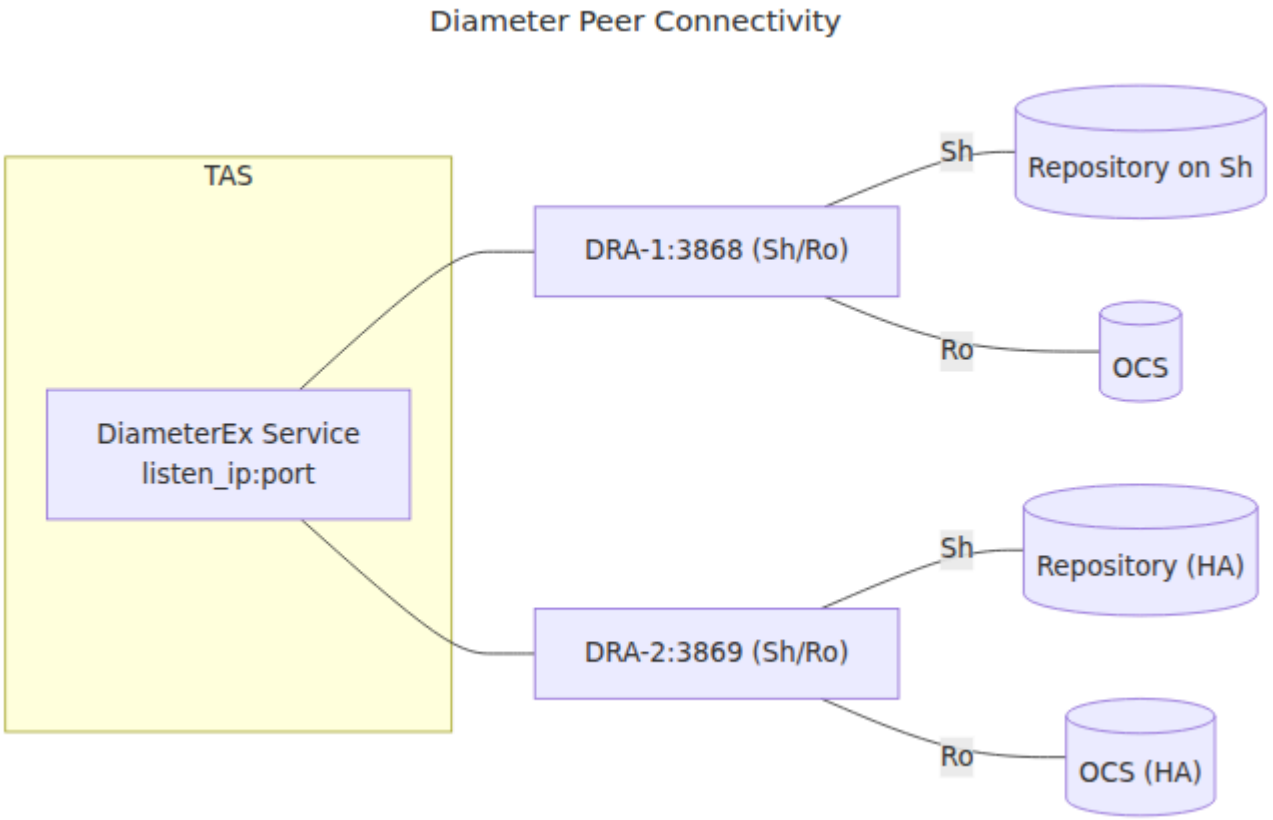
See the **Sh Interface** guide for the Data-Reference table, accepted shapes, and the Session-Id call-disposition (`;mo` / `;mt`) tag added to every UDR for correlation.

Diameter Peer Config

Dimeter peers must be defined in the runtime config.

This config is largely boilerplate.

The Ro interface does not need to be included in the Applications if Ro is not used in your deployment.



```
config :diameter_ex,
  diameter: %{
    service_name: :omnitouch_tas,
    listen_ip: "10.8.82.60",
    listen_port: 3868,
    decode_format: :map,
    host: "example-dc01-as01",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    product_name: "OmniTAS",
    request_timeout: 5000,
    peer_selection_algorithm: :random,
    allow_undefined_peers_to_connect: true,
    log_unauthorized_peer_connection_attempts: true,
    control_module: Tas.Control.Diameter,
    processor_module: DiameterEx.Processor,
    auth_application_ids: [],
    acct_application_ids: [],
    vendor_id: 10415,
    supported_vendor_ids: [10415],
    # Optional: Global destination_realm for all applications
    # destination_realm: "global.destination.realm",
    applications: [
      %{
        application_name: :sh,
        application_dictionary: :diameter_gen_3gpp_sh,
        # Optional: Application-specific destination_realm for Sh
requests
        # destination_realm: "sh.destination.realm",
        vendor_specific_application_ids: [
          %{
            vendor_id: 10415,
            auth_application_id: 16_777_217,
            acct_application_id: nil
          }
        ]
      },
      %{
        application_name: :ro,
        application_dictionary: :diameter_gen_3gpp_ro,
        # Optional: Application-specific destination_realm for Ro
requests
        # destination_realm: "ocs.destination.realm",
        vendor_specific_application_ids: [
```

```
          %{
            vendor_id: 0,
            auth_application_id: 4,
            acct_application_id: nil
          }
        ]
      }
    ],
    peers: [
      %{
        port: 3868,
        host: "example-dc01-
dra01.epc.mnc001.mcc001.3gppnetwork.org",
        ip: "1.2.3.4",
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        tls: false,
        transport: :diameter_tcp,
        initiate_connection: true
      },
      %{
        port: 3869,
        host: "example-dc01-
dra02.epc.mnc001.mcc001.3gppnetwork.org",
        ip: "1.2.3.44",
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        tls: false,
        transport: :diameter_tcp,
        initiate_connection: true
      }
    ]
  }
```

Diameter Configuration Parameters

Service Configuration:

- **`service_name`** (atom): Unique identifier for this Diameter service instance
  - Example: `:omnitouch_tas`
  - Used internally for service management

- **`listen_ip`** (string): IP address to bind for Diameter connections
  - Example: `"10.8.82.60"`
  - Use `"0.0.0.0"` to listen on all interfaces
  - Peers will connect to this IP
- **`listen_port`** (integer): TCP port for Diameter connections
  - Standard Diameter port: `3868`
  - Must not conflict with other services
- **`host`** (string): Diameter host identity (without realm)
  - Example: `"example-dc01-as01"`
  - Combined with `realm` to form Origin-Host AVP
  - Must be unique within the Diameter network
- **`realm`** (string): Diameter realm identity
  - Example: `"epc.mnc001.mcc001.3gppnetwork.org"`
  - Used in Origin-Realm AVP
  - Must match 3GPP network identifier conventions
- **`product_name`** (string): Product identifier in CER/CEA messages
  - Example: `"OmniTAS"`
  - Used in Capabilities-Exchange messages
- **`request_timeout`** (integer): Timeout in milliseconds for Diameter requests
  - Example: `5000` (5 seconds)
  - Requests without response within this time will timeout
- **`peer_selection_algorithm`** (atom): Algorithm for selecting peer when multiple available
  - Values: `:random` | `:round_robin` | `:priority`
  - `:random`: Random peer selection
  - `:round_robin`: Distribute requests evenly across peers

- **`vendor_id`** (integer): 3GPP vendor ID
    - Standard 3GPP vendor ID: `10415`
    - Used in Vendor-Specific-Application-Id AVP

Destination Realm Configuration

The `destination_realm` parameter controls the `Destination-Realm` AVP included in Diameter requests. This AVP tells the Diameter Routing Agent (DRA) where to route the request.

Three levels of configuration:

1. **Application-specific** (highest priority): Set `destination_realm` within each application configuration
2. **Global**: Set `destination_realm` at the top level of the diameter config
3. **Fallback** (lowest priority): Uses the `realm` value if neither of the above are configured

Configuration Examples:

```
# Example 1: Application-specific destination realms
config :diameter_ex,
  diameter: %{
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    applications: [
      %{
        application_name: :sh,
        destination_realm:
"hss.epc.mnc001.mcc001.3gppnetwork.org",
        # ... other config
      },
      %{
        application_name: :ro,
        destination_realm:
"ocs.epc.mnc001.mcc001.3gppnetwork.org",
        # ... other config
      }
    ]
  }

# Example 2: Global destination realm with app-specific override
config :diameter_ex,
  diameter: %{
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    destination_realm: "dra.epc.mnc001.mcc001.3gppnetwork.org",  #
Default for all apps
    applications: [
      %{
        application_name: :sh,
        # Will use global: "dra.epc.mnc001.mcc001.3gppnetwork.org"
      },
      %{
        application_name: :ro,
        destination_realm:
"ocs.epc.mnc001.mcc001.3gppnetwork.org",  # Override
      }
    ]
  }

# Example 3: No destination_realm configured (uses realm)
config :diameter_ex,
  diameter: %{
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
```

```
    # No destination_realm specified anywhere
    applications: [
      %{
        application_name: :sh,
        # Will use realm fallback:
"epc.mnc001.mcc001.3gppnetwork.org"
      }
    ]
  }
```

When to Use Destination Realm:

- **Different backend systems**: When Sh goes to HSS and Ro goes to OCS in different realms
- **DRA routing**: When DRA uses Destination-Realm to route to different backend clusters
- **Multi-tenant deployments**: Route different applications to different tenant realms
- **Testing scenarios**: Override destination realm per application for testing without changing peers

Fallback Hierarchy:

```
Application-specific destination_realm
  ↓ (if not set)
Global destination_realm
  ↓ (if not set)
realm
```

This ensures the mandatory `Destination-Realm` AVP is always present in outgoing requests.

You can check the status of Diameter peers from the **Diameter** tab on the Web UI.

You can also test retriving `Sh` data from the **Sh** tab on the Web UI to try to fetch any of the data from Sh.

Dialplan Configuration & Call Routing

Back to Main Documentation

Comprehensive guide to XML dialplan configuration, call routing logic, and dialplan variables.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - SIP trunk and gateway configuration
- **Operations Guide** - Dialplan testing and templates viewer

Call Processing Flow

- **Number Translation** - E.164 normalization (happens before dialplan)
- **Sh Interface** - Subscriber data retrieved for dialplan variables
- **SS7 MAP** - MSRN/HLR data in dialplan variables
- **Online Charging** - OCS authorization in call flow

Services Implementation

- ⚙ **Supplementary Services** - Implementing call forwarding, CLI blocking in dialplan
- **Voicemail** - Voicemail routing and deposit/retrieval in dialplan
- **TTS Prompts** - Using prompts in dialplan with playback

Monitoring

- **Dialplan Metrics** - Dialplan-specific metrics and monitoring
- **Metrics Reference** - General system metrics

Dialplan Config / Call Routing

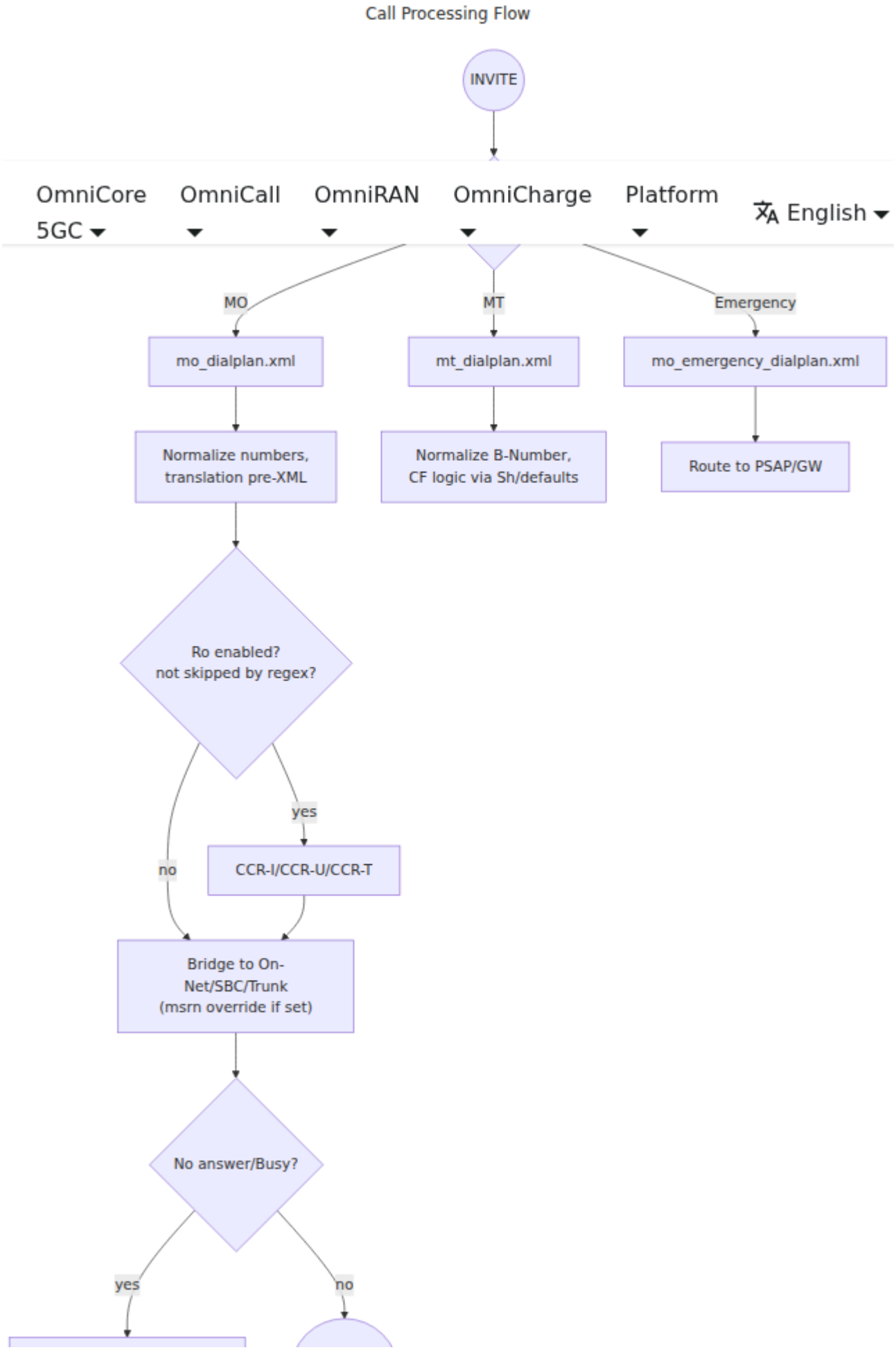
The TAS uses XML dialplans with a schema compatible with standard telecom XML dialplan formats, with variables populated by the TAS. This means you can define your own dialplan as needed, with the business logic for the operator, but have all the required data such as Repository Data, SS7 routing info, IMPI / IMPU identities, dialplan normalization, etc, etc.

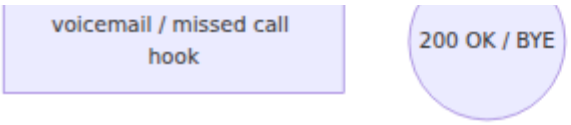
Dialplans are written into `priv/templates` and take the form:

- `mo_dialplan.xml` - Mobile Originated Call Dialplan
- `mo_emergency_dialplan.xml` - Mobile Originated Emergency Call Dialplan
- `mt_dialplan.xml` - Mobile Terminated Call Dialplan

You can view the Dialplans from inside the Web UI.

Various variables are set by the TAS before the XML gets parsed, these variables are printed to the log at the start of the call with their current values and are very helpful when defining your own call logic.


Call Processing Flow
INVITE
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
English
MO
MT
Emergency
mo_dialplan.xml
mt_dialplan.xml
mo_emergency_dialplan.xml
Normalize numbers, translation pre-XML
Normalize B-Number, CF logic via Sh/defaults
Route to PSAP/GW
Ro enabled? not skipped by regex?
yes
no
CCR-I/CCR-U/CCR-T
Bridge to On-Net/SBC/Trunk (msrn override if set)
No answer/Busy?
yes
no



FreeSWITCH XML Dialplan Fundamentals

OmniTAS uses the same XML call routing system as the FreeSWITCH project, which allows for flexible call routing to meet your needs.

This section explains the core concepts and provides practical examples.

Basic Structure

A dialplan consists of **extensions** containing **conditions** and **actions**:

```
<extension name="description-of-what-this-does">
  <condition field="${variable}" expression="regex-pattern">
    <action application="app_name" data="parameters"/>
    <anti-action application="app_name" data="parameters"/>
  </condition>
</extension>
```

Extensions are evaluated in order from top to bottom. When a condition matches, its actions execute.

Conditions and Regex Matching

Conditions test variables against regular expressions. If the regex matches, actions execute; if not, anti-actions execute.

Basic exact match:

```
<condition field="${tas_destination_number}" expression="2222">
  <action application="log" data="INFO Calling voicemail access
number"/>
</condition>
```

Multiple number match:

```
<condition field="${tas_destination_number}"
expression="^(2222|3444|3445)$">
  <action application="log" data="INFO Calling special service"/>
</condition>
```

Pattern matching with capture groups:

```
<condition field="${tas_destination_number}" expression="^1(8[0-9]
{9})$">
  <!-- Matches 1 followed by 8 and 9 more digits -->
  <action application="log" data="INFO Matched toll-free: $1"/>
  <action application="bridge"
data="sofia/gateway/trunk/${tas_destination_number}"/>
</condition>
```

Prefix matching:

```
<condition field="${tas_destination_number}" expression="^00">
  <!-- Matches any number starting with 00 (international) -->
  <action application="log" data="INFO International call
detected"/>
</condition>
```

Range matching:

```
<condition field="${msisdn}" expression="^5551241[0-9]{4}$">
  <!-- Matches 55512410000 through 55512419999 -->
  <action application="log" data="INFO Subscriber in range"/>
</condition>
```

Actions vs Anti-Actions

Actions execute when a condition matches. **Anti-actions** execute when a condition does NOT match.

```
<condition field="${cli_withheld}" expression="true">
  <!-- Executes if CLI is withheld -->
  <action application="set"
data="effective_caller_id_number=anonymous"/>
  <action application="set"
data="origination_privacy=hide_number"/>

  <!-- Executes if CLI is NOT withheld -->
  <anti-action application="log" data="DEBUG CLI is normal"/>
  <anti-action application="set"
data="effective_caller_id_number=${msisdn}"/>
</condition>
```

The continue="true" Attribute

By default, when an extension's condition matches, the dialplan stops processing further extensions. The `continue="true"` attribute allows processing to continue to the next extension.

Without continue (default behavior):

```
<extension name="First-Check">
  <condition field="${tas_destination_number}"
expression="^(.*)$">
    <action application="log" data="INFO Processing call"/>
  </condition>
</extension>

<extension name="Never-Reached">
  <!-- This NEVER executes because the previous extension matched
-->
  <condition field="${tas_destination_number}"
expression="^(.*)$">
    <action application="log" data="INFO This won't print"/>
  </condition>
</extension>
```

With continue="true":

```
<extension name="Print-Vars" continue="true">
  <condition field="${tas_destination_number}"
expression="^(.*)$">
    <action application="info" data=""/>
  </condition>
</extension>

<extension name="Check-Balance" continue="true">
  <condition field="${hangup_case}"
expression="OUTGOING_CALL_BARRED">
    <action application="log" data="ERROR Insufficient balance"/>
    <action application="hangup" data="${hangup_case}"/>
  </condition>
</extension>

<extension name="Route-Call">
  <!-- This extension still gets evaluated -->
  <condition field="${tas_destination_number}"
expression="^(.*)$">
    <action application="bridge"
data="sofia/gateway/trunk/${tas_destination_number}"/>
  </condition>
</extension>
```

Use `continue="true"` for:

- Logging/debugging extensions
- Setting variables that apply to multiple scenarios
- Validation checks that don't route the call

Common Applications

call control

answer - Answer the call (send 200 OK)

```
<action application="answer" data=""/>
```

hangup - Terminate the call with a specific cause

```
<action application="hangup" data="NORMAL_CLEARING"/>
<action application="hangup" data="USER_BUSY"/>
<action application="hangup" data="NO_ANSWER"/>
```

bridge - Connect the call to another destination

```
<!-- Bridge to external gateway -->
<action application="bridge"
data="sofia/gateway/trunk/+12125551234"/>

<!-- Bridge to internal extension with codec preferences -->
<action application="bridge" data="{absolute_codec_string=AMR-
WB,AMR,PCMA}sofia/internal/sip:user@domain.com"/>

<!-- Bridge with timeout -->
<action application="bridge" data="
{originate_timeout=30}sofia/gateway/trunk/${tas_destination_number}"/
```

Variables and Channel Data

set - Set a channel variable

```
<action application="set" data="my_variable=my_value"/>
<action application="set" data="sip_h_X-Custom-
Header=CustomValue"/>
<action application="set"
data="effective_caller_id_number=anonymous"/>
```

unset - Remove a channel variable

```
<action application="unset" data="sip_h_P-Asserted-Identity"/>
```

export - Set variable and export to B-leg (bridged call)

```
<action application="export" data="sip_h_X-Account-Code=ABC123"/>
```

Media and Prompts

playback - Play an audio file

```
<action application="playback"
data="/sounds/en/us/callie/misc/8000/out_of_credit.wav"/>
<action application="playback"
data="$${base_dir}/sounds/custom_prompt.wav"/>
```

sleep - Pause for specified milliseconds

```
<action application="sleep" data="1000"/>  <!-- Sleep for 1 second
-->
```

echo - Echo audio back to caller (testing)

```
<action application="echo" data=""/>
```

conference - Place call into conference

```
<action application="conference"
data="room-${destination_number}@wideband"/>
```

voicemail (TAS ESL voicemail)

Voicemail is served by the TAS over an ESL outbound socket, not the FreeSWITCH `voicemail` application. Set the mode/mailbox/caller variables and hand the call to the socket. See **Voicemail** for the full flow.

```
<!-- Leave voicemail for mailbox (deposit) -->
<action application="set" data="tas_vm_mode=deposit"/>
<action application="set" data="tas_vm_mailbox=${msisdn}"/>
<action application="set"
data="tas_vm_caller=${effective_caller_id_number}"/>
<action application="socket" data="127.0.0.1:8084 async full"/>

<!-- Check voicemail (retrieval) -->
<action application="set" data="tas_vm_mode=retrieve"/>
<action application="set" data="tas_vm_mailbox=${msisdn}"/>
<action application="socket" data="127.0.0.1:8084 async full"/>
```

Logging and Debugging

log - Write to log file

```
<action application="log" data="INFO Processing call from
${msisdn}"/>
<action application="log" data="DEBUG Destination:
${tas_destination_number}"/>
<action application="log" data="ERROR Call failed with cause:
${hangup_cause}"/>
```

info - Dump all channel variables to log

```
<action application="info" data=""/>
```

Misc Applications

say - Text-to-speech number reading

```
<action application="say" data="en number iterated
${tas_destination_number}"/>
```

send_dtmf - Send DTMF tones

```
<action application="send_dtmf" data="1234#"/>
```

Practical Examples

Emergency Services Routing:

```
<extension name="Emergency-911">
  <condition field="${tas_destination_number}"
expression="^(911|112)$">
    <action application="log" data="ALERT Emergency call from
${msisdn}"/>
    <action application="answer" data=""/>
    <action application="playback"
data="/sounds/emergency_services_transfer.wav"/>
    <action application="bridge"
data="sofia/gateway/emergency_gw/${tas_destination_number}"/>
  </condition>
</extension>
```

Conditional Routing Based on Balance:

```
<extension name="Check-Credit">
  <condition field="${hangup_case}"
expression="OUTGOING_CALL_BARRED">
    <action application="answer" data=""/>
    <action application="playback"
data="/sounds/out_of_credit.wav"/>
    <action application="hangup" data="CALL_REJECTED"/>
  </condition>
</extension>
```

On-Net vs Off-Net Routing (with HOMER X-CID correlation):

```
<extension name="Route-Decision">
  <condition field="${on_net_status}" expression="true">
    <!-- On-net: route back through TAS for MT processing -->
    <action application="log" data="INFO Routing to on-net subscriber
    <action application="set" data="sip_copy_custom_headers=false"/>
    <action application="bridge" data="{sip_h_X-
CID=${original_call_id},absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU'}sofia/internal/${tas_destination_number}@${local_ip

    <!-- Off-net: route to SIP trunk -->
    <anti-action application="log" data="INFO Routing off-net"/>
    <anti-action application="set" data="sip_copy_custom_headers=fals
    <anti-action application="bridge" data="{sip_h_X-
CID=${original_call_id},absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU'}sofia/gateway/trunk/+${tas_destination_number}"/>
  </condition>
</extension>
```

> **Note:** `sip_h_X-CID` must be set as an inline bridge variable `{sip_h_X-CID=...}` not via `set`, because `sip_copy_custom_headers=false` strips `sip_h_` variables set with `set`. See HOMER Integration for full details on cross-leg correlation.

Anonymous Caller ID Handling:

```
<extension name="CLI-Privacy" continue="true">
  <condition field="${cli_withheld}" expression="true">
    <action application="set"
data="effective_caller_id_name=anonymous"/>
    <action application="set"
data="effective_caller_id_number=anonymous"/>
    <action application="set"
data="origination_privacy=hide_number"/>
  </condition>
</extension>
```

Voicemail on No Answer:

```
<extension name="Try-Bridge-Then-VM">
  <condition field="${tas_destination_number}"
expression="^(555124115\d{2})$">
    <action application="set" data="call_timeout=30"/>
    <action application="bridge"
data="sofia/internal/${tas_destination_number}@domain.com"/>

    <!-- If bridge fails, go to voicemail (TAS ESL voicemail; see
voicemail.md) -->
    <action application="log" data="INFO Bridge failed, routing to
voicemail"/>
    <action application="answer" data=""/>
    <action application="set" data="tas_vm_mode=deposit"/>
    <action application="set"
data="tas_vm_mailbox=${tas_destination_number}"/>
    <action application="set"
data="tas_vm_caller=${effective_caller_id_number}"/>
    <action application="socket" data="127.0.0.1:8084 async
full"/>
  </condition>
</extension>
```

Number Range Routing:

```
<extension name="Local-Numbers">
  <condition field="${tas_destination_number}" expression="^([2-
9]\d{2})$">
    <!-- 3-digit local extensions 200-999 -->
    <action application="log" data="INFO Local extension: $1"/>
    <action application="bridge"
data="sofia/internal/$1@pbx.local"/>
  </condition>
</extension>

<extension name="National-Numbers">
  <condition field="${tas_destination_number}"
expression="^555\d{7}$">
    <!-- National mobile numbers -->
    <action application="log" data="INFO National mobile call"/>
    <action application="bridge"
data="sofia/gateway/national_trunk/${tas_destination_number}"/>
  </condition>
</extension>

<extension name="International">
  <condition field="${tas_destination_number}"
expression="^00\d+$">
    <!-- International calls starting with 00 -->
    <action application="log" data="INFO International call"/>
    <action application="bridge"
data="sofia/gateway/intl_trunk/${tas_destination_number}"/>
  </condition>
</extension>
```

Further Documentation

For complete details on each application:

- **FreeSWITCH Dialplan Documentation:** https://freeswitch.org/confluence/display/FREESWITCH/Dialplan
- **FreeSWITCH mod_dptools:** https://freeswitch.org/confluence/display/FREESWITCH/mod_dptools (complete application reference)
- **Regular Expression Reference:** https://freeswitch.org/confluence/display/FREESWITCH/Regular+Expression

- **Channel Variables:** https://freeswitch.org/confluence/display/FREESWITCH/Channel+Variables

The FreeSWITCH wiki contains detailed documentation for every dialplan application, including all parameters and use cases.

Dialplan Variables

Variables set by the TAS in the XML dialplan logic:

Common Variables (All Call Types)

Initial Setup:

- `destination_number` - translated destination number
- `tas_destination_number` - translated destination number
- `effective_caller_id_number` - translated source number

Call Direction (CDR tagging):

- `tas_direction` - coarse call direction for CDRs: `MO`, `MT`, or `UNKNOWN`
- `tas_disposition` - the full routing disposition: `mo`, `mt`, `emergency`, or `conference`

These are set on the A-leg via ESL (`uuid_setvar`) in `Tas.Dialplan.process_call/1`, immediately after `Authorization.determine_disposition/1` computes the disposition. Without this, every call reaches FreeSWITCH as an inbound INVITE from the S-CSCF, so the built-in `direction` channel variable is always `inbound` and CDRs cannot distinguish MO from MT.

Direction is derived from the disposition (see `Tas.Dialplan.direction_label/1`):

Disposition	How it is determined (`Authorization`)	`tas_direction`
`:mo`	Source IP is an allowed CSCF	`MO`
`:mt`	Source IP is an allowed SBC (or local on-net B-leg)	`MT`
`:emergency`	Emergency URN / code (subscriber-originated)	`MO`
`:conference`	IMS conference-factory INVITE (subscriber-originated)	`MO`

> Internal re-entries (e.g. `:credit_exhausted` transfers) are **not** stamped, so they cannot clobber the direction set on the original call leg.

To surface these in the CDR, the corresponding columns must exist in the CDR backend (`cdr_sqlite` / `cdr_csv`) and be listed in the template — see the OmniCore ansible `applicationserver` role (`cdr_sqlite.conf.xml` template + the `Recreate cdr table schema` task). Stamping is A-leg only (matching the existing translated-number variables); the B-leg row is correlated via `bleg_uuid`.

Emergency Calls

- `hangup_case` - "none"
- `ims_private_identity` - private user identity
- `ims_public_identity` - public user identity
- `msisdn` - subscriber number (stripped of +)
- `imsi` - IMSI from private identity
- `ims_domain` - domain from private identity

MT Calls (Mobile Terminated)

- `ims_private_identity` - private user identity
- `ims_public_identity` - public user identity
- `msisdn` - subscriber number (stripped of +)

- `imsi` - IMSI from private identity
- `ims_domain` - domain from private identity
- `call_forward_all_destination` - CFA destination or "none"
- `call_forward_not_reachable_destination` - CFNRc destination
- `scscf_address` - S-CSCF address or "none"
- `scscf_domain` - S-CSCF domain or "none"
- `no_reply_timer` - timeout for no reply
- `hangup_case` - "none" or "UNALLOCATED_NUMBER"
- `msrn` - MSRN from PRN (if roaming) or forwarded number from SRI (if call forwarding active)
- `tas_destination_number` - Routing destination override (set to MSRN or forwarded number)

MO Calls (Mobile Originated)

- `hangup_case` - "none", "OUTGOING_CALL_BARRED", or "UNALLOCATED_NUMBER"
- `ims_private_identity` - private user identity
- `ims_public_identity` - public user identity
- `msisdn` - subscriber number (stripped of +)
- `imsi` - IMSI from private identity
- `ims_domain` - domain from private identity
- `allocated_time` - time allocated by OCS (if online charging enabled)
- `cli_withheld` - "true" or "false" string
- `on_net_status` - "true" or "false" string (whether destination is on-net)
- `original_call_id` - original A-leg SIP Call-ID from the UE (for HOMER X-CID correlation across B2BUA legs)
- `msrn` - MSRN for roaming subscribers (if applicable)
- `tas_destination_number` - MSRN override (if roaming)

Emergency Calling

Emergency calling is controlled through the `emergency_call_codes` configuration parameter and is automatically detected during call authorization.

Configuration

Configure the emergency call codes in your TAS configuration file:

Configuration parameters:

- `emergency_call_codes`: List of emergency service numbers to detect
- Common codes: "911" (US), "112" (EU), "000" (AU), "999" (UK), "sos"
- These codes are checked in addition to SIP emergency URNs (e.g., `<urn:service:sos>`)
- The system performs **exact match** comparison against the destination number

Example configuration values:

- US deployment: `["911", "933"]` - 911 for emergency, 933 for test
- European deployment: `["112", "999"]`
- Australian deployment: `["000", "106"]` - 000 for emergency, 106 for text relay
- Multi-region: `["911", "112", "000", "sos"]`

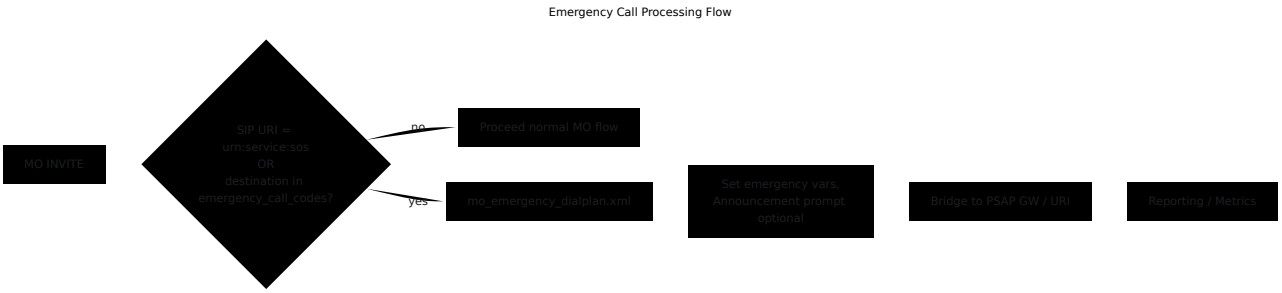
How Emergency Detection Works

The system checks two conditions:

1. **SIP URI Emergency Service URN**: Detects `<urn:service:sos>` or any URI containing "service:sos"
2. **Destination Number Match**: Compares `Caller-Destination-Number` against configured `emergency_call_codes`

If **either condition** is true, the call is classified as emergency.

Processing Flow



Call Flow Details:

1. Call arrives at TAS
2. Authorization module checks destination against emergency patterns
3. If emergency detected:
   - Call type is set to `:emergency`
   - `mo_emergency_dialplan.xml` template is used
   - OCS authorization is typically bypassed
   - Call is routed to PSAP gateway
4. Metrics are recorded with `call_type: emergency` label

Dialplan Routing

Define the routing for emergency calls in `priv/templates/mo_emergency_dialplan.xml`. This template determines how calls are routed to your PSAP (Public Safety Answering Point) gateway or SIP URI based on your market requirements.

Example emergency dialplan:

```
<extension name="Emergency-SOS">
  <condition field="${destination_number}"
expression="^(911|912|913|sos)$">
    <action application="log" data="ALERT Emergency call from
${msisdn}"/>
    <action application="answer" data=""/>
    <action application="bridge"
data="sofia/gateway/psap_gw/${destination_number}"/>
  </condition>
</extension>
```

Best Practices

- **Always include "sos"** in your emergency codes list for SIP URN compatibility
- **Include all local emergency numbers** for your jurisdiction (e.g., 911, 112, 000, 999)
- **Test emergency routing** regularly using the Call Simulator
- **Bypass OCS** for emergency calls to ensure they always connect (configured via `skipped_regex`)
- **Configure PSAP gateway** with high availability and redundancy
- **Monitor emergency call metrics** to ensure system reliability

IVR Menus

IVR menus allow callers to select options via DTMF digits before routing. Menus are defined as XML files and invoked from the dialplan using the `ivr` application.

Adding an IVR Menu

1. Create an XML file in `hosts/<environment>/group_vars/ivr_menus/`:

```
<include>
  <menu name="my_menu"
        greet-
long="$${base_dir}/sounds/en/us/callie/misc/8000/my_greeting.wav"
        greet-
short="$${base_dir}/sounds/en/us/callie/misc/8000/my_greeting.wav"
        invalid-sound="silence_stream://250"
        exit-sound=""
        timeout="5000"
        max-failures="1"
        max-timeouts="1"
        digit-len="1">

    <!-- Each digit can execute multiple actions in sequence -->
    <entry action="menu-exec-app" digits="1" param="playback
$${base_dir}/sounds/option1.wav"/>
    <entry action="menu-exec-app" digits="1" param="bridge
sofia/internal/100@$${domain}"/>

    <entry action="menu-exec-app" digits="2" param="playback
$${base_dir}/sounds/option2.wav"/>
    <entry action="menu-exec-app" digits="2" param="bridge
sofia/internal/200@$${domain}"/>

    <!-- Handle timeout and invalid input -->
    <entry action="menu-exec-app" digits="timeout" param="bridge
sofia/internal/default@$${domain}"/>
    <entry action="menu-exec-app" digits="invalid" param="bridge
sofia/internal/default@$${domain}"/>
  </menu>
</include>
```

2. Invoke the menu from your dialplan:

```
<extension name="my_ivr_extension">
  <condition field="destination_number" expression="^1234$">
    <action application="answer"/>
    <action application="ivr" data="my_menu"/>
  </condition>
</extension>
```

3. Deploy via Ansible

Menu Attributes

Attribute	Description
`name`	Menu identifier used in dialplan `ivr` application
`greet-long`	Audio file played on first entry
`greet-short`	Audio file played after invalid input
`timeout`	Milliseconds to wait for input
`max-failures`	Invalid inputs before triggering `invalid` action
`max-timeouts`	Timeouts before triggering `timeout` action
`digit-len`	Number of digits to collect

Entry Actions

Entries use `action="menu-exec-app"` to execute dialplan applications. Multiple entries with the same digit execute in sequence:

Entry Type	Description
`digits="1"`	Execute when caller presses 1
`digits="timeout"`	Execute when caller doesn't press anything
`digits="invalid"`	Execute when caller presses unrecognized digit

Common applications: `playback`, `bridge`, `transfer`, `hangup`

On-Net Mobile Originated call to an On-Net Mobile-Terminating Subscriber

When a subscriber calls another subscriber on your network (on-net call), the proper approach is to route the MO call back through the TAS for MT processing. This ensures the called party receives full MT call treatment including call forwarding, voicemail, MSRN routing for roaming, and all other subscriber services.

Why Route MO to MT?

Without MT processing (direct routing):

- Called party's call forwarding settings are ignored
- No voicemail on no-answer
- No MSRN routing for roaming subscribers
- Missing subscriber service logic

With MT processing (route back to TAS):

- Full call forwarding support (CFU, CFB, CFNRy, CFNRc)
- Voicemail on busy/no-answer
- MSRN routing for CS roaming subscribers
- Complete subscriber service experience
- Proper call state tracking for both parties

Implementation

The MO dialplan checks if the destination is on-net (served by your TAS), and if so, routes the call back to the TAS itself. The TAS receives this as a new MT call and processes it through the `mt_dialplan.xml` template.

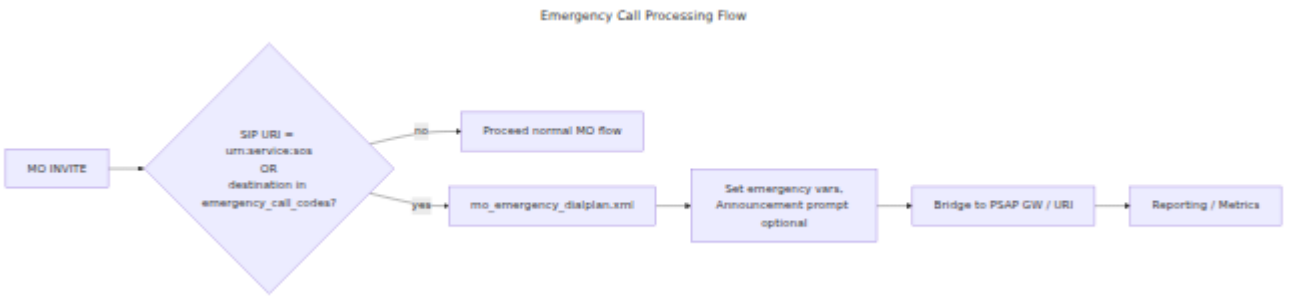
Example dialplan snippet:

```
<extension name="On-Net-Route">
  <condition field="${on_net_status}" expression="true">
    <action application="log" data="DEBUG On-Net MO call - Routing ba

    <!-- Clean up headers for internal routing -->
    <action application="set" data="sip_copy_multipart=false"/>
    <action application="set" data="sip_h_Request-Disposition=no-fork

    <!-- Route back to TAS (becomes MT call) -->
    <action application="bridge"
      data="{absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU',originate_retries=1,originate_timeout=60,sip_invite
/>
    <action application="hangup" data="" />
  </condition>
</extension>
```

Key parameters:

- `${sip_local_network_addr}` - TAS IP address (e.g., `10.179.3.60`)
- `${tas_destination_number}` - Called party's MSISDN
- `sip_invite_call_id=${sip_call_id}` - Preserves call-id for tracking
- `sip_copy_multipart=false` - Prevents multipart message copying
- `sip_h_Request-Disposition=no-fork` - Ensures sequential processing

Call Flow:



Important configuration:

- The TAS IP (e.g., `10.179.3.60`) must be in your `allowed_sbc_source_ips` configuration list
- This allows the TAS to receive calls from itself for MT processing

- Without this, the TAS will reject the call as coming from an unauthorized source

---

MSRN Usage for 2G/3G Roaming Subscribers

When a subscriber is roaming in a 2G/3G Circuit-Switched (CS) network, the TAS must obtain an MSRN (Mobile Station Roaming Number) to route the incoming call to the subscriber's current location. This section explains how MSRN retrieval and routing works.

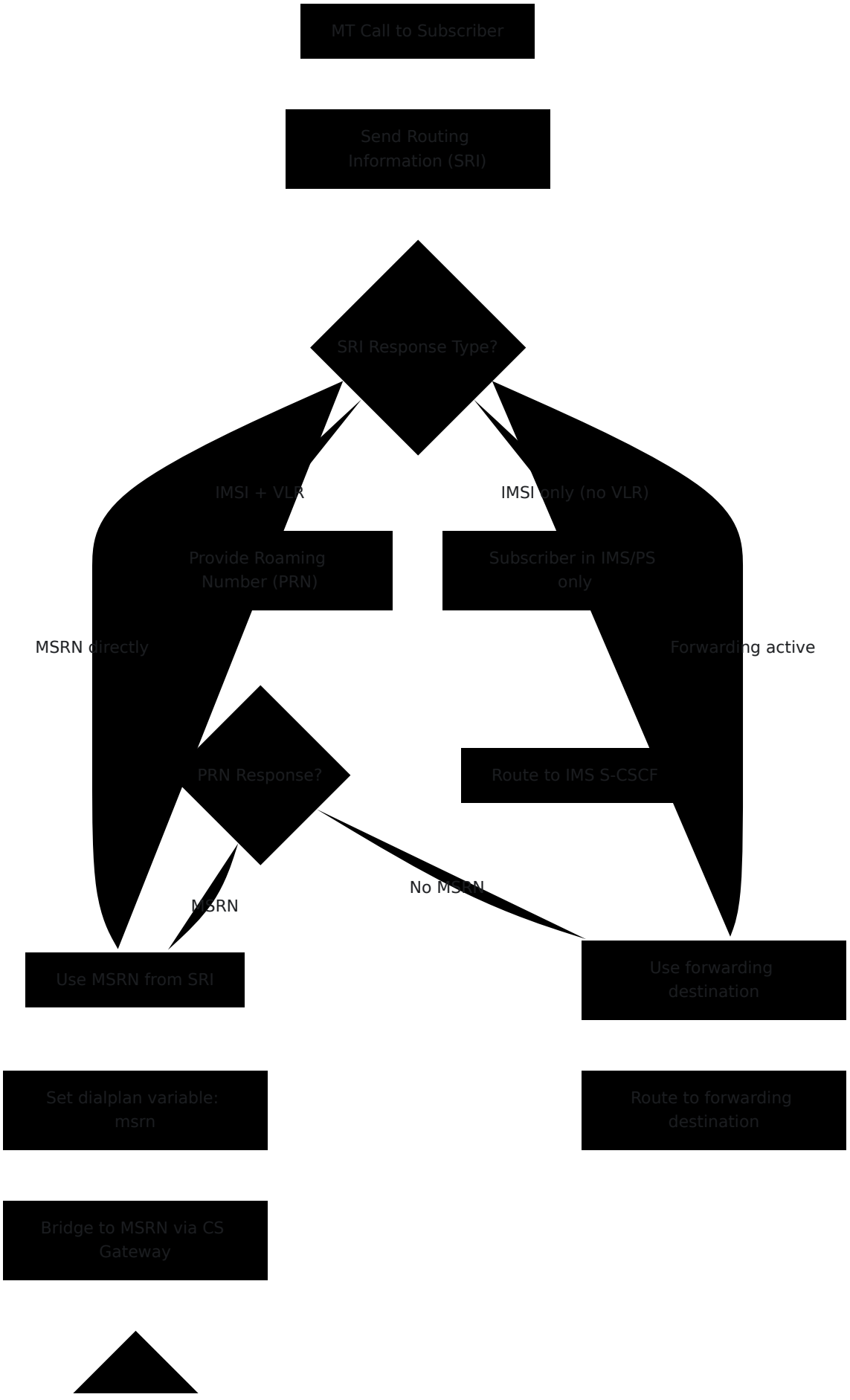
What is MSRN?

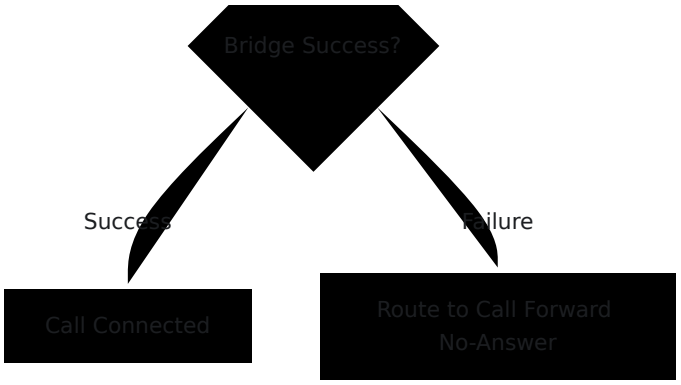
MSRN (Mobile Station Roaming Number) is a temporary routing number assigned by the visited network's VLR (Visitor Location Register) to route calls to a roaming subscriber. It acts as a temporary destination number that points to the subscriber's current location in the CS network.

MSRN Retrieval Flow

The TAS retrieves MSRN data via SS7 MAP (Mobile Application Part) protocol using a two-step process:

MSRN Retrieval for MT Calls to Roaming Subscribers





Implementation Details

Step 1: Send Routing Information (SRI)

The TAS queries the HLR via SS7 MAP to get routing information for the called subscriber.

SRI Response Scenarios:

1. **MSRN directly in SRI** - Roaming subscriber with MSRN already available
    - Response includes: MSISDN, GMSC, IMSI, and MSRN
    - Example MSRN: `61412345678` (Australian mobile number format)
2. **IMSI + VLR number** - Subscriber registered in CS network (requires PRN)
    - Response includes: MSISDN, GMSC, IMSI, and MSC/VLR number
    - Indicates subscriber is in CS network but MSRN must be requested
3. **IMSI only (no VLR)** - Subscriber not in CS network (IMS/PS only)
    - Response includes: MSISDN, GMSC, IMSI
    - Indicates subscriber is registered in IMS/4G only, not in CS network
4. **Call forwarding active** - SRI returns forwarding information
    - Response includes forwarding reason (unconditional, busy, no-reply, not-reachable)
    - Response includes forwarded-to number

Step 2: Provide Roaming Number (PRN) - If Needed

If SRI returns IMSI + VLR but no MSRN, the TAS sends a PRN request to the VLR to obtain the MSRN.

The VLR allocates a temporary MSRN from its pool and returns it to the TAS. This MSRN is valid only for this specific call setup.

Example PRN Response: MSRN `61412345678`

Dialplan Variable: `msrn`

Once the MSRN is retrieved via SS7 MAP, it's set as a dialplan variable that can be used in the MT dialplan.

Variable: `${msrn}`

- **Type:** String (E.164 number without leading +)
- **Example:** `"61412345678"` (Australian mobile format)
- **Usage:** Route calls to CS roaming subscribers
- **Set by:** HLR data retrieval process during MT call processing

Routing to MSRN in mt_dialplan.xml

The MSRN variable is used in the MT dialplan template to route calls to roaming subscribers.

Dialplan logic:

1. **Check for MSRN**: Extension checks if `msrn` variable is set (contains digits)
2. **Set timeout parameters**:
   - Progress timeout: 10 seconds to receive early media
   - Bridge answer timeout: Uses subscriber's configured no-reply timer
3. **Bridge to MSRN**: Route call to MSRN via CS gateway
   - Uses `ignore_early_media=ring_ready` for consistent ringback
   - Codec preference: AMR (mobile), PCMA/PCMU (wireline)
   - Gateway: `sofia/gateway/CS_Gateway/+${msrn}`
4. **Fallback on failure**: If bridge fails, route to call forwarding destination

Example dialplan snippet:

```
<extension name="Route-to-CS-MSRN" continue="false">
  <condition field="msrn" expression="^(\d+)$">
    <!-- Configure timeouts -->
    <action application="set" data="progress_timeout=10" />
    <action application="set" data="bridge_answer_timeout=${no_reply_

    <!-- Bridge to MSRN via CS gateway -->
    <action application="bridge"
      data="
{ignore_early_media=ring_ready,absolute_codec_string='AMR,PCMA,PCMU',
/>

    <!-- Fallback to voicemail/call forwarding -->
    <action application="bridge"
      data="sofia/internal/${call_forward_not_reachable_destination}@
  </condition>
</extension>
```

Key Points

1. **MSRN is temporary** - Valid only for the duration of the call setup
2. **CS network only** - MSRN is used for 2G/3G roaming, not VoLTE/IMS roaming
3. **Priority in MT flow** - MSRN check happens before standard IMS routing
4. **Fallback to forwarding** - If MSRN bridge fails, routes to call forwarding destination
5. **HLR overrides Sh** - MSRN from HLR takes precedence over Sh subscriber data

Configuration

SS7 MAP integration must be enabled in the TAS configuration:

Required settings:

- **enabled**: Set to `true` to enable SS7 MAP queries
- **http_map_server_url_base**: URL of your SS7 MAP gateway (e.g., `"http://10.1.1.100:5001"`)

- **gmsc**: Gateway MSC number for SRI/PRN requests (e.g., `"61400000000"`)
- **timeout_ms**: Query timeout in milliseconds (default: 5000ms)

See **SS7 MAP Documentation** for complete configuration details.

Call Forwarding Data Usage

Call forwarding settings determine how calls are routed when the primary destination is unavailable. The TAS retrieves call forwarding data from two sources: the Sh interface (HSS) and SS7 MAP (HLR), with HLR data taking precedence.

Call Forwarding Types

The system supports four types of call forwarding:

Forwarding Type	Variable	When Act
Call Forward Unconditional (CFU)	`call_forward_all_destination`	Always forwa all calls immediately
Call Forward Busy (CFB)	`call_forward_not_reachable_destination`	Subscriber's busy
Call Forward No Reply (CFNRy)	`call_forward_not_reachable_destination`	Subscriber d answer withi timeout
Call Forward Not Reachable (CFNRc)	`call_forward_not_reachable_destination`	Subscriber is unreachable/

Data Sources

1. Sh Interface (HSS)

Static configuration stored in the HSS subscriber profile.

The TAS retrieves call forwarding settings from the HSS via Sh interface during call processing. These are the provisioned/default settings for the subscriber.

Example retrieved data:

- `call_forward_all_destination`: CFU destination (e.g., `"61412345678"`)
- `call_forward_not_reachable_destination`: CFB/CFNRy/CFNRc destination (e.g., `"61487654321"`)
- `no_reply_timer`: Seconds before CFNRy triggers (e.g., `"20"`)

2. SS7 MAP (HLR)

Real-time data from the HLR, which may differ from HSS if subscriber changed settings via USSD/MMI codes (e.g., dialing `*21*` codes).

The TAS queries the HLR via SS7 MAP during call setup to get the current/active forwarding settings.

HLR forwarding response includes:

- **forwarded_to_number**: The destination number for forwarding (e.g., `"61412345678"`)
- **reason**: Forwarding type (unconditional, busy, no-reply, not-reachable)
- **notification flags**: Whether to notify calling party, forwarding party, etc.

Mapping to dialplan variables:

- If reason is **unconditional** → Sets `call_forward_all_destination`
- If reason is **busy, no-reply, or not-reachable** → Sets `call_forward_not_reachable_destination`

Variable Merging Priority

HLR data overrides Sh data when both are present.

The TAS retrieves subscriber data from both sources during MT call processing:

1. First, retrieves static configuration from HSS via Sh interface
2. Then, queries HLR via SS7 MAP for real-time settings
3. Merges the data, with HLR values taking precedence over Sh values

This ensures that recent subscriber changes (via USSD codes) are respected even if HSS hasn't been updated yet.

Dialplan Variables

Available in MT calls:

Variable	Type	Example	D
`call_forward_all_destination`	String	`"61412345678"`	CFU num
`call_forward_not_reachable_destination`	String	`"61487654321"`	CFB dest
`no_reply_timer`	String	`"20"`	Tim secc CFN

Default values:

- If not configured: `"none"` (string)
- Check for presence: Use regex `^(?!none$).*` to match any value except "none"

Call Forwarding in mt_dialplan.xml

Example 1: Call Forward Unconditional (CFU)

Routes ALL incoming calls immediately to the forwarding destination. The forwarding destination is typically an off-net number, so it uses an external gateway.

Gateway used: `sofia/gateway/ExternalSIPGateway` (your PSTN/interconnect gateway)

Template example:

```
<extension name="Check-Call-Forward-All">
  <condition field="${call_forward_all_destination}" expression="^(?!
    <action application="log" data="INFO Call Forward All Set to redi

    <!-- Set History-Info header for call forwarding -->
    <action application="set" data="sip_h_History-Info=<sip:${destina

    <!-- Mark call-id to indicate call forwarding type -->
    <action application="set" data="sip_call_id=${sip_call_id};CALL_F

    <!-- Bridge to off-net forwarding destination -->
    <action application="bridge"
      data="{absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU',originate_retries=1,originate_timeout=60}sofia/gate
/>
  </condition>
</extension>
```

Key points:

- Uses external gateway because forwarding is typically to off-net number
- Marks call-id with `;CALL_FORWARD_UNCONDITIONAL` for tracking
- Sets `History-Info` header to identify original called number
- Example: Subscriber `61412345678` has CFU to `61487654321` - all calls immediately forwarded

Example 2: Call Forward No Reply/Not Reachable

Used as fallback when bridge to primary destination fails (subscriber doesn't answer, is busy, or unreachable).

Example dialplan snippet:

```
<!-- After bridge to MSRN or IMS fails... -->
<action application="log" data="INFO Failed to bridge Call - Routing

<!-- Set History-Info to indicate forwarding -->
<action application="set" data="sip_h_History-Info=<sip:${destinatior

<!-- Route to forwarding destination -->
<action application="bridge"
  data="
{absolute_codec_string='AMR,PCMU,PCMA',originate_timeout=65}sofia/gat
/>
```

Example scenario:

- Subscriber `61412345678` has CFNRy to voicemail number `61487654321`
- Incoming call attempts to reach subscriber
- No answer after 20 seconds (no_reply_timer)
- Call forwarded to `61487654321` with History-Info header preserving original destination

History-Info Header

The `History-Info` SIP header tracks call forwarding:

```
<action application="set" data="sip_h_History-Info=
<sip:${destination_number}@${ims_domain}>;index=1.1" />
```

Purpose:

- Indicates the call was originally for `${destination_number}`
- Allows downstream systems to identify forwarded calls
- Used by voicemail systems to deposit to correct mailbox

Example in voicemail routing:

```
<extension name="Voicemail Route" continue="false">
  <condition field="${tas_destination_number}"
expression="^(555121|555122)$">
    <!-- Extract the phone number from the History Info -->
    <action application="set"
data="history_info_value=${sip_i_history_info}"/>
    <action application="log" data="DEBUG Called Voicemail Deposit
Number for ${history_info_value}" />

    <!-- Deposit voicemail to ORIGINAL called party, not voicemail
number
         (TAS ESL voicemail; see voicemail.md) -->
    <action application="set" data="tas_vm_mode=deposit"/>
    <action application="set"
data="tas_vm_mailbox=${history_info_value}"/>
    <action application="set"
data="tas_vm_caller=${effective_caller_id_number}"/>
    <action application="socket" data="127.0.0.1:8084 async
full"/>
  </condition>
</extension>
```

How it works:

- Voicemail service numbers: `555121`, `555122` (generic short codes)
- When call is forwarded to voicemail, History-Info contains original destination
- Voicemail system extracts original number from History-Info header
- Voicemail deposited to original called party's mailbox, not voicemail service number

Best Practices

1. **Always check for "none"** - Use regex `^(?!none$).*` to avoid routing to literal string "none"
2. **Set History-Info** - Always set when forwarding for proper call tracking
3. **Use continue_on_fail** - Allow fallback to forwarding if primary route fails
4. **Adjust CLI format** - National vs international prefix formatting (see Caller ID section)

5. **Test forwarding loops** - Ensure forwarding destinations don't create routing loops

---

Caller ID (CLI) Management

The TAS manages Calling Line Identification (CLI) presentation and formatting throughout the call flow, handling privacy requests, prefix normalization, and network-specific formatting requirements.

CLI Variables

Core CLI variables in dialplans:

Variable	Usage	Example
`msisdn`	Subscriber's number (no +)	`"61412345678"`
`effective_caller_id_number`	Displayed caller number	`"+61412345678"` or `"anonymous"`
`effective_caller_id_name`	Displayed caller name	`"+61412345678"` or `"anonymous"`
`origination_caller_id_number`	CLI for outbound leg	`"+61412345678"`
`caller_id_number`	Standard FreeSWITCH CLI var	`"+61412345678"`
`sip_from_user`	SIP From header user part	`"0412345678"` or `"+61412345678"`
`cli_withheld`	Privacy flag	`"true"` or `"false"` (string)
`origination_privacy`	Privacy setting	`"hide_number"`

CLI Privacy (Withheld/Anonymous)

Detection Methods

The TAS detects CLI privacy requests through three methods:

1. Blocked Prefix in Dialed Number

Subscriber dials a prefix before the destination number to block their caller ID.

Common prefixes:

- `*67` - North American standard
- `#31#` - European/GSM standard
- `1831` - Alternative format

The TAS checks if the dialed number starts with any configured blocked CLI prefix. If detected, the `cli_withheld` variable is set to `"true"`.

Example: Subscriber dials `*67555 1234` - the `*67` prefix is detected and removed, call proceeds to `5551234` with CLI withheld.

2. Anonymous in From Header

The user equipment (UE) sets the caller name to "anonymous" in the SIP From header.

The TAS checks the `Caller-Orig-Caller-ID-Name` field (case-insensitive) for the string "anonymous". If found, `cli_withheld` is set to `"true"`.

3. SIP Privacy Headers

The S-CSCF may set `Privacy: id` headers in the SIP INVITE, which are honored by the dialplan.

Dialplan Implementation

The dialplan checks the `cli_withheld` variable and sets all CLI-related variables accordingly.

Example dialplan snippet:

```
<extension name="Manage-Caller-ID" continue="true">
  <condition field="${cli_withheld}" expression="true">
    <!-- CLI is withheld - set to anonymous -->
    <action application="log" data="DEBUG CLI withheld detected"
/>
    <action application="set"
data="effective_caller_id_name=anonymous" />
    <action application="set"
data="effective_caller_id_number=anonymous" />
    <action application="set"
data="origination_caller_id_number=anonymous" />
    <action application="set"
data="origination_privacy=hide_number" />

    <!-- CLI is NOT withheld - use normal MSISDN -->
    <anti-action application="log" data="DEBUG CLI is normal (not
withheld)" />
    <anti-action application="set"
data="effective_caller_id_number=${msisdn}" />
  </condition>
</extension>
```

Note: This extension uses `continue="true"` so call processing continues to routing extensions even after CLI is set.

CLI Format: National vs International

Different destinations may require different CLI formats depending on your network's requirements.

Example: National Format

For national calls within your country, you may need to present CLI without the country code.

Example dialplan snippet (Australian mobile network):

```
<extension name="Outgoing-Call-CLI-National" continue="true">
  <condition field="${msisdn}" expression="^61(.*)$">
    <action application="log" data="Setting source CLI to $1 for
national" />
    <action application="set"
data="effective_caller_id_number=$1"/>  <!-- 0412345678 -->
    <action application="set" data="effective_caller_id_name=$1"/>
    <action application="set" data="sip_from_user=$1"/>
    <action application="set" data="sip_cid_type=pid"/>
  </condition>
</extension>
```

How it works:

- Regex `^61(.*)$` captures everything after country code `61`
- Input: `msisdn="61412345678"` → Output: `$1="412345678"` or `"0412345678"`
- Presents CLI in national format for domestic calls

Example: International Format

For international calls, present CLI in full E.164 format with `+` prefix.

Example dialplan snippet:

```
<extension name="Outgoing-Call-CLI-International" continue="true">
  <condition field="${tas_destination_number}"
expression="^61(.*)$">
    <action application="log" data="Call is to national" />

    <!-- Anti-action runs when destination is NOT national -->
    <anti-action application="log" data="Setting source CLI for
international" />
    <anti-action application="set"
data="effective_caller_id_number=+${msisdn}"/>  <!-- +61412345678
-->
    <anti-action application="set"
data="effective_caller_id_name=+${msisdn}"/>
    <anti-action application="set"
data="sip_from_user=+${msisdn}"/>
    <anti-action application="set" data="sip_cid_type=pid"/>
  </condition>
</extension>
```

How it works:

- Condition checks if destination starts with national prefix (e.g., `61` for Australia)
- `<anti-action>` executes when condition does NOT match (international call)
- Adds `+` prefix for full E.164 format on international calls

CLI Format for Call Forwarding

When routing to call forwarding destinations, you may need to adjust CLI format depending on whether forwarding to on-net or off-net numbers.

Example: Adjusting CLI prefix for call forwarding

```
<!-- Adjust CLI format if needed for forwarding destination -->
<action application="set"
data="effective_caller_id_number=${effective_caller_id_number:3}"/>
<action application="set"
data="effective_caller_id_name=${effective_caller_id_name:3}"/>
```

String Slicing: `${variable:N}` removes first N characters

- Input: `effective_caller_id_number="+61412345678"` with `:3` → Output: `"412345678"`
- Input: `effective_caller_id_number="+61412345678"` with `:1` → Output: `"61412345678"`

Use cases:

- Remove `+` for national forwarding: Use `:1`
- Remove country code for local format: Use appropriate offset (`:3` for `+61`, `:2` for `+1`, etc.)

SIP P-Asserted-Identity (PAI)

The `sip_cid_type=pid` setting controls how caller ID is presented:

```
<action application="set" data="sip_cid_type=pid"/>
```

Effect:

- Sets SIP `P-Asserted-Identity` header with caller information
- Used for trusted network caller ID assertion
- Standard for IMS networks

Removing Proprietary Headers

To prevent leaking internal network information, dialplans should remove proprietary or internal headers before routing calls off-net.

Example: Cleaning headers before external routing

```
<action application="set" data="sip_copy_multipart=false"/>
<action application="set" data="sip_copy_custom_headers=false"/>
<action application="unset" data="sip_h_P-Internal-Correlation-
ID"/>
<action application="unset" data="sip_h_P-Access-Network-Info"/>
<!-- Add more vendor-specific or internal headers as needed -->
```

Purpose:

- Prevents internal routing data from reaching external networks
- Removes vendor-specific proprietary headers
- Privacy and security best practice
- Reduces SIP message size

Common headers to remove:

- Internal correlation/tracking IDs
- Access network information (may reveal network topology)
- Vendor-specific P-headers
- Custom application headers meant for internal use only

Best Practices

1. **Use `continue="true"` for CLI extensions** - Allows multiple CLI formatting rules
2. **Set `sip_cid_type=pid`** - Required for IMS network compliance
3. **Test CLI withholding** - Verify `*67` and `#31#` prefixes work
4. **Format per destination** - National vs international CLI formatting
5. **Remove proprietary headers** - Prevent internal data leakage
6. **Handle anonymous gracefully** - Both display and routing should work with anonymous CLI

Bridging to Gateways

The TAS bridges calls to external gateways (IMS core, PSTN, etc.) using FreeSWITCH's `bridge` application with carefully configured parameters for codec negotiation, timeout handling, and retry logic.

Gateway Configuration

Gateways are configured as SIP trunks to external systems. The TAS uses a single SIP interface for all traffic, with different gateways defined for different destinations.

Example gateway configuration:

```
<gateway name="CS_Gateway">
  <param name="proxy" value="10.1.1.100:5060"/>
  <param name="register" value="false"/>
  <param name="caller-id-in-from" value="true"/>
  <param name="extension-in-contact" value="true"/>
</gateway>
```

See **Configuration Guide** for complete gateway setup.

Bridge Syntax

Calls are bridged to gateways using the following syntax:

Basic syntax:

```
<action application="bridge"
data="sofia/gateway/GATEWAY_NAME/DESTINATION_NUMBER" />
```

With parameters:

```
<action application="bridge" data="
{param1=value1,param2=value2}sofia/gateway/GATEWAY_NAME/DESTINATION_N
/>
```

Where `GATEWAY_NAME` is the name of the gateway defined in your configuration (e.g., `IMS_Core`, `PSTN_Primary`, `International_Gateway`).

Bridge Parameters

Codec Selection

`absolute_codec_string` - Prioritized codec list for negotiation:

```
<action application="bridge" data="
{absolute_codec_string='AMR,PCMA,PCMU'}sofia/gateway/IMS_Gateway/+${m
/>
```

Codec priority order:

1. **AMR** (Adaptive Multi-Rate) - Mobile-optimized, preferred for cellular
2. **PCMA** (G.711 a-law) - Fixed-line standard in Europe/international
3. **PCMU** (G.711 μ-law) - Fixed-line standard in North America

Template usage: `priv/templates/mt_dialplan.xml:80`, `mo_dialplan.xml:124`, `mo_dialplan.xml:202`

Timeout Configuration

`originate_timeout` - Maximum seconds to wait for answer (includes ringing):

```
<action application="set" data="originate_timeout=60"/>
<action application="bridge" data="
{originate_timeout=60}sofia/gateway/CS_Gateway/+${msisdn}" />
```

`progress_timeout` - Seconds to wait for 180/183 (early media/ringing):

```
<action application="set" data="progress_timeout=10" />
```

`bridge_answer_timeout` - Seconds to wait for 200 OK after ringing starts:

```
<action application="set"
data="bridge_answer_timeout=${no_reply_timer}" />
```

`leg_progress_timeout` - Per-leg progress timeout:

```
<action application="set"
data="leg_progress_timeout=${no_reply_timer}" />
```

Template example: `priv/templates/mt_dialplan.xml:73-76`

```
<action application="set" data="progress_timeout=10" />
<!-- How long do we wait between the INVITE and a 200 OK
(Including RINGING) -->
<action application="set"
data="bridge_answer_timeout=${no_reply_timer}" />
<action application="set"
data="leg_progress_timeout=${no_reply_timer}" />
```

Variable: `${no_reply_timer}` comes from subscriber data (typically 20-30 seconds)

Retry and Failure Handling

`originate_retries` - Number of retry attempts:

```
<action application="bridge" data="
{originate_retries=1}sofia/gateway/CS_Gateway/+${msisdn}" />
```

`continue_on_fail` - Continue dialplan execution after bridge failure:

```
<action application="set" data="continue_on_fail=true" />
<action application="bridge" data="
{continue_on_fail=true}sofia/gateway/CS_Gateway/+${msisdn}" />
<!-- Subsequent actions execute if bridge fails -->
<action application="log" data="INFO Bridge failed - routing to
voicemail" />
```

`hangup_after_bridge` - Hangup A-leg when B-leg hangs up:

```
<action application="set" data="hangup_after_bridge=true"/>
```

Early Media Handling

`ignore_early_media` - Control early media behavior:

```
<action application="set" data="ignore_early_media=ring_ready" />
<action application="bridge" data="
{ignore_early_media=ring_ready}sofia/gateway/CS_Gateway/+${msisdn}"
/>
```

Options:

- `ring_ready` - Generate local ringback, ignore remote early media
- `true` - Completely ignore early media
- `false` (default) - Pass through early media (announcements, tones)

Why use `ring_ready`? - Prevents caller from hearing network announcements or tones from remote network

Template example: `priv/templates/mt_dialplan.xml:78-79`

```
<action application="set" data="ignore_early_media=ring_ready" />
<action application="bridge" data="
{ignore_early_media=ring_ready,...}sofia/gateway/CS_Gateway/+${msrn}"
/>
```

On-net vs Off-net caller handling:

```
<extension name="Route-to-IMS-Sub-Early-Media" continue="true">
  <condition field="${on_net_caller}" expression="true">
    <!-- On-net caller - use ring_ready -->
    <action application="log" data="INFO On-net caller
${effective_caller_id_number} - using
ignore_early_media=ring_ready"/>
    <action application="set"
data="ignore_early_media=ring_ready"/>

    <!-- Off-net caller - provide instant ringback -->
    <anti-action application="log" data="INFO Off-net caller
${effective_caller_id_number} - setting instant ringback"/>
    <anti-action application="set" data="instant_ringback=true"/>
    <anti-action application="set" data="ringback=${fr-ring}"/>
    <anti-action application="set" data="transfer_ringback=${fr-
ring}"/>
  </condition>
</extension>
```

Note: The `${on_net_caller}` variable is set based on your network's subscriber numbering plan. You can also use regex patterns to match your specific number ranges.

Caller ID Parameters

`sip_cid_type=pid` - Use P-Asserted-Identity for caller ID:

```
<action application="set" data="sip_cid_type=pid" />
<action application="bridge" data="
{sip_cid_type=pid}sofia/gateway/CS_Gateway/+${msisdn}" />
```

Common Bridge Patterns

Pattern 1: Route to IMS Subscriber via IMS Domain

Route MT call to IMS subscriber by sending to the IMS domain (S-CSCF will resolve and route).

Template example:

```
<extension name="Route-to-IMS-Sub" continue="false">
  <condition field="destination_number" expression="^(.*)$">
    <action application="set" data="continue_on_fail=true" />
    <action application="set" data="hangup_after_bridge=true"/>
    <action application="set" data="progress_timeout=10" />

    <!-- How long do we wait between the INVITE and a 200 OK (Includi
    <action application="set" data="bridge_answer_timeout=${no_reply_
    <action application="set" data="leg_progress_timeout=${no_reply_t

    <!-- Send call to IMS domain (S-CSCF resolves) -->
    <action application="set" data="ignore_early_media=ring_ready" />
    <action application="set" data="sip_cid_type=pid" />

    <action application="bridge"
      data="{absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU',ignore_early_media=ring_ready,continue_on_fail=true
/>

    <!-- Fallback to call forwarding if bridge fails -->
    <action application="log" data="INFO Failed to bridge Call - Rout
    <action application="set" data="sip_h_History-Info=<sip:${destina
    <action application="set" data="sip_h_Diversion=<sip:${destinatio

    <!-- Route to off-net gateway for call forwarding -->
    <action application="bridge"
      data="{absolute_codec_string='AMR-WB,AMR,PCMU,PCMA',originate_t
  </condition>
</extension>
```

Key points:

- Routes to `${msisdn}@${ims_domain}` (e.g., `5551234567@ims.mnc001.mcc001.3gppnetwork.org`)
- IMS core (S-CSCF/I-CSCF) handles final routing to subscriber
- `ignore_early_media=ring_ready` provides consistent ringback
- On failure, uses external gateway for off-net call forwarding
- Sets `History-Info` and `Diversion` headers for call forwarding tracking

Pattern 2: Route to MSRN (CS Roaming)

Route to roaming subscriber via CS network:

Template: `priv/templates/mt_dialplan.xml:67-80`

```
<extension name="Route-to-CS-MSRN" continue="false">
  <condition field="msrn" expression="^(\d+)$">
    <action application="set" data="continue_on_fail=true" />
    <action application="set" data="hangup_after_bridge=true"/>
    <action application="set" data="progress_timeout=10" />
    <action application="set" data="bridge_answer_timeout=${no_reply_
    <action application="set" data="leg_progress_timeout=${no_reply_t

    <!-- Send call to MSRN via Gateway -->
    <action application="set" data="ignore_early_media=ring_ready" />
    <action application="set" data="sip_cid_type=pid" />
    <action application="bridge"
      data="
{ignore_early_media=ring_ready,absolute_codec_string='AMR,PCMA,PCMU',
/>
  </condition>
</extension>
```

Pattern 3: On-Net Routing (MO to MT via TAS)

When a subscriber calls another on-net subscriber, the call must be routed back to the TAS for full MT processing. This pattern is critical for ensuring that on-net calls receive the same service treatment as external MT calls.

Why this pattern is required:

Without routing back to TAS, on-net calls would bypass MT processing entirely, meaning:

- Call forwarding settings would not be honored
- Voicemail-on-busy/no-answer would not work
- MSRN routing for roaming subscribers would fail
- Subscriber service logic would be skipped
- Call tracking and CDRs would be incomplete

By routing the MO call back to the TAS as a new MT call, the destination subscriber gets full service treatment.

Template example:

```
<extension name="On-Net-Route">
  <condition field="${on_net_status}" expression="true">
    <action application="log" data="DEBUG On-Net MO call - Routing ba

    <!-- Clean up headers for internal routing -->
    <action application="set" data="sip_copy_multipart=false"/>
    <action application="set" data="sip_h_Request-Disposition=no-fork

    <!-- Route back to TAS (becomes MT call) -->
    <action application="bridge"
      data="{absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU',originate_retries=1,originate_timeout=60,sip_invite
/>
    <action application="hangup" data="" />
  </condition>
</extension>
```

How it works:

1. **MO Call Arrives**: Subscriber A calls Subscriber B (both on-net)
2. **Check On-Net Status**: TAS determines destination is on-net via `${on_net_status}` variable
3. **Route to TAS**: Bridge to `sofia/internal/${tas_destination_number}@${sip_local_network_addr}`
   - Uses TAS's own IP address as destination
   - Preserves original call-id for tracking
4. **MT Processing**: TAS receives call as new MT call and processes `mt_dialplan.xml`
   - Checks call forwarding settings (CFU, CFB, CFNRy, CFNRc)
   - Queries for MSRN if subscriber is roaming
   - Routes to IMS domain or forwards appropriately
5. **Complete Service**: Destination subscriber gets full MT treatment

Key points:

- Routes to `${sip_local_network_addr}` (TAS IP address, e.g., `10.179.3.60`)
- Call is re-processed as MT call to destination subscriber
- Preserves call-id with `sip_invite_call_id` parameter for end-to-end tracking
- Enables all MT features: call forwarding, voicemail, MSRN routing, subscriber services
- Proper call state tracking and CDR generation for both parties
- On-net calls get identical service treatment to external MT calls
- TAS IP must be in `allowed_sbc_source_ips` configuration list

Variable: `${on_net_status}` is set to `"true"` when the destination number is served by your TAS. This is determined during MO call authorization by checking if the destination MSISDN exists in your subscriber database.

Pattern 4: Off-Net Routing (MO to PSTN/External)

Route MO call to external PSTN, interconnect, or other external network via gateway.

Gateway used: `sofia/gateway/ExternalSIPGateway` or `sofia/gateway/PSTN_Gateway`

Template example:

```
<extension name="Outgoing-Call-Off-Net">
  <condition field="${tas_destination_number}" expression="^(.*)$">
    <action application="log" data="Sending call off-net" />

    <!-- Clean up headers before external routing -->
    <action application="set" data="sip_copy_multipart=false"/>

    <!-- Set call event hooks for CDR/billing -->
    <action application="set" data='api_body=caller=${msisdn}&called=
    <action application="set" data='api_on_answer=curl http://localhc
${api_body}'/>
    <action application="set" data='api_body=caller=${msisdn}&called=
    <action application="set" data='api_hangup_hook=curl http://local
${api_body}'/>

    <!-- Set P-Asserted-Identity for trusted network -->
    <action application="set" data="sip_h_Request-Disposition=no-fork
    <action application="set" data="sip_h_P-Asserted-Identity=<sip:${

    <action application="set" data="hangup_after_bridge=true"/>
    <action application="set" data="continue_on_fail=true"/>

    <!-- Bridge to external PSTN/interconnect gateway -->
    <action application="set" data="used_gateway=ExternalSIPGateway"/
    <action application="bridge"
      data="{absolute_codec_string='AMR-
WB,AMR,PCMA,PCMU',originate_retries=1,originate_timeout=60,sip_invite
/>

    <!-- If bridge fails, provide error treatment -->
    <action application="answer" data="" />
    <action application="log" data="INFO Bridge failed with SIP code
    <action application="sleep" data="500"/>
    <action application="transfer" data="${last_bridge_proto_specific
  </condition>
</extension>
```

Key points:

- Uses `sofia/gateway/ExternalSIPGateway` for external routing
- Sets `P-Asserted-Identity` for caller ID on trusted interconnect

- Call event hooks for CDR/billing tracking
- `continue_on_fail=true` allows error handling
- On failure, transfers to error announcement based on SIP code
- `used_gateway` variable for reporting/troubleshooting

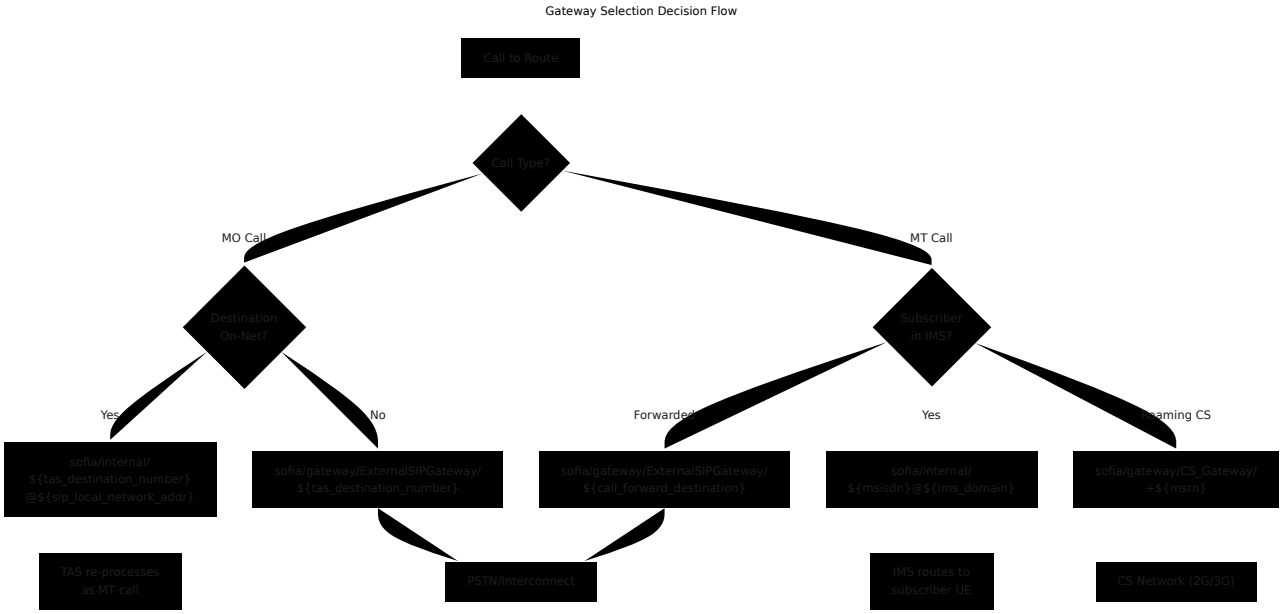
Common off-net gateways:

- `ExternalSIPGateway` - Primary PSTN trunk/interconnect
- `BackupSIPGateway` - Backup PSTN trunk (for failover)
- `International_GW` - International calling gateway
- `Emergency_GW` - Emergency services gateway

Gateway Routing Strategy

All calls are routed via gateways. Understanding which gateway to use for different call scenarios is critical for proper call routing:

On-Net vs Off-Net Decision Tree



Gateway Types and Organization

You will need to define your own gateways based on your network's interconnection requirements. Gateways are typically organized by traffic type, destination, or provider to enable flexible routing policies and cost optimization.

Common gateway usage patterns:

Gateway Type	Usage	Examples
PSTN/Interconnect	Off-net call termination	• PSTN termination • International carriers • Other domestic providers
CS Network	Circuit-switched network	• 2G/3G roaming (MSRN) • CS network integration
Emergency Services	Emergency call routing	• 911/112/000 calls • PSAP routing
Voicemail Platform	Voicemail services	• Voicemail deposit • Message retrieval

Common gateway organization patterns:

1. **By Destination Type:**
   - `sofia/gateway/International_Gateway` - International calls (least-cost routing)
   - `sofia/gateway/National_Gateway` - Domestic/national calls
   - `sofia/gateway/Mobile_Gateway` - Mobile-to-mobile interconnect
   - `sofia/gateway/Emergency_Gateway` - Emergency services (911/112/000)
2. **By Provider:**
   - `sofia/gateway/Provider_A_Primary` - Primary carrier for Provider A traffic
   - `sofia/gateway/Provider_A_Backup` - Backup route for Provider A
   - `sofia/gateway/Provider_B` - Secondary carrier interconnect
   - `sofia/gateway/Transit_Provider` - Transit/hubbing provider
3. **By Geographic Region:**

- `sofia/gateway/APAC_Gateway` - Asia-Pacific region
- `sofia/gateway/EMEA_Gateway` - Europe/Middle East/Africa
- `sofia/gateway/Americas_Gateway` - North/South America

4. **By Function:**

- `sofia/gateway/Voice_Gateway` - Standard voice traffic
- `sofia/gateway/SMS_Gateway` - SMS over SIP (if supported)
- `sofia/gateway/Wholesale_Gateway` - Wholesale/carrier traffic
- `sofia/gateway/CS_Network_Gateway` - Circuit-switched (2G/3G) integration

Example: Multi-gateway configuration

```
<profile name="external">
  <gateways>
    <!-- International traffic -->
    <gateway name="International_Primary">
      <param name="proxy" value="10.1.1.100:5060"/>
      <param name="register" value="false"/>
    </gateway>

    <!-- National/domestic providers -->
    <gateway name="Domestic_Provider_A">
      <param name="proxy" value="10.1.2.100:5060"/>
      <param name="register" value="false"/>
    </gateway>

    <gateway name="Domestic_Provider_B">
      <param name="proxy" value="10.1.3.100:5060"/>
      <param name="register" value="false"/>
    </gateway>

    <!-- Emergency services -->
    <gateway name="Emergency_PSAP">
      <param name="proxy" value="10.1.4.100:5060"/>
      <param name="register" value="false"/>
    </gateway>

    <!-- CS network integration (for MSRN routing) -->
    <gateway name="CS_Network">
      <param name="proxy" value="10.1.5.100:5060"/>
      <param name="register" value="false"/>
    </gateway>
  </gateways>
</profile>
```

Routing logic examples:

You can then route calls to different gateways based on your business logic:

```
<!-- International calls (country codes other than domestic) -->
<extension name="Route-International">
  <condition field="${tas_destination_number}"
expression="^(?!61).*$">
    <action application="bridge"
data="sofia/gateway/International_Primary/${tas_destination_number}"
/>
  </condition>
</extension>

<!-- Route to specific provider based on destination prefix -->
<extension name="Route-Provider-A">
  <condition field="${tas_destination_number}"
expression="^614\d{8}$">
    <action application="bridge"
data="sofia/gateway/Domestic_Provider_A/${tas_destination_number}"
/>
  </condition>
</extension>

<!-- Fallback routing with multiple gateways -->
<extension name="Route-With-Failover">
  <condition field="${tas_destination_number}" expression="^(.*)$">
    <action application="set" data="continue_on_fail=true"/>
    <action application="bridge"
data="sofia/gateway/Primary_Gateway/$1"/>
    <!-- If primary fails, try backup -->
    <action application="bridge"
data="sofia/gateway/Backup_Gateway/$1"/>
  </condition>
</extension>
```

Best practices for gateway organization:

- **Use descriptive names** that reflect the gateway's purpose
- **Plan for redundancy** with primary/backup gateways
- **Organize by cost** to enable least-cost routing policies
- **Separate critical traffic** (emergency calls on dedicated gateway)
- **Document interconnection details** for each gateway (SIP trunk specifications, codec support)

- **Monitor gateway health** and implement failover logic in dialplans

Common Routing Patterns

Pattern: On-net MO call

```
Subscriber A (MO) → TAS → sofia/internal/B@TAS_IP → TAS (MT
processing) → IMS → Subscriber B
```

Pattern: Off-net MO call

```
Subscriber A (MO) → TAS →
sofia/gateway/ExternalSIPGateway/+123456789 → PSTN
```

Pattern: MT call to IMS subscriber

```
External → TAS (MT) → sofia/internal/msisdn@ims.domain → I-CSCF →
S-CSCF → Subscriber
```

Pattern: MT call with CFU to off-net

```
External → TAS (MT) → CFU detected →
sofia/gateway/ExternalSIPGateway/+forwarding_number → PSTN
```

Pattern: MT call to CS roaming subscriber

```
External → TAS (MT) → MSRN retrieved →
sofia/gateway/CS_Gateway/+msrn → CS Network → Subscriber
```

Configuration

See **Configuration Guide** for complete gateway configuration details.

Gateway configuration example:

```
<!-- External PSTN Gateway -->
<gateway name="ExternalSIPGateway">
  <param name="proxy" value="10.1.1.100:5060"/>
  <param name="register" value="false"/>
  <param name="caller-id-in-from" value="true"/>
</gateway>

<!-- CS Network Gateway (for MSRN routing) -->
<gateway name="CS_Gateway">
  <param name="proxy" value="10.1.1.200:5060"/>
  <param name="register" value="false"/>
  <param name="caller-id-in-from" value="true"/>
</gateway>

<!-- Emergency Services Gateway -->
<gateway name="Emergency_GW">
  <param name="proxy" value="10.1.1.250:5060"/>
  <param name="register" value="false"/>
  <param name="caller-id-in-from" value="true"/>
</gateway>
```

Best Practices

1. **Always set codecs** - Use `absolute_codec_string` to ensure proper codec negotiation
2. **Configure timeouts** - Set `progress_timeout` and `bridge_answer_timeout` appropriately
3. **Handle failures** - Use `continue_on_fail=true` with fallback actions
4. **Clean headers** - Remove proprietary headers before external routing
5. **Use early media wisely** - `ring_ready` prevents unexpected announcements
6. **Prevent forking** - Set `Request-Disposition: no-fork` for sequential routing
7. **Log bridge results** - Add logging before/after bridge for troubleshooting
8. **Test retry logic** - Verify `originate_retries` works as expected

Troubleshooting

Enable FreeSWITCH debug logging:

```
<action application="set" data="sip_trace=on"/>
<action application="info" data=""/>  <!-- Dumps all variables to log -->
```

Common issues:

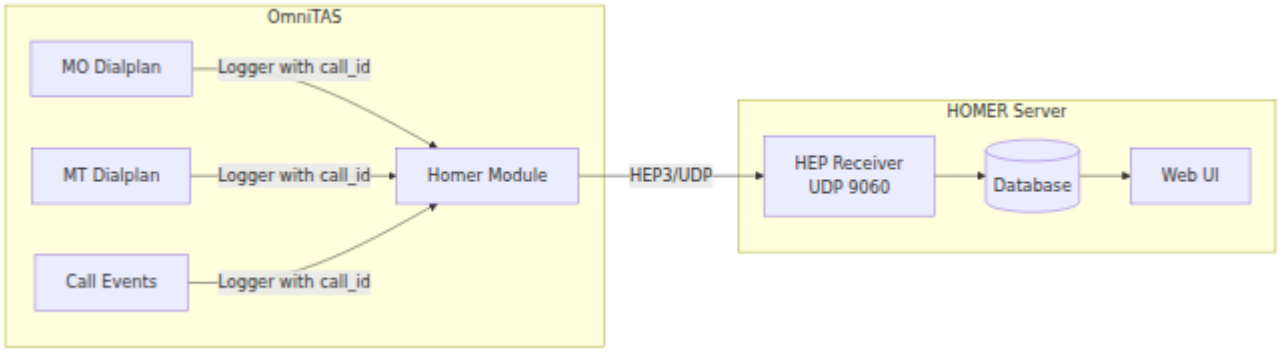
Issue	Cause	Solution
No audio	Codec mismatch	Check `absolute_codec_string`
Call drops immediately	Gateway unreachable	Verify gateway configuration
Timeout too short	`originate_timeout` too low	Increase timeout values
Unwanted announcements	Early media passing through	Use `ignore_early_media=ring_ready`
Wrong caller ID	`sip_cid_type` not set	Set `sip_cid_type=pid`

HOMER Integration

This document covers the optional HOMER integration for correlating application logs with SIP traces.

Overview

OmniTAS can forward application logs to a HOMER server using the HEP3 (Homer Encapsulation Protocol) over UDP. Logs are correlated with SIP traces using the SIP Call-ID, enabling end-to-end visibility of call processing alongside signaling.



How It Works

1. **Log Metadata**: During call processing, OmniTAS sets the SIP Call-ID in Logger metadata
2. **Log Capture**: The Homer module subscribes to log messages via PubSub
3. **Filtering**: Only logs containing `call_id` metadata are forwarded
4. **HEP3 Encoding**: Logs are encoded as HEP3 packets with protocol type 100 (log)
5. **Correlation**: HOMER correlates logs with SIP traces using the Call-ID

Configuration

Add the `homer_server` configuration to your `config/runtime.exs`:

```
config :tas,
  homer_server: %{
    host: "10.179.2.136",
    port: 9060,
    auth_key: nil,
    capture_id: 2001
  }
```

Parameters

Parameter	Type	Required	Default	Description
`host`	String	Yes	-	HOMER server IP address or hostname
`port`	Integer	No	9060	HEP receiver UDP port
`auth_key`	String	No	nil	Optional HEP authentication key. Set to match `authkey` in heplify-server config if enabled
`capture_id`	Integer	No	2001	Capture agent identifier. Appears in HOMER UI to identify the log source

Disabling HOMER Integration

To disable HOMER integration, either:

- Remove the `homer_server` configuration entirely, or
- Comment out the configuration block

When disabled, the Homer module is not started and no logs are forwarded.

HOMER Server Requirements

The HOMER server must be running heplify-server with HEP reception enabled on UDP.

heplify-server Configuration

Verify your heplify-server.toml includes:

```
HEPAddr = "0.0.0.0:9060"
```

Supported Protocol Types

OmniTAS sends logs with HEP protocol type 100, which heplify-server routes to `logs_capture`. Ensure your HOMER database schema supports log storage.

B2BUA Call Leg Correlation (X-CID)

OmniTAS acts as a B2BUA, creating new SIP dialogs (with new Call-IDs) when bridging calls. HOMER needs the `X-CID` header to link all legs into a single call flow.

How It Works

1. The TAS sets `original_call_id` as a channel variable during MO processing, containing the original A-leg SIP Call-ID from the UE
2. The MO dialplan bridge includes `sip_h_X-CID=${original_call_id}` as an inline variable, injecting the header on the B-leg INVITE
3. The MT dialplan reads `${sip_h_X-CID}` from the incoming INVITE and passes it through to its outbound bridge
4. Every B-leg INVITE carries `X-CID` pointing back to the original UE Call-ID

```
UE INVITE (Call-ID: abc@ue)
  -> S-CSCF -> TAS (MO processing)
    -> B-leg INVITE (Call-ID: fs-123, X-CID: abc@ue)  <- on-net
bridge
      -> TAS (MT processing)
        -> B-leg INVITE (Call-ID: fs-456, X-CID: abc@ue)  <- MT
bridge
          -> S-CSCF -> callee
```

All three Call-IDs (`abc@ue`, `fs-123`, `fs-456`) appear as one call in HOMER.

Dialplan Configuration

X-CID must be set as an **inline bridge variable**, not via `set`, because `sip_copy_custom_headers=false` strips `sip_h_` variables set with `set`.

MO dialplan (on-net bridge):

```
<action application="bridge" data="{sip_h_X-
CID=${original_call_id},...}sofia/internal/${tas_destination_number}@
/>
```

MO dialplan (off-net bridge):

```
<action application="bridge" data="{sip_h_X-
CID=${original_call_id},...}sofia/gateway/trunk/${tas_destination_num
/>
```

MT dialplan (all bridges) -- passes through the X-CID received from the MO bridge:

```
<action application="bridge" data="{sip_h_X-CID=${sip_h_X-
CID},...}sofia/internal/${tas_destination_number}@${domain}" />
```

heplify-server Configuration

The HOMER heplify-server must be configured to extract and correlate on X-CID:

```
AlegIDs                 = ["X-CID"]
CustomHeader            = ["X-CID"]
SIPHeader               =
["callid","callid_aleg","method","ruri_user","from_user","to_user","v
```

- `AlegIDs` tells heplify-server to extract the A-leg Call-ID from the X-CID header and store it in `callid_aleg`
- `CustomHeader` stores the raw X-CID value in the data_header JSON for searching
- `SIPHeader` must include `callid_aleg` for HOMER to use it in the call flow correlation

After changing heplify-server.toml, restart: `systemctl restart heplify-server`

Log Correlation

Automatic Call-ID Injection

OmniTAS automatically injects the Call-ID into Logger metadata during:

- **MO Call Processing**: When processing mobile-originated calls
- **MT Call Processing**: When processing mobile-terminated calls
- **Call Events**: When handling answer and hangup events

Log Format

Logs sent to HOMER include:

Field	Description
Correlation ID	SIP Call-ID for trace correlation
Timestamp	Microsecond-precision timestamp
Level	Log level (debug, info, warning, error)
Message	Log message content
Source IP	OmniTAS server IP address
Capture ID	Configured capture agent identifier

Example Log in HOMER

After a call is processed, HOMER displays correlated data:

```
[info] Processing MO call from: 61400123456 to: 61400654321
[debug] Sh lookup for caller: 61400123456
[info] OCS authorization: GRANTED (120 seconds)
[debug] Setting variable: hangup_case = none
```

These logs appear alongside the SIP INVITE, 200 OK, and BYE messages in the HOMER call detail view.

HEP3 Protocol Details

OmniTAS implements HEP3 as defined by the HOMER project, compatible with Kamailio's siptrace module.

Chunk Types Used

Chunk ID	Name	Description
0x0001	IP Family	Always 2 (IPv4)
0x0002	IP Protocol	Always 17 (UDP)
0x0003	Source IPv4	OmniTAS server address
0x0004	Dest IPv4	HOMER server address
0x0007	Source Port	0 (not applicable for logs)
0x0008	Dest Port	Configured HEP port
0x0009	Timestamp Sec	Unix timestamp seconds
0x000A	Timestamp USec	Microseconds component
0x000B	Protocol Type	100 (log)
0x000C	Capture ID	Configured capture_id
0x0011	Correlation ID	SIP Call-ID
0x000F	Payload	Log message
0x000E	Auth Key	Optional authentication

Troubleshooting

Logs Not Appearing in HOMER

Symptoms: Calls complete successfully but no logs appear in HOMER

Possible causes:

- HOMER server unreachable on configured port
- Firewall blocking UDP traffic to port 9060
- heplify-server not configured to receive HEP
- Logs missing call_id metadata

Resolution:

1. Verify network connectivity: `nc -zvu <homer_host> 9060`
2. Check heplify-server is listening: `ss -ulnp | grep 9060`
3. Review heplify-server logs for incoming packets
4. Verify call_id appears in OmniTAS logs (check for `call_id=` in log output)

Authentication Failures

Symptoms: heplify-server logs show authentication errors

Possible causes:

- Mismatched auth_key between OmniTAS and heplify-server
- Auth key configured on one side but not the other

Resolution:

1. If heplify-server has no auth key configured, set `auth_key: nil` in OmniTAS
2. If heplify-server requires auth, configure matching key in both systems

Missing Correlation

Symptoms: Logs appear in HOMER but are not correlated with SIP traces

Possible causes:

- Call-ID format mismatch between SIP and logs
- HOMER correlation configuration

Resolution:

1. Verify the Call-ID in logs matches the SIP Call-ID header exactly
2. Check HOMER's correlation settings for the logs_capture table

Metrics

When HOMER integration is enabled, monitor these indicators:

OmniTAS Logs

Watch for warnings indicating HEP transmission failures:

```
[warning] Failed to send HEP packet to Homer: <reason>
```

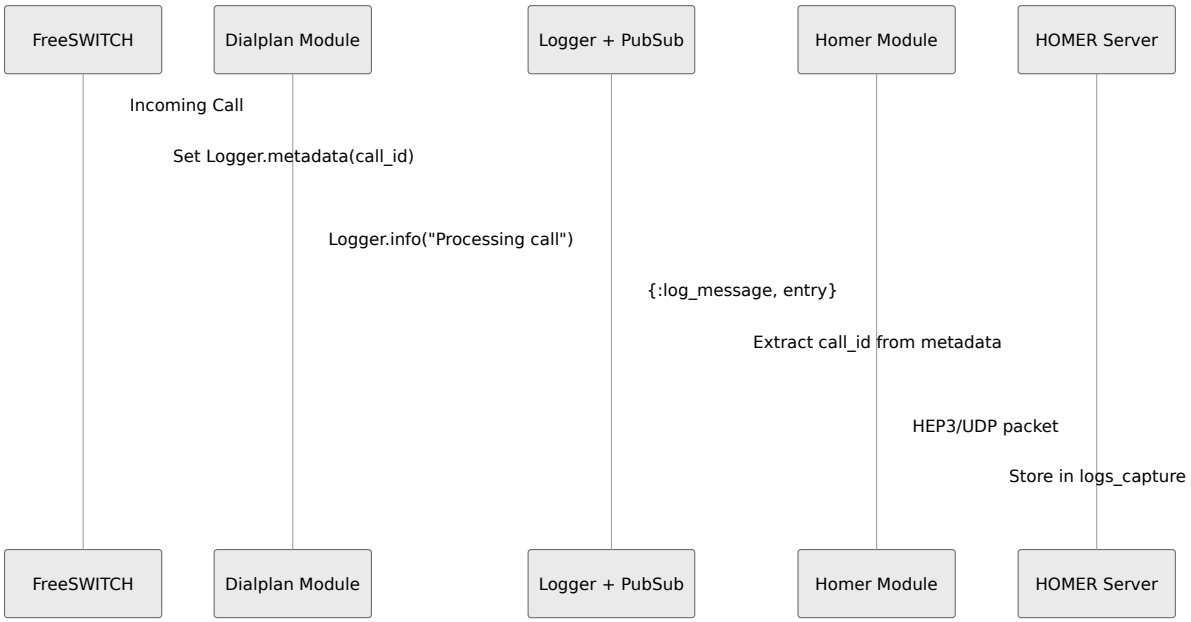
heplify-server Metrics

If Prometheus metrics are enabled on heplify-server, monitor:

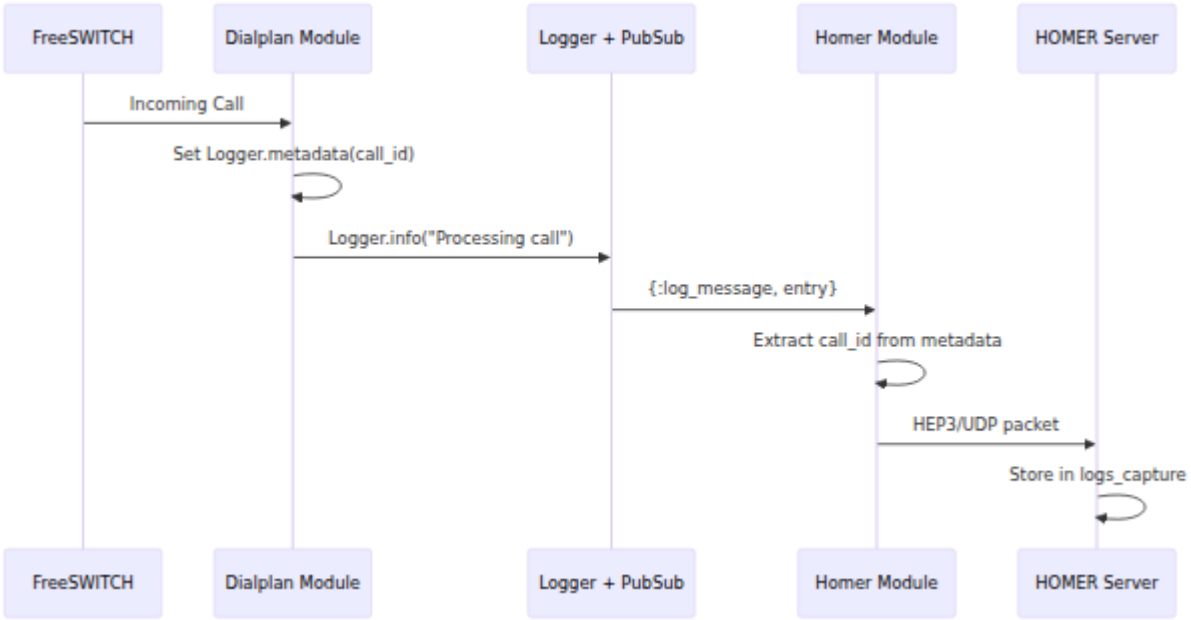
- `heplify_packets_total{type="log"}` - Total log packets received
- `heplify_packets_dropped_total` - Dropped packets (buffer/parse errors)

Architecture Reference

Component Interaction



Data Flow



See Also

- HOMER Project - Open source VoIP/RTC capture and monitoring
- HEP Protocol Specification - HEP/EEP protocol documentation
- heplify-server - HEP capture server

Metrics Documentation

This document describes the Prometheus metrics exposed by the IMS Application Server components.

Table of Contents

- Metrics Endpoints
- Port 9090 - System Metrics
  - Call and Session Metrics
  - System Resource Metrics
  - Memory Metrics
  - Codec Status Metrics
  - Endpoint Status Metrics
  - Module Status Metrics
  - Registration Metrics
  - Sofia Gateway Metrics
  - Exporter Health Metrics
- Port 8080 - TAS Engine Metrics
  - Application Call Metrics
  - Diameter Protocol Metrics
  - Telephony Operations Metrics
  - Online Charging System (OCS) Metrics
  - Dialplan & Processing Metrics
  - Event Socket Metrics
  - Feature Usage Metrics
  - SMS Trigger Metrics
  - Erlang Mnesia Database Metrics
  - Erlang VM Memory Metrics
  - Erlang VM Statistics
  - Erlang VM System Information
  - Erlang VM Microstate Accounting (MSACC)

    - Erlang VM Allocators
- Port 9093 - Media & Call Quality Metrics
    - RTP Audio - Byte Counters
    - RTP Audio - Packet Counters
    - RTP Audio - Special Packet Types
    - RTP Audio - Jitter & Quality Metrics
    - RTCP Metrics
- Go Runtime Metrics
- Process Metrics
- Prometheus HTTP Metrics
- Metric Types
- Usage
- Example Queries
- Metric Types
- Grafana Dashboard Integration
- Alerting Examples
- Troubleshooting with Metrics
- Performance Baselines
- Best Practices

Metrics Endpoints

Port	Endpoint	Purpose	Jump to Section
9090	`/metrics`	System, gateway, and core telephony metrics	Port 9090 →
8080	`/metrics`	TAS engine, Diameter, HLR, OCS, and Erlang VM metrics	Port 8080 →
9093	`/esl?module=default`	RTP/RTCP media quality and call statistics	Port 9093 →

Port 9090 - System Metrics

Call and Session Metrics

Metric Name	Port	Description
`freeswitch_bridged_calls`	9090	Number of bridged calls currently active
`freeswitch_detailed_bridged_calls`	9090	Number of detailed bridged calls active
`freeswitch_current_calls`	9090	Number of calls currently active
`freeswitch_detailed_calls`	9090	Number of detailed calls active
`freeswitch_current_channels`	9090	Number of channels currently active
`freeswitch_current_sessions`	9090	Number of sessions currently active
`freeswitch_current_sessions_peak`	9090	Peak number of sessions since startup
`freeswitch_current_sessions_peak_last_5min`	9090	Peak number of sessions in the last 5 minutes

Metric Name	Port	Description
`freeswitch_sessions_total`	9090	Total number of sessions since startup (counter)
`freeswitch_current_sps`	9090	Current sessions per second
`freeswitch_current_sps_peak`	9090	Peak sessions per second since startup
`freeswitch_current_sps_peak_last_5min`	9090	Peak sessions per second in the last 5 minutes
`freeswitch_max_sessions`	9090	Maximum number of sessions allowed
`freeswitch_max_sps`	9090	Maximum sessions per second allowed

System Resource Metrics

Metric Name	Port	Description
`freeswitch_current_idle_cpu`	9090	Current CPU idle percentage
`freeswitch_min_idle_cpu`	9090	Minimum CPU idle percentage recorded
`freeswitch_uptime_seconds`	9090	Uptime in seconds
`freeswitch_time_synced`	9090	Whether system time is in sync with exporter host time (1=synced, 0=not synced)

Memory Metrics

Metric Name	Port	Description
`freeswitch_memory_arena`	9090	Total non-mmapped bytes (malloc arena)
`freeswitch_memory_ordblks`	9090	Number of free chunks
`freeswitch_memory_smblks`	9090	Number of free fastbin blocks
`freeswitch_memory_hblks`	9090	Number of mapped regions
`freeswitch_memory_hblkhd`	9090	Bytes in mapped regions
`freeswitch_memory_usmblks`	9090	Maximum total allocated space
`freeswitch_memory_fsmblks`	9090	Free bytes held in fastbins
`freeswitch_memory_uordblks`	9090	Total allocated space
`freeswitch_memory_fordblks`	9090	Total free space
`freeswitch_memory_keepcost`	9090	Topmost releasable block

Codec Status Metrics

Metric Name	Port	Description
`freeswitch_codec_status`	9090	Codec status with labels: ikey (module), name (codec name), type (codec). Value=1 indicates codec is available

Available Codecs Include:

- G.711 alaw/ulaw
- PROXY PASS-THROUGH
- PROXY VIDEO PASS-THROUGH
- RAW Signed Linear (16 bit)
- Speex
- VP8/VP9 Video
- AMR variants
- B64
- G.723.1, G.729, G.722, G.726 variants
- OPUS
- MP3
- ADPCM, GSM, LPC-10

Endpoint Status Metrics

Metric Name	Port	Description
`freeswitch_endpoint_status`	9090	Endpoint status with labels: ikey (module), name (endpoint name), type (endpoint). Value=1 indicates endpoint is available

Available Endpoints Include:

- error, group, pickup, user (mod_dptools)
- loopback, null (mod_loopback)
- rtc (mod_rtc)
- rtp, sofia (mod_sofia)
- modem (mod_spandsp)

Module Status Metrics

Metric Name	Port	Description
`freeswitch_load_module`	9090	Module load status (1=loaded, 0=not loaded) with label: module

Key Modules Monitored:

- mod_sofia (SIP)
- mod_conference, mod_conference_ims
- mod_opus, mod_g729, mod_amr, etc.
- mod_event_socket
- mod_dptools
- mod_python3
- mod_rtc
- And many more...

Registration Metrics

Metric Name	Port	Description
`freeswitch_registrations`	9090	Total number of active registrations
`freeswitch_registration_defails`	9090	Detailed registration information with labels: expires, hostname, network_ip, network_port, network_proto, realm, reg_user, token, url

Sofia Gateway Metrics

Metric Name	Port	Description
`freeswitch_sofia_gateway_status`	9090	Gateway status with labels: context, name, profile, proxy, scheme, status (UP/DOWN)
`freeswitch_sofia_gateway_call_in`	9090	Number of inbound calls through gateway
`freeswitch_sofia_gateway_call_out`	9090	Number of outbound calls through gateway
`freeswitch_sofia_gateway_failed_call_in`	9090	Number of failed inbound calls
`freeswitch_sofia_gateway_failed_call_out`	9090	Number of failed outbound calls
`freeswitch_sofia_gateway_ping`	9090	Last ping timestamp (Unix epoch)
`freeswitch_sofia_gateway_pingtime`	9090	Last ping time in milliseconds
`freeswitch_sofia_gateway_pingfreq`	9090	Ping frequency in seconds
`freeswitch_sofia_gateway_pingcount`	9090	Number of pings sent

Metric Name	Port	Description
`freeswitch_sofia_gateway_pingmin`	9090	Minimum ping time recorded
`freeswitch_sofia_gateway_pingmax`	9090	Maximum ping time recorded

Exporter Health Metrics

Metric Name	Port	Description
`freeswitch_up`	9090	Whether the last scrape was successful (1=success, 0=failure)
`freeswitch_exporter_total_scrapes`	9090	Total number of scrapes performed (counter)
`freeswitch_exporter_failed_scrapes`	9090	Total number of failed scrapes (counter)

↑ Back to top

Port 8080 - TAS Engine Metrics

These metrics are exposed by the Telephony Application Server engine and provide insight into call processing, database operations, and Erlang VM performance.

Application Call Metrics

Metric Name	Port	Description
`call_simulations_total`	8080	Total number of call simulations (counter)
`call_attempts_total`	8080	Total number of call attempts (counter)
`call_rejections_total`	8080	Total number of call rejections by reason (counter)
`call_param_errors_total`	8080	Total number of call parameter parsing errors (counter)
`active_calls`	8080	Number of currently active calls with labels: call_type (mo/mt/emergency)
`tracked_call_sessions`	8080	Number of currently tracked call sessions in ETS

Diameter Protocol Metrics

Metric Name	Port	Description
`diameter_peer_state`	8080	State of Diameter peers (1=up, 0=down) with labels: peer_host, peer_realm, application. Polled every 10s for every configured Ro and Sh peer.
`diameter_requests_total`	8080	Total number of Diameter requests (counter) with labels: application, command
`diameter_responses_total`	8080	Total number of Diameter responses (counter) with labels: application, command, result_code
`diameter_response_duration_milliseconds`	8080	Duration of Diameter requests in milliseconds (histogram) with labels: application, command, result

`application` / `command` label values:

application	command	Interface
`ro`	`ccr_i`	Credit-Control-Request, INITIAL (call setup)
`ro`	`ccr_u`	Credit-Control-Request, UPDATE (interim re-authorisation)
`ro`	`ccr_t`	Credit-Control-Request, TERMINATION (call teardown)
`sh`	`udr`	User-Data-Request

> **Note:** Ro Credit-Control metrics are split by request type (`ccr_i`/`ccr_u`/`ccr_t`) so spikes in, for example, failed interim updates are visible independently of setup/teardown. The Sh `udr` path records request count and latency (`diameter_requests_total` / `diameter_response_duration_milliseconds`) in addition to response codes, matching the Ro path. `result_code` carries the Diameter Result-Code (e.g. `2001`); `0` denotes a timeout or unparseable reply. The `result` label on the duration histogram is one of `success`, `nocredit`, `error`, plus `unknown_user` / `no_reply` for Sh.

Telephony Operations Metrics

Metric Name	Port	Description
`hlr_lookups_total`	8080	Total number of HLR lookups (counter)
`hlr_data_duration_milliseconds`	8080	Duration of HLR data retrieval in milliseconds (histogram)
`subscriber_data_lookups_total`	8080	Total number of subscriber data lookups (counter)
`subscriber_data_duration_milliseconds`	8080	Duration of Sh subscriber data retrieval in milliseconds (histogram)
`ss7_map_operations_total`	8080	Total number of SS7 MAP operations (counter)
`ss7_map_http_duration_milliseconds`	8080	Duration of SS7 MAP HTTP requests in milliseconds (histogram)
`tracked_registrations`	8080	Number of currently tracked SIP registrations

Online Charging System (OCS) Metrics

Metric Name	Port	Description
`ocs_authorization_attempts_total`	8080	Total number of OCS authorization attempts (counter)
`ocs_authorization_duration_milliseconds`	8080	Duration of OCS authorization in milliseconds (histogra
`online_charging_events_total`	8080	Total number of online charging events (coun with labels: event_typ result
`authorization_decisions_total`	8080	Total number of authorization decision (counter)
`ro_charging_quota_seconds`	8080	Ro Credit-Control quot seconds (histogram) w labels: request_type (`ccr_i`/`ccr_u`/`ccr_t`) (`requested`/`granted`/

`online_charging_events_total` labels:

- **event_type:** `authorize`, `answer`, `reauth`, `hangup`, `credit_exhaustion_hangup`, `hangup_rescheduled`
- **result:** `success`, `nocredit`, `timeout`, `error`, `triggered`

`ro_charging_quota_seconds` — quota observed on each CCR:

- `kind="requested"` — reservation requested on CCR-I/CCR-U (`Requested-Service-Unit` CC-Time)

- `kind="granted"` — quota the OCS approved in the CCA (`Granted-Service-Unit` CC-Time; `0` = no credit)
- `kind="used"` — units reported as consumed on CCR-T (`Used-Service-Unit` CC-Time)

Comparing `granted` against `used` exposes over- or under-reservation and is the primary charging-correctness signal for the Ro interface.

Dialplan & Processing Metrics

Metric Name	Port	Description
`http_requests_total`	8080	Total number of HTTP requests with labels: endpoint, status_code (counter)
`http_dialplan_request_duration_milliseconds`	8080	Duration of HTTP dialplan requests in milliseconds (histogram)
`dialplan_module_duration_milliseconds`	8080	Duration of individual dialplan module processing (histogram)
`freeswitch_variable_set_duration_milliseconds`	8080	Duration of variable setting operations (histogram)

Event Socket Metrics

Metric Name	Port	Description
`event_socket_connected`	8080	Event Socket connection state (1=connected, 0=disconnected) with label: connection_type
`event_socket_reconnections_total`	8080	Total number of Event Socket reconnection attempts (counter) with labels: connection_type, result
`event_socket_commands_total`	8080	Total number of Event Socket commands executed (counter) with labels: command_type, result
`event_socket_command_timeouts_total`	8080	Total number of Event Socket command timeouts (counter) with label: command_type

Command Types Tracked:

- uuid_setvar, uuid_dump, uuid_kill, uuid_transfer
- uuid_set_media_stats
- sched_hangup, sched_transfer
- vm_boxcount
- status, echo, show, sofia

Result Values:

- success: Command completed successfully

- timeout: Command exceeded timeout threshold
- error: Command returned unexpected response

Feature Usage Metrics

Metric Name	Port	Description
`feature_invocations_total`	8080	Total number of TAS feature invocations (counter) with labels: feature, call_type, result
`feature_data_source_total`	8080	Total number of feature data source usages (counter) with labels: feature, source

Features:

- `call_forward_all` - Unconditional call forwarding
- `call_forward_not_reachable` - Call forwarding when subscriber not reachable
- `call_forward_no_reply` - Call forwarding on no reply
- `call_barring` - OCS-based call barring (insufficient credit)
- `cli_withheld` - CLI privacy/screening

Call Types: `mo`, `mt`

Data Sources: `sh_interface`, `hlr`, `config_fallback`

Result Values: `success`, `error`, `skipped`

SMS Trigger Metrics

Metric Name	Port	Description
`sms_trigger_attempts_total`	8080	Total number of SMS trigger attempts (counter) with labels: trigger_type, result
`sms_trigger_errors_total`	8080	Total number of SMS trigger errors (counter) with labels: trigger_type, error_stage
`smsc_requests_total`	8080	Total number of SMSC HTTP requests (counter) with labels: message_type, result

Trigger Types: `voicemail_deposit`, `voicemail_clear`

Error Stages: `vm_boxcount`, `template_render`, `smsc_request`

Message Types: `notification`, `mwi`

Result Values: `success`, `error`

Erlang Mnesia Database Metrics

Metric Name	Port	Description
`erlang_mnesia_held_locks`	8080	Number of held locks
`erlang_mnesia_lock_queue`	8080	Number of transactions waiting for a lock
`erlang_mnesia_transaction_participants`	8080	Number of participant transactions
`erlang_mnesia_transaction_coordinators`	8080	Number of coordinator transactions
`erlang_mnesia_failed_transactions`	8080	Number of failed (aborted) transactions (counter)
`erlang_mnesia_committed_transactions`	8080	Number of committed transactions (counter)
`erlang_mnesia_logged_transactions`	8080	Number of transactions logged (counter)
`erlang_mnesia_restarted_transactions`	8080	Total number of transaction

Metric Name	Port	Description
		restarts (counter)
`erlang_mnesia_memory_usage_bytes`	8080	Total bytes allocated by all mnesia tables
`erlang_mnesia_tablewise_memory_usage_bytes`	8080	Bytes allocated per mnesia table with label: table
`erlang_mnesia_tablewise_size`	8080	Number of rows per table with label: table

Erlang VM Memory Metrics

Metric Name	Port	Description
`erlang_vm_memory_atom_bytes_total`	8080	Memory allocated for a with label: usage (used,
`erlang_vm_memory_bytes_total`	8080	Total memory allocated label: kind (system/processes)
`erlang_vm_memory_dets_tables`	8080	DETS tables count
`erlang_vm_memory_ets_tables`	8080	ETS tables count
`erlang_vm_memory_processes_bytes_total`	8080	Memory allocated for processes with label: us (used/free)
`erlang_vm_memory_system_bytes_total`	8080	Memory for emulator (r process-related) with la usage (atom/binary/code/ets/c

Erlang VM Statistics

Metric Name	Port	Des
`erlang_vm_statistics_bytes_output_total`	8080	Tota outp port (cou
`erlang_vm_statistics_bytes_received_total`	8080	Tota rece thro port (cou
`erlang_vm_statistics_context_switches`	8080	Tota swit sinc star (cou
`erlang_vm_statistics_dirty_cpu_run_queue_length`	8080	Leng dirty run-
`erlang_vm_statistics_dirty_io_run_queue_length`	8080	Leng dirty que
`erlang_vm_statistics_garbage_collection_number_of_gcs`	8080	Num garb colle (cou
`erlang_vm_statistics_garbage_collection_bytes_reclaimed`	8080	Byte recl

Metric Name	Port	Des
		by C (cou
`erlang_vm_statistics_garbage_collection_words_reclaimed`	8080	Wor recl by C (cou
`erlang_vm_statistics_reductions_total`	8080	Tota redu (cou
`erlang_vm_statistics_run_queues_length`	8080	Leng norm que
`erlang_vm_statistics_runtime_milliseconds`	8080	Sum runt all t (cou
`erlang_vm_statistics_wallclock_time_milliseconds`	8080	Rea mea (cou

Erlang VM System Information

Metric Name	Port	Description
`erlang_vm_dirty_cpu_schedulers`	8080	Number of dirty CPU scheduler threads
`erlang_vm_dirty_cpu_schedulers_online`	8080	Number of dirty CPU schedulers online
`erlang_vm_dirty_io_schedulers`	8080	Number of dirty I/O scheduler threads
`erlang_vm_ets_limit`	8080	Maximum number of ETS tables allowed
`erlang_vm_logical_processors`	8080	Number of logical processors configured
`erlang_vm_logical_processors_available`	8080	Number of logical processors available
`erlang_vm_logical_processors_online`	8080	Number of logical processors online
`erlang_vm_port_count`	8080	Number of ports currently existing
`erlang_vm_port_limit`	8080	Maximum number of ports allowed
`erlang_vm_process_count`	8080	Number of processes currently existing
`erlang_vm_process_limit`	8080	Maximum number of processes allowed

Metric Name	Port	Description
`erlang_vm_schedulers`	8080	Number of scheduler threads
`erlang_vm_schedulers_online`	8080	Number of schedulers online
`erlang_vm_smp_support`	8080	1 if compiled with SMP support, 0 otherwise
`erlang_vm_threads`	8080	1 if compiled with thread support, 0 otherwise
`erlang_vm_thread_pool_size`	8080	Number of async threads in pool
`erlang_vm_time_correction`	8080	1 if time correction enabled, 0 otherwise
`erlang_vm_wordsize_bytes`	8080	Size of Erlang term words in bytes
`erlang_vm_atom_count`	8080	Number of atoms currently existing
`erlang_vm_atom_limit`	8080	Maximum number of atoms allowed

Erlang VM Microstate Accounting (MSACC)

Detailed time tracking for scheduler activities with labels: type, id

Metric Name	Port	Description
`erlang_vm_msacc_aux_seconds_total`	8080	Time spent handling auxiliary jobs (counter)
`erlang_vm_msacc_check_io_seconds_total`	8080	Time spent checking for new I/O events (counter)
`erlang_vm_msacc_emulator_seconds_total`	8080	Time spent executing Erlang processes (counter)
`erlang_vm_msacc_gc_seconds_total`	8080	Time spent in garbage collection (counter)
`erlang_vm_msacc_other_seconds_total`	8080	Time spent on unaccounted activities (counter)
`erlang_vm_msacc_port_seconds_total`	8080	Time spent executing ports (counter)
`erlang_vm_msacc_sleep_seconds_total`	8080	Time spent sleeping (counter)
`erlang_vm_msacc_alloc_seconds_total`	8080	Time spent managing memory (counter)
`erlang_vm_msacc_bif_seconds_total`	8080	Time spent in BIFs (counter)

Metric Name	Port	Description
`erlang_vm_msacc_busy_wait_seconds_total`	8080	Time spent busy waiting (counter)
`erlang_vm_msacc_ets_seconds_total`	8080	Time spent in ETS BIFs (counter)
`erlang_vm_msacc_gc_full_seconds_total`	8080	Time spent in fullsweep GC (counter)
`erlang_vm_msacc_nif_seconds_total`	8080	Time spent in NIFs (counter)
`erlang_vm_msacc_send_seconds_total`	8080	Time spent sending messages (counter)
`erlang_vm_msacc_timers_seconds_total`	8080	Time spent managing timers (counter)

Erlang VM Allocators

Detailed memory allocator metrics with labels: alloc, instance_no, kind, usage

Metric Name	Port	Description
`erlang_vm_allocators`	8080	Allocated (carriers_size) and used (blocks_size) memory for different allocators. See erts_alloc(3).

Allocator types include: temp_alloc, sl_alloc, std_alloc, ll_alloc, eheap_alloc, ets_alloc, fix_alloc, literal_alloc, binary_alloc, driver_alloc

---

↑ Back to top

Port 9093 - Media & Call Quality Metrics

These metrics provide real-time RTP/RTCP statistics and call quality information per channel.

Metric Name	Port	Description
`freeswitch_info`	9093	System info with label: version
`freeswitch_up`	9093	Ready status (1=ready, 0=not ready)
`freeswitch_stack_bytes`	9093	Stack size in bytes
`freeswitch_session_total`	9093	Total number of sessions
`freeswitch_session_active`	9093	Active number of sessions
`freeswitch_session_limit`	9093	Session limit
`rtp_channel_info`	9093	RTP channel info with labels for channel details

RTP Audio - Byte Counters

Metric Name	Port	Description
`rtp_audio_in_raw_bytes_total`	9093	Total bytes received (including headers)
`rtp_audio_out_raw_bytes_total`	9093	Total bytes sent (including headers)
`rtp_audio_in_media_bytes_total`	9093	Total media bytes received (payload only)
`rtp_audio_out_media_bytes_total`	9093	Total media bytes sent (payload only)

RTP Audio - Packet Counters

Metric Name	Port	Description
`rtp_audio_in_packets_total`	9093	Total packets received
`rtp_audio_out_packets_total`	9093	Total packets sent
`rtp_audio_in_media_packets_total`	9093	Total media packets received
`rtp_audio_out_media_packets_total`	9093	Total media packets sent
`rtp_audio_in_skip_packets_total`	9093	Inbound packets discarded
`rtp_audio_out_skip_packets_total`	9093	Outbound packets discarded

RTP Audio - Special Packet Types

Metric Name	Port	Description
`rtp_audio_in_jitter_packets_total`	9093	Jitter buffer packets received
`rtp_audio_in_dtmf_packets_total`	9093	DTMF packets received
`rtp_audio_out_dtmf_packets_total`	9093	DTMF packets sent
`rtp_audio_in_cng_packets_total`	9093	Comfort Noise Generation packets received
`rtp_audio_out_cng_packets_total`	9093	Comfort Noise Generation packets sent
`rtp_audio_in_flush_packets_total`	9093	Flushed packets (buffer resets)

RTP Audio - Jitter & Quality Metrics

Metric Name	Port	Description
`rtp_audio_in_jitter_buffer_bytes_max`	9093	Largest jitter buffer size in bytes
`rtp_audio_in_jitter_seconds_min`	9093	Minimum jitter in seconds
`rtp_audio_in_jitter_seconds_max`	9093	Maximum jitter in seconds
`rtp_audio_in_jitter_loss_rate`	9093	Packet loss rate due to jitter (ratio)
`rtp_audio_in_jitter_burst_rate`	9093	Packet burst rate due to jitter (ratio)
`rtp_audio_in_mean_interval_seconds`	9093	Mean interval between inbound packets
`rtp_audio_in_flaw_total`	9093	Total audio flaws detected (glitches, artifacts)
`rtp_audio_in_quality_percent`	9093	Audio quality as percentage (0-100)
`rtp_audio_in_quality_mos`	9093	Mean Opinion Score (1-5, where 5 is best)

RTCP Metrics

Metric Name	Port	Description
`rtcp_audio_bytes_total`	9093	Total RTCP bytes
`rtcp_audio_packets_total`	9093	Total RTCP packets

Go Runtime Metrics

Metric Name	Port	Description
`go_goroutines`	9090	Number of goroutines currently running
`go_threads`	9090	Number of OS threads created
`go_info`	9090	Information about the Go environment (with version label)
`go_gc_duration_seconds`	9090	Pause duration of garbage collection cycles (summary)
`go_memstats_alloc_bytes`	9090	Number of bytes allocated and still in use
`go_memstats_alloc_bytes_total`	9090	Total number of bytes allocated (counter)
`go_memstats_heap_alloc_bytes`	9090	Heap bytes allocated and still in use
`go_memstats_heap_idle_bytes`	9090	Heap bytes waiting to be used
`go_memstats_heap_inuse_bytes`	9090	Heap bytes currently in use
`go_memstats_heap_objects`	9090	Number of allocated heap objects
`go_memstats_heap_released_bytes`	9090	Heap bytes released to OS

Metric Name	Port	Description
`go_memstats_heap_sys_bytes`	9090	Heap bytes obtained from system
`go_memstats_sys_bytes`	9090	Total bytes obtained from system

Process Metrics

Metric Name	Port	Description
`process_cpu_seconds_total`	9090	Total user and system CPU time spent (counter)
`process_max_fds`	9090	Maximum number of open file descriptors
`process_open_fds`	9090	Current number of open file descriptors
`process_resident_memory_bytes`	9090	Resident memory size in bytes
`process_virtual_memory_bytes`	9090	Virtual memory size in bytes
`process_virtual_memory_max_bytes`	9090	Maximum amount of virtual memory available
`process_start_time_seconds`	9090	Process start time since Unix epoch

Prometheus HTTP Metrics

Metric Name	Port	Description
`promhttp_metric_handler_requests_in_flight`	9090	Current number of scrapes being served
`promhttp_metric_handler_requests_total`	9090	Total number of scrapes by HTTP status code (counter)

↑ Back to top

Metric Types

- **gauge**: A metric that can go up or down (e.g., current_calls, cpu_idle)
- **counter**: A metric that only increases (e.g., sessions_total, failed_scrapes)
- **summary**: A metric that tracks quantiles over a sliding time window (e.g., gc_duration_seconds)

↑ Back to top

Usage

To scrape these metrics, configure your Prometheus server to scrape all three endpoints:

```yaml
scrape_configs:
  - job_name: 'ims_as_system'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'ims_as_engine'
    static_configs:
      - targets: ['localhost:8080']
    metrics_path: '/metrics'

  - job_name: 'ims_as_media'
    static_configs:
      - targets: ['localhost:9093']
    metrics_path: '/esl'
    params:
      module: ['default']
```

↑ Back to top

Example Queries

Quick Links:

- General Metrics (Port 9090)
- Media Quality Metrics (Port 9093)
- TAS Engine Metrics (Port 8080)

General Metrics

Current call volume:

```
freeswitch_current_calls
```

Gateway health:

```
freeswitch_sofia_gateway_status{status="UP"}
```

Average ping time to gateways:

```
avg(freeswitch_sofia_gateway_pingtime)
```

Sessions per second rate:

```
freeswitch_current_sps
```

Memory usage:

```
freeswitch_memory_uordblks
```

Media Quality Metrics

Call quality (MOS score):

```
rtp_audio_in_quality_mos
```

Audio quality percentage:

```
rtp_audio_in_quality_percent
```

Jitter rate:

```
rate(rtp_audio_in_jitter_packets_total[5m])
```

Packet loss rate:

```
rtp_audio_in_jitter_loss_rate
```

Average jitter:

```
avg(rtp_audio_in_jitter_seconds_max -
rtp_audio_in_jitter_seconds_min)
```

RTP bandwidth (inbound):

```
rate(rtp_audio_in_media_bytes_total[1m]) * 8
```

Audio flaws detected:

```
increase(rtp_audio_in_flaw_total[5m])
```

TAS Engine Metrics

Active calls by type:

```
active_calls
```

Diameter peer health:

```
diameter_peer_state{application="sh"}
```

Call attempt rate:

```
rate(call_attempts_total[5m])
```

HLR lookup latency (95th percentile):

```
histogram_quantile(0.95, hlr_data_duration_milliseconds)
```

OCS authorization latency:

```
histogram_quantile(0.99, ocs_authorization_duration_milliseconds)
```

Subscriber data lookup rate:

```
rate(subscriber_data_lookups_total[5m])
```

Diameter request success rate:

```
rate(diameter_responses_total[5m]) /
rate(diameter_requests_total[5m])
```

CCR rate by request type (spot spikes in interim updates):

```
sum by (command) (rate(diameter_requests_total{application="ro"}
[5m]))
```

CCR error rate, per request type:

```
sum by (command) (rate(diameter_responses_total{application="ro",
result_code!="2001"}[5m]))
```

Sh (UDR) P95 latency:

```
histogram_quantile(0.95,
rate(diameter_response_duration_milliseconds_bucket{application="sh"}
[5m]))
```

Median granted credit-control quota on call setup:

```
histogram_quantile(0.5,
rate(ro_charging_quota_seconds_bucket{request_type="ccr_i",
kind="granted"}[5m]))
```

No-credit grants (zero-second allocations) per second:

```
rate(ro_charging_quota_seconds_bucket{kind="granted", le="0"}[5m])
```

Event Socket connection status:

```
event_socket_connected
```

Mnesia transaction performance:

```
rate(erlang_mnesia_committed_transactions[5m])
```

Mnesia failed transaction rate:

```
rate(erlang_mnesia_failed_transactions[5m])
```

Erlang VM process count:

```
erlang_vm_process_count
```

Erlang VM memory usage:

```
erlang_vm_memory_bytes_total
```

Garbage collection rate:

```
rate(erlang_vm_statistics_garbage_collection_number_of_gcs[5m])
```

Scheduler run queue length:

```
erlang_vm_statistics_run_queues_length
```

ETS table count:

```
erlang_vm_memory_ets_tables
```

HTTP dialplan request duration (median):

```
histogram_quantile(0.5,
http_dialplan_request_duration_milliseconds)
```

↑ Back to top

Grafana Dashboard Integration

The metrics can be visualized in Grafana using the Prometheus data source.

Recommended Dashboard Layout

Row 1: Call Volume & Health

- Active calls gauge (`active_calls`)
- Call attempts rate by type (`rate(call_attempts_total[5m])`)
- Call rejection rate (`rate(call_rejections_total[5m])`)
- Gateway health (`freeswitch_sofia_gateway_status`)

Row 2: Performance (Latency Percentiles)

- P95 HTTP dialplan request time by call type
- P95 Sh subscriber data lookup time
- P95 HLR lookup time
- P95 OCS authorization time
- P95 Diameter response time by application

Row 3: Success Rates

- Subscriber data lookup success rate
- HLR lookup success rate

- OCS authorization success rate
- Diameter peer state

Row 4: Media Quality

- Call quality MOS score (`rtp_audio_in_quality_mos`)
- Audio quality percentage (`rtp_audio_in_quality_percent`)
- Jitter statistics
- Packet loss rate

Row 5: System Resources

- Erlang VM process count
- Erlang VM memory usage
- ETS table count
- Scheduler run queue length
- Garbage collection rate

Row 6: Error Tracking

- Call parameter errors
- Authorization failures
- Event Socket connection status
- Mnesia transaction failures

Example Panel Queries

Active Calls by Type:

```
sum by (call_type) (active_calls)
```

P95 Dialplan Generation Latency:

```
histogram_quantile(0.95,
  rate(http_dialplan_request_duration_milliseconds_bucket[5m])
)
```

Diameter Success Rate:

```
rate(diameter_responses_total{result="success"}[5m]) /
rate(diameter_requests_total[5m]) * 100
```

Media Quality - Average MOS:

```
avg(rtp_audio_in_quality_mos)
```

↑ Back to top

Alerting Examples

Critical Alerts (Page Immediately)

System Down - No Call Attempts:

```yaml
alert: SystemDown
expr: rate(call_attempts_total[5m]) == 0
for: 2m
labels:
  severity: critical
annotations:
  summary: "TAS system appears down - no call attempts"
  description: "No call attempts detected for 2 minutes"
```

Diameter Peer Down:

```
alert: DiameterPeerDown
expr: diameter_peer_state == 0
for: 1m
labels:
  severity: critical
annotations:
  summary: "Diameter peer {{ $labels.peer_host }} is down"
  description: "Peer for {{ $labels.application }} application is
unavailable"
```

Event Socket Disconnected:

```
alert: EventSocketDisconnected
expr: event_socket_connected == 0
for: 30s
labels:
  severity: critical
annotations:
  summary: "Event Socket {{ $labels.connection_type }}
disconnected"
  description: "Critical communication channel down"
```

High Severity Alerts

High Diameter Latency:

```
alert: HighDiameterLatency
expr: |
  histogram_quantile(0.95,
    rate(diameter_response_duration_milliseconds_bucket[5m])
  ) > 1000
for: 5m
labels:
  severity: high
annotations:
  summary: "High Diameter latency detected"
  description: "P95 latency is {{ $value }}ms"
```

OCS Authorization Failures:

```
alert: OCSAuthFailures
expr: |
  rate(ocs_authorization_attempts_total{result="no_credit"}[5m]) /
  rate(ocs_authorization_attempts_total[5m]) > 0.1
for: 5m
labels:
  severity: high
annotations:
  summary: "High rate of OCS no-credit responses"
  description: "{{ $value | humanizePercentage }} of requests
denied credit"
```

High Call Rejection Rate:

```
alert: HighCallRejectionRate
expr: |
  rate(call_rejections_total[5m]) /
  rate(call_attempts_total[5m]) > 0.05
for: 5m
labels:
  severity: high
annotations:
  summary: "Call rejection rate above 5%"
  description: "{{ $value | humanizePercentage }} of calls
rejected"
```

Poor Media Quality:

```
alert: PoorMediaQuality
expr: avg(rtp_audio_in_quality_mos) < 3.5
for: 3m
labels:
  severity: high
annotations:
  summary: "Poor call quality detected"
  description: "Average MOS score is {{ $value }}"
```

Warning Alerts

High Memory Usage:

```
alert: HighMemoryUsage
expr: |
  erlang_vm_memory_bytes_total{kind="processes"} /
  (erlang_vm_process_limit * 1000000) > 0.8
for: 10m
labels:
  severity: warning
annotations:
  summary: "Erlang VM memory usage high"
  description: "Process memory at {{ $value | humanizePercentage
}}"
```

High Scheduler Run Queue:

```
alert: HighSchedulerRunQueue
expr: erlang_vm_statistics_run_queues_length > 10
for: 5m
labels:
  severity: warning
annotations:
  summary: "High scheduler run queue length"
  description: "Run queue length is {{ $value }}"
```

Mnesia Transaction Failures:

```
alert: MnesiaTransactionFailures
expr: rate(erlang_mnesia_failed_transactions[5m]) > 1
for: 5m
labels:
  severity: warning
annotations:
  summary: "Mnesia transaction failures detected"
  description: "{{ $value }} failures per second"
```

↑ Back to top

Troubleshooting with Metrics

Problem: Calls are slow

Investigation Steps:

1. **Check overall dialplan generation time:**

```
histogram_quantile(0.95,
rate(http_dialplan_request_duration_milliseconds_bucket[5m]))
```

2. **Break down by component:**

```
# Subscriber data lookup
histogram_quantile(0.95,
rate(subscriber_data_duration_milliseconds_bucket[5m]))

# HLR lookup
histogram_quantile(0.95,
rate(hlr_data_duration_milliseconds_bucket[5m]))

# OCS authorization
histogram_quantile(0.95,
rate(ocs_authorization_duration_milliseconds_bucket[5m]))
```

3. **Check module-specific delays:**

```
histogram_quantile(0.95,
  rate(dialplan_module_duration_milliseconds_bucket[5m])
) by (module)
```

Common Causes:

- External system latency (HSS, HLR, OCS)
- Network issues

- Database contention
- High system load

Problem: Calls are failing

Investigation Steps:

1. **Check call rejection reasons:**

```
sum by (reason) (rate(call_rejections_total[5m]))
```

2. **Check authorization decisions:**

```
sum by (decision) (rate(authorization_decisions_total[5m]))
```

3. **Check Diameter peer health:**

```
diameter_peer_state
```

4. **Check Event Socket connection:**

```
event_socket_connected
```

Problem: High load

Investigation Steps:

1. **Check call volume:**

```
rate(call_attempts_total[5m])
active_calls
```

2. **Check Erlang VM resources:**

```
erlang_vm_process_count
erlang_vm_statistics_run_queues_length
erlang_vm_memory_bytes_total
```

3. **Check garbage collection:**

```
rate(erlang_vm_statistics_garbage_collection_number_of_gcs[5m])
```

Problem: Poor Media Quality

Investigation Steps:

1. **Check MOS scores:**

```
rtp_audio_in_quality_mos
rtp_audio_in_quality_percent
```

2. **Check jitter:**

```
rtp_audio_in_jitter_seconds_max
rtp_audio_in_jitter_loss_rate
```

3. **Check packet loss:**

```
rtp_audio_in_skip_packets_total
rtp_audio_in_flaw_total
```

4. **Check bandwidth usage:**

```
rate(rtp_audio_in_media_bytes_total[1m]) * 8
```

↑ Back to top

Performance Baselines

Typical Values (Well-Tuned System)

Latency (P95):

- HTTP dialplan request: 200-500ms
- Subscriber data (Sh) lookup: 50-150ms
- HLR data lookup: 100-300ms
- OCS authorization: 100-250ms
- Diameter requests: 50-200ms
- Dialplan module processing: 10-50ms per module

Success Rates:

- Call completion: >95%
- Subscriber data lookups: >99%
- HLR lookups: >98%
- OCS authorizations: >99% (excluding legitimate no-credit)
- Diameter peer uptime: >99.9%

Media Quality:

- MOS score: >4.0
- Audio quality percentage: >80%
- Jitter: <30ms
- Packet loss rate: <1%

System Resources:

- Erlang process count: <50% of limit
- Erlang memory usage: <70% of available
- Scheduler run queue: <5
- ETS tables: <1000

Capacity Planning

Per-Server Capacity (recommended maximums):

- Concurrent calls: 500-1000 (depends on hardware)
- Calls per second: 20-50 CPS
- Registered subscribers: 10,000-50,000

Scaling Indicators (add capacity when):

- Active calls consistently >70% of capacity
- Erlang process count >70% of limit
- P95 latency degrading
- Scheduler run queues consistently >10

↑ Back to top

Best Practices

Monitoring Strategy

1. **Set up dashboards for different audiences:**
    - Operations dashboard: Call volume, success rates, system health
    - Engineering dashboard: Latency percentiles, error rates, resource usage
    - Executive dashboard: High-level KPIs, uptime, cost metrics
2. **Configure alerts at multiple levels:**
    - Critical: Page on-call (system down, major outage)
    - High: Alert during business hours (degraded performance)
    - Warning: Track in ticket system (potential issues)
3. **Use appropriate time ranges:**

- Real-time monitoring: 5-minute windows
- Troubleshooting: 15-minute to 1-hour windows
- Capacity planning: Daily/weekly aggregates

4. **Focus on user impact:**
   - Prioritize end-to-end latency metrics
   - Track success rates over individual error counters
   - Monitor media quality for user experience

Query Performance

1. **Use recording rules for frequently-used queries:**

```
groups:
  - name: ims_as_aggregations
    interval: 30s
    rules:
      - record: job:call_attempts:rate5m
        expr: rate(call_attempts_total[5m])

      - record: job:dialplan_latency:p95
        expr: histogram_quantile(0.95,
rate(http_dialplan_request_duration_milliseconds_bucket[5m]))
```

2. **Avoid high-cardinality labels** in queries (e.g., don't group by phone number)

3. **Use appropriate rate intervals:**
   - Short-term trends: `[5m]`
   - Medium-term trends: `[1h]`
   - Long-term trends: `[1d]`

Metric Cardinality

Monitor cardinality to prevent Prometheus performance issues:

```
# Check metric cardinality
count by (__name__) ({__name__=~".+"})
```

High-cardinality risks:

- Labels with unique values per call (phone numbers, call IDs)
- Unbounded label values
- Labels with >1000 unique values

Solution:

- Use labels for categories, not unique identifiers
- Aggregate high-cardinality data in external systems
- Use recording rules to pre-aggregate

↑ Back to top

Number Translation

Back to Main Documentation

Number translation converts phone numbers between different formats to ensure consistent E.164 formatting throughout the system.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - Number translation configuration (`number_translate`)
- **Operations Guide** - Number translation testing in Control Panel

Call Processing Flow

- **Dialplan Configuration** - Using translated numbers in dialplan (translation happens first)
- **Sh Interface** - Sh lookup uses translated numbers
- **Online Charging** - OCS receives translated numbers
- **SS7 MAP** - HLR queries use translated numbers

Related Services

- ⚙ **Supplementary Services** - CLI blocking prefix stripping during translation
- **Voicemail** - Voicemail numbers in translation

Monitoring

- **Metrics Reference** - Number translation metrics

Number Translation

Number translation converts phone numbers between different formats (local, national, international) to ensure consistent E.164 formatting throughout the system.

What is Number Translation?

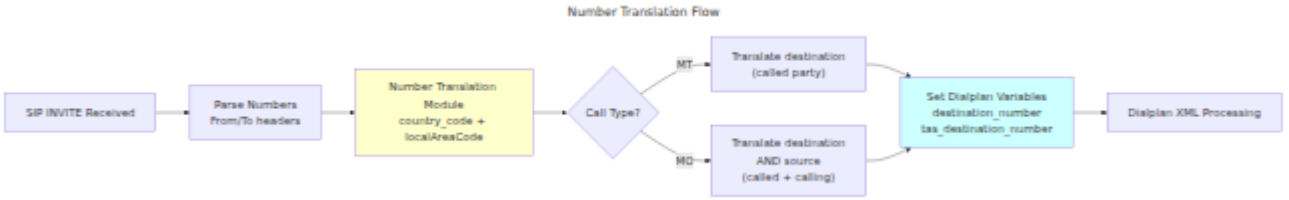
Number translation normalizes phone numbers to E.164 format (international standard) before call processing. This ensures:

- Consistent numbering throughout the system
- Proper routing to on-net and off-net destinations
- Compatibility with international SIP trunks and IMS networks

E.164 Format: `[Country Code][National Number]` (no + prefix, no spaces)

- Example: `61403123456` (Australia mobile)
- Example: `16505551234` (US number)

When Translation Occurs



Translation happens:

- **Before** Sh lookups
- **Before** HLR lookups
- **Before** OCS authorization
- **Before** dialplan XML is generated

For MT Calls: Translate destination number (called party) **For MO Calls:** Translate both source and destination numbers

Configuration

```
config :tas,
  number_translate: %{
    country_code: :PF,        # ISO 3166-1 alpha-2 country code
    localAreaCode: "617"      # Default area code for short
numbers
  }
```

Parameters:

- **`country_code`:** ISO country code as atom (e.g., `:AU`, `:US`, `:PF`)
- **`localAreaCode`:** Area code prepended to short local numbers

Supported Country Codes

The TAS includes translation logic for **248 countries and territories**, covering all ISO 3166-1 alpha-2 codes with telephone service.

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:AC	Ascension Island	247	5	None	00	
:AD	Andorra	376	6-9	None	00	
:AE	United Arab Emirates	971	9	0	00	
:AF	Afghanistan	93	9	0	00	
:AG	Antigua and Barbuda	1	10	None	011	NANP, delegat to US
:AI	Anguilla	1	10	None	011	NANP, delegat to US
:AL	Albania	355	9	0	00	
:AM	Armenia	374	8	0	00	
:AO	Angola	244	9	None	00	
:AR	Argentina	54	10	0	00	
:AS	American Samoa	1	10	None	011	NANP, delegat to US
:AT	Austria	43	10-13	0	00	
:AU	Australia	61	9	0	0011	
:AW	Aruba	297	7	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:AX	Aland Islands	358	9-10	0	00	Delegat to FI
:AZ	Azerbaijan	994	9	0	00	
:BA	Bosnia and Herzegovina	387	8	0	00	
:BB	Barbados	1	10	None	011	NANP, delegat to US
:BD	Bangladesh	880	10	0	00	
:BE	Belgium	32	8-9	0	00	
:BF	Burkina Faso	226	8	None	00	
:BG	Bulgaria	359	9	0	00	
:BH	Bahrain	973	8	None	00	
:BI	Burundi	257	8	None	00	
:BJ	Benin	229	8	None	00	
:BL	Saint Barthelemy	590	9	0	00	
:BM	Bermuda	1	10	None	011	NANP, delegat to US
:BN	Brunei	673	7	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:BO`	Bolivia	`591`	8	None	00	
`:BQ`	Bonaire/Sint Eustatius/Saba	`599`	7	None	00	
`:BR`	Brazil	`55`	10-11	0	00	
`:BS`	Bahamas	`1`	10	None	011	NANP, delegat to US
`:BT`	Bhutan	`975`	8	None	00	
`:BV`	Bouvet Island	`47`	8	None	00	Delegat to NO
`:BW`	Botswana	`267`	8	None	00	
`:BY`	Belarus	`375`	9	8	810	
`:BZ`	Belize	`501`	7	None	00	
`:CA`	Canada	`1`	10	None	011	NANP, delegat to US
`:CC`	Cocos Islands	`61`	9	0	0011	Delegat to AU
`:CD`	DR Congo	`243`	9	0	00	
`:CF`	Central African Republic	`236`	8	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:CG	Republic of Congo	242	9	None	00	
:CH	Switzerland	41	9	0	00	
:CI	Ivory Coast	225	10	None	00	
:CK	Cook Islands	682	5	None	00	
:CL	Chile	56	9	0	00	
:CM	Cameroon	237	9	None	00	
:CN	China	86	11	None	00	
:CO	Colombia	57	10	0	00	
:CR	Costa Rica	506	8	None	00	
:CU	Cuba	53	8	0	119	
:CV	Cape Verde	238	7	None	00	
:CW	Curacao	599	7-8	None	00	
:CX	Christmas Island	61	9	0	0011	Delegat to AU
:CY	Cyprus	357	8	None	00	
:CZ	Czech Republic	420	9	None	00	
:DE	Germany	49	10-12	0	00	
:DJ	Djibouti	253	8	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:DK`	Denmark	`45`	8	None	00	
`:DM`	Dominica	`1`	10	None	011	NANP, delegat to US
`:DO`	Dominican Republic	`1`	10	None	011	NANP, delegat to US
`:DZ`	Algeria	`213`	9	0	00	
`:EC`	Ecuador	`593`	9	0	00	
`:EE`	Estonia	`372`	7-8	None	00	
`:EG`	Egypt	`20`	10	0	00	
`:EH`	Western Sahara	`212`	9	0	00	Delegat to MA
`:ER`	Eritrea	`291`	7	0	00	
`:ES`	Spain	`34`	9	None	00	
`:ET`	Ethiopia	`251`	9	0	00	
`:FI`	Finland	`358`	9-10	0	00	
`:FJ`	Fiji	`679`	7	None	00	
`:FK`	Falkland Islands	`500`	5	None	00	
`:FM`	Micronesia	`691`	7	None	011	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:FO	Faroe Islands	298	6	None	00	
:FR	France	33	9	0	00	
:GA	Gabon	241	7	None	00	
:GB	United Kingdom	44	10	0	00	
:GD	Grenada	1	10	None	011	NANP, delegat to US
:GE	Georgia	995	9	0	00	
:GF	French Guiana	594	9	0	00	
:GG	Guernsey	44	10	0	00	Delegat to GB
:GH	Ghana	233	9	0	00	
:GI	Gibraltar	350	8	None	00	
:GL	Greenland	299	6	None	00	
:GM	Gambia	220	7	None	00	
:GN	Guinea	224	9	None	00	
:GP	Guadeloupe	590	9	0	00	
:GQ	Equatorial Guinea	240	9	None	00	
:GR	Greece	30	10	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:GS`	South Georgia	`500`	5	None	00	Delegat to FK
`:GT`	Guatemala	`502`	8	None	00	
`:GU`	Guam	`1`	10	None	011	NANP, delegat to US
`:GW`	Guinea-Bissau	`245`	7	None	00	
`:GY`	Guyana	`592`	7	None	001	
`:HK`	Hong Kong	`852`	8	None	001	
`:HM`	Heard and McDonald Islands	`61`	9	0	0011	Delegat to AU
`:HN`	Honduras	`504`	8	None	00	
`:HR`	Croatia	`385`	9	0	00	
`:HT`	Haiti	`509`	8	None	00	
`:HU`	Hungary	`36`	9	06	00	
`:ID`	Indonesia	`62`	10-12	0	001	
`:IE`	Ireland	`353`	9	0	00	
`:IL`	Israel	`972`	9	0	00	
`:IM`	Isle of Man	`44`	10	0	00	Delegat to GB

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:IN`	India	`91`	10	0	00	
`:IO`	British Indian Ocean Territory	`246`	7	None	00	
`:IQ`	Iraq	`964`	10	0	00	
`:IR`	Iran	`98`	10	0	00	
`:IS`	Iceland	`354`	7	None	00	
`:IT`	Italy	`39`	9-10	None	00	Leading retained for landline
`:JE`	Jersey	`44`	10	0	00	Delegat to GB
`:JM`	Jamaica	`1`	10	None	011	NANP, delegat to US
`:JO`	Jordan	`962`	9	0	00	
`:JP`	Japan	`81`	10	0	010	
`:KE`	Kenya	`254`	9	0	000	
`:KG`	Kyrgyzstan	`996`	9	0	00	
`:KH`	Cambodia	`855`	8-9	0	001	
`:KI`	Kiribati	`686`	5	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:KM	Comoros	269	7	None	00	
:KN	Saint Kitts and Nevis	1	10	None	011	NANP, delegat to US
:KP	North Korea	850	10	0	00	
:KR	South Korea	82	9-10	0	001	
:KW	Kuwait	965	8	None	00	
:KY	Cayman Islands	1	10	None	011	NANP, delegat to US
:KZ	Kazakhstan	7	10	8	810	
:LA	Laos	856	10	0	00	
:LB	Lebanon	961	8	0	00	
:LC	Saint Lucia	1	10	None	011	NANP, delegat to US
:LI	Liechtenstein	423	7	None	00	
:LK	Sri Lanka	94	9	0	00	
:LR	Liberia	231	7-9	None	00	
:LS	Lesotho	266	8	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:LT	Lithuania	370	8	8	00	
:LU	Luxembourg	352	9	None	00	
:LV	Latvia	371	8	None	00	
:LY	Libya	218	9	0	00	
:MA	Morocco	212	9	0	00	
:MC	Monaco	377	8-9	None	00	
:MD	Moldova	373	8	0	00	
:ME	Montenegro	382	8	0	00	
:MF	Saint Martin	590	9	0	00	
:MG	Madagascar	261	9	0	00	
:MH	Marshall Islands	692	7	None	011	
:MK	North Macedonia	389	8	0	00	
:ML	Mali	223	8	None	00	
:MM	Myanmar	95	8-10	0	00	
:MN	Mongolia	976	8	None	001	
:MO	Macau	853	8	None	00	
:MP	Northern Mariana Islands	1	10	None	011	NANP, delegat

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
						to US
`:MQ`	Martinique	`596`	9	0	00	
`:MR`	Mauritania	`222`	8	None	00	
`:MS`	Montserrat	`1`	10	None	011	NANP, delegat to US
`:MT`	Malta	`356`	8	None	00	
`:MU`	Mauritius	`230`	8	None	00	
`:MV`	Maldives	`960`	7	None	00	
`:MW`	Malawi	`265`	9	0	00	
`:MX`	Mexico	`52`	10	None	00	
`:MY`	Malaysia	`60`	9-10	0	00	
`:MZ`	Mozambique	`258`	9	None	00	
`:NA`	Namibia	`264`	9	0	00	
`:NC`	New Caledonia	`687`	6	None	00	
`:NE`	Niger	`227`	8	None	00	
`:NF`	Norfolk Island	`672`	5-6	None	00	
`:NG`	Nigeria	`234`	10	0	009	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:NI`	Nicaragua	`505`	8	None	00	
`:NL`	Netherlands	`31`	9	0	00	
`:NO`	Norway	`47`	8	None	00	
`:NP`	Nepal	`977`	10	0	00	
`:NR`	Nauru	`674`	7	None	00	
`:NU`	Niue	`683`	4	None	00	
`:NZ`	New Zealand	`64`	8-9	0	00	
`:OM`	Oman	`968`	8	None	00	
`:PA`	Panama	`507`	8	None	00	
`:PE`	Peru	`51`	9	0	00	
`:PF`	French Polynesia	`689`	8	None	00	
`:PG`	Papua New Guinea	`675`	8	None	05	
`:PH`	Philippines	`63`	10	0	00	
`:PK`	Pakistan	`92`	10	0	00	
`:PL`	Poland	`48`	9	None	00	
`:PM`	Saint Pierre and Miquelon	`508`	6	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:PN`	Pitcairn Islands	`64`	8-9	0	00	Delegat to NZ
`:PR`	Puerto Rico	`1`	10	None	011	NANP, delegat to US
`:PS`	Palestine	`970`	9	0	00	
`:PT`	Portugal	`351`	9	None	00	
`:PW`	Palau	`680`	7	None	011	
`:PY`	Paraguay	`595`	9	0	00	
`:QA`	Qatar	`974`	8	None	00	
`:RE`	Reunion	`262`	9	0	00	
`:RO`	Romania	`40`	9	0	00	
`:RS`	Serbia	`381`	9	0	00	
`:RU`	Russia	`7`	10	8	810	
`:RW`	Rwanda	`250`	9	0	00	
`:SA`	Saudi Arabia	`966`	9	0	00	
`:SB`	Solomon Islands	`677`	5-7	None	00	
`:SC`	Seychelles	`248`	7	None	00	
`:SD`	Sudan	`249`	9	0	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
:SE	Sweden	46	9	0	00	
:SG	Singapore	65	8	None	001	
:SH	Saint Helena	290	4-5	None	00	
:SI	Slovenia	386	8	0	00	
:SJ	Svalbard	47	8	None	00	Delegat to NO
:SK	Slovakia	421	9	0	00	
:SL	Sierra Leone	232	8	0	00	
:SM	San Marino	378	10	None	00	
:SN	Senegal	221	9	None	00	
:SO	Somalia	252	8	None	00	
:SR	Suriname	597	7	None	00	
:SS	South Sudan	211	9	0	00	
:ST	Sao Tome and Principe	239	7	None	00	
:SV	El Salvador	503	8	None	00	
:SX	Sint Maarten	1721	7	None	00	
:SY	Syria	963	9	0	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:SZ`	Eswatini	`268`	8	None	00	
`:TC`	Turks and Caicos	`1`	10	None	011	NANP, delegat to US
`:TD`	Chad	`235`	8	None	00	
`:TG`	Togo	`228`	8	None	00	
`:TH`	Thailand	`66`	9	0	001	
`:TJ`	Tajikistan	`992`	9	None	810	
`:TK`	Tokelau	`690`	4	None	00	
`:TL`	Timor-Leste	`670`	7-8	None	00	
`:TM`	Turkmenistan	`993`	8	8	810	
`:TN`	Tunisia	`216`	8	None	00	
`:TO`	Tonga	`676`	5-7	None	00	
`:TR`	Turkey	`90`	10	0	00	
`:TT`	Trinidad and Tobago	`1`	10	None	011	NANP, delegat to US
`:TV`	Tuvalu	`688`	5	None	00	
`:TW`	Taiwan	`886`	9	0	002	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:TZ`	Tanzania	`255`	9	0	00	
`:UA`	Ukraine	`380`	9	0	00	
`:UG`	Uganda	`256`	9	0	00	
`:US`	United States	`1`	10	None	011	NANP base rules
`:UY`	Uruguay	`598`	8	0	00	
`:UZ`	Uzbekistan	`998`	9	None	810	
`:VA`	Vatican City	`39`	9-10	None	00	Delegat to IT
`:VC`	Saint Vincent and the Grenadines	`1`	10	None	011	NANP, delegat to US
`:VE`	Venezuela	`58`	10	0	00	
`:VG`	British Virgin Islands	`1`	10	None	011	NANP, delegat to US
`:VI`	US Virgin Islands	`1`	10	None	011	NANP, delegat to US
`:VN`	Vietnam	`84`	9-10	0	00	
`:VU`	Vanuatu	`678`	5-7	None	00	

Code	Country/Territory	E.164 Prefix	National Length	Trunk	IDD	Notes
`:WF`	Wallis and Futuna	`681`	6	None	00	
`:WS`	Samoa	`685`	5-7	None	0	
`:XK`	Kosovo	`383`	8	0	00	
`:YE`	Yemen	`967`	9	0	00	
`:YT`	Mayotte	`262`	9	0	00	Delegat to RE
`:ZA`	South Africa	`27`	9	0	00	
`:ZM`	Zambia	`260`	9	0	00	
`:ZW`	Zimbabwe	`263`	9	0	00	

Special Translation Behaviors

1. CLI Blocking Prefix Stripping

Before format translation, CLI blocking prefixes are removed:

```
Input:  *67555123456
Step 1: Strip *67 → 555123456
Step 2: Translate → 1555123456 (if US)
```

2. SIP Parameter Stripping

Parameters after semicolons are removed:

```
Input:  61403123456;npdi;rn=+61400000000
Step 1: Strip ;npdi;rn=... → 61403123456
Step 2: Translate → 61403123456
```

3. Non-Digit Character Removal

All non-digit characters (except +) are stripped:

```
Input:  +61 (403) 123-456
Step 1: Strip formatting → +61403123456
Step 2: Translate → 61403123456
```

Variables Set After Translation

Variable	Value	Description
`destination_number`	E.164 format	Normalized destination number
`tas_destination_number`	E.164 format	Same as `destination_number` (both set for compatibility)
`effective_caller_id_number`	E.164 format	Normalized source number (MO calls)

What Happens When Translation Fails

Scenario: Undefined Country Code

```
config :tas, number_translate: %{country_code: :XX}  # Invalid
```

Result: `{:error, "Undefined Country Code"}` - call rejected

Scenario: Invalid Number Format

```
Input: "abc123" (contains letters)
Step 1: Strip non-digits → "123"
Step 2: Too short, cannot match any pattern
Result: May pass through as-is or reject based on dialplan logic
```

Best Practice: Always validate subscriber provisioning with correct E.164 numbers in HSS.

Testing Number Translation

Web UI Translation Tester (`/translate`):

1. Navigate to `/translate` in Control Panel
2. Select country code from dropdown
3. Enter test number in any format
4. View translated E.164 output
5. Test multiple formats to validate

Common Test Scenarios:

- Local short codes → E.164
- National format (0NSN) → E.164
- International format (+CC) → E.164
- Numbers with CLI prefixes → stripped and translated
- Numbers with formatting (spaces, dashes) → clean E.164

Troubleshooting Number Translation

Problem: Calls failing with "UNALLOCATED_NUMBER"

1. **Check translated number format:**
    - Use `/translate` tool to test number
    - Verify output matches expected E.164 format
    - Confirm country code and area code are correct
2. **Check Sh lookup:**

- Translated number is used for Sh query
- Use `/sh_test` with translated number
- Verify subscriber exists with that MSISDN

3. **Check dialplan variables:**

- Review logs for `destination_number` value
- Confirm translation occurred before dialplan

Problem: Wrong area code applied

```
# Configuration
config :tas, number_translate: %{
  country_code: :AU,
  localAreaCode: "617"  # Wrong for your region
}

# Input: 12345678 (8-digit local)
# Output: 6161712345678 (incorrect - double area code)
# Fix: Set correct localAreaCode for your deployment
```

Problem: International numbers not recognized

Check if number includes country code:

- ✅ `+61403123456` or `61403123456` → Recognized
- ❌ `0403123456` in wrong country_code config → Misrouted

MO vs MT Translation Behavior

MT (Mobile Terminated) Calls:

- Only destination number (called party) is translated
- Source number (caller) passed through as-is from SIP
- Destination used for Sh lookup of called subscriber

MO (Mobile Originating) Calls:

- Destination number (called party) translated

- Source number (calling party) also translated
- Source used for Sh lookup of calling subscriber
- Both numbers normalized for consistent logging/CDR

Best Practices

1. **Use Correct Country Code:**
    - Set `country_code` to match your deployment region
    - Test thoroughly before production
2. **Configure Appropriate Local Area Code:**
    - `localAreaCode` should match your network's default area
    - Used for short numbers without area code
3. **Test All Number Formats:**
    - Local (short codes)
    - National (0NSN format)
    - International (+CC format)
    - Special service numbers (emergency, voicemail)
4. **Monitor Translation Logs:**
    - Check for "Undefined Country Code" errors
    - Watch for unexpected number formats
    - Validate E.164 output matches expectations
5. **Document Your Numbering Plan:**
    - Define which formats subscribers will use
    - Test each format in `/translate` tool
    - Train operations staff on expected formats

Online Charging System (OCS) Integration

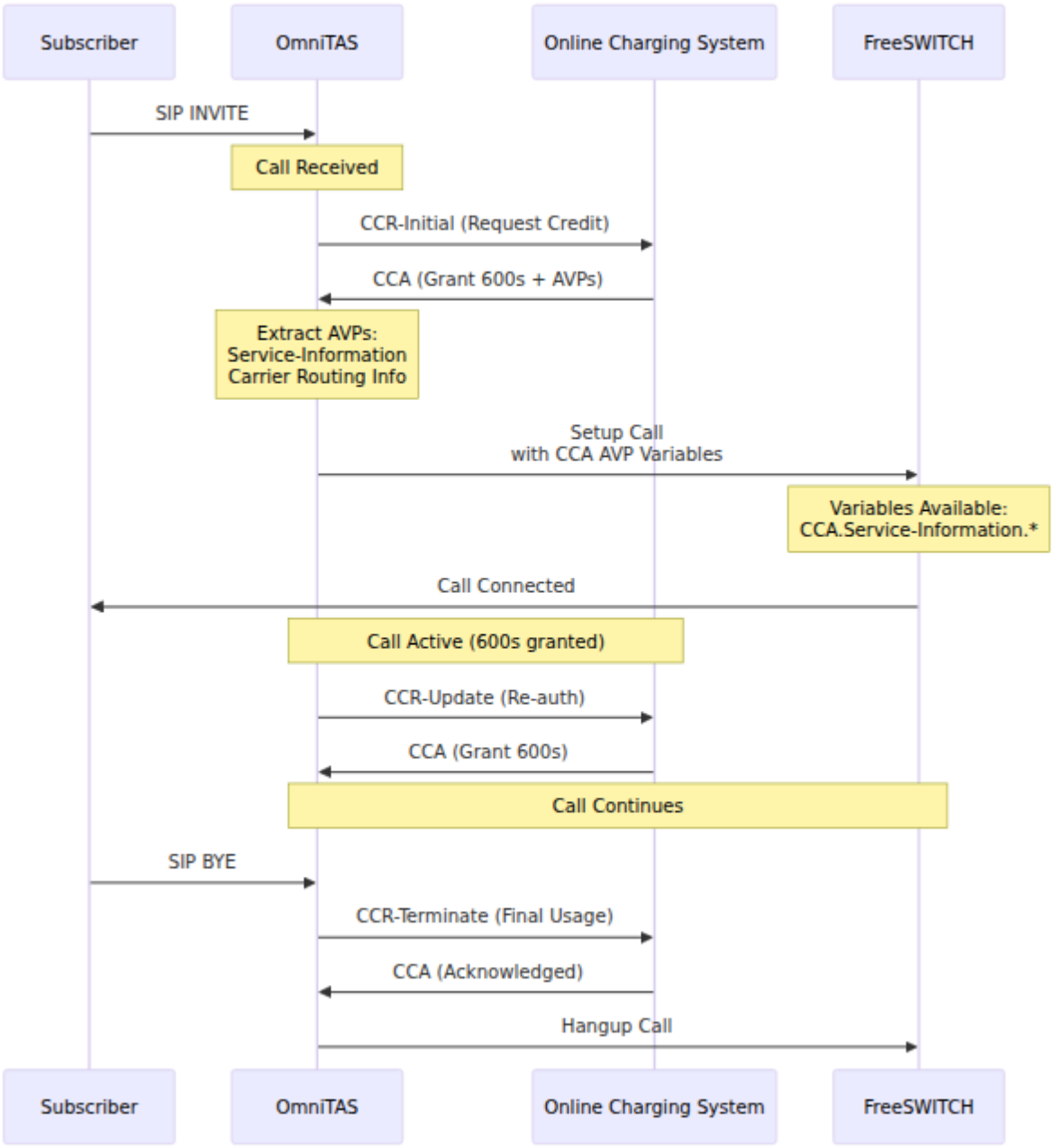
Comprehensive guide to OmniTAS integration with Online Charging Systems via Diameter Ro interface, including real-time credit control, AVP extraction, and FreeSWITCH variable mapping.

Table of Contents

- Architecture Overview
- What Controls Whether a Call Is Charged
- Credit Control Flow
- AVP Parsing and Variable Mapping
- Configuration
- FreeSWITCH Integration
- Answer & Hangup Notifications
- Diameter Messages
- Metrics
- Troubleshooting
- Reference
  - FreeSWITCH Channel Variables
  - AVP Codes Reference

Architecture Overview

OmniTAS implements the Diameter Ro interface per 3GPP TS 32.299 for real-time online charging. The system authorizes calls by requesting credit from an OCS before call setup, monitors credit during the call, and reports final usage on termination.



Key Components

Credit-Control-Request (CCR):

- **CCR-Initial (Type 1)**: Sent before call setup to request initial credit authorization
- **CCR-Update (Type 2)**: Sent during active calls for re-authorization or interim updates
- **CCR-Terminate (Type 3)**: Sent on call termination with final usage reporting

Credit-Control-Answer (CCA):

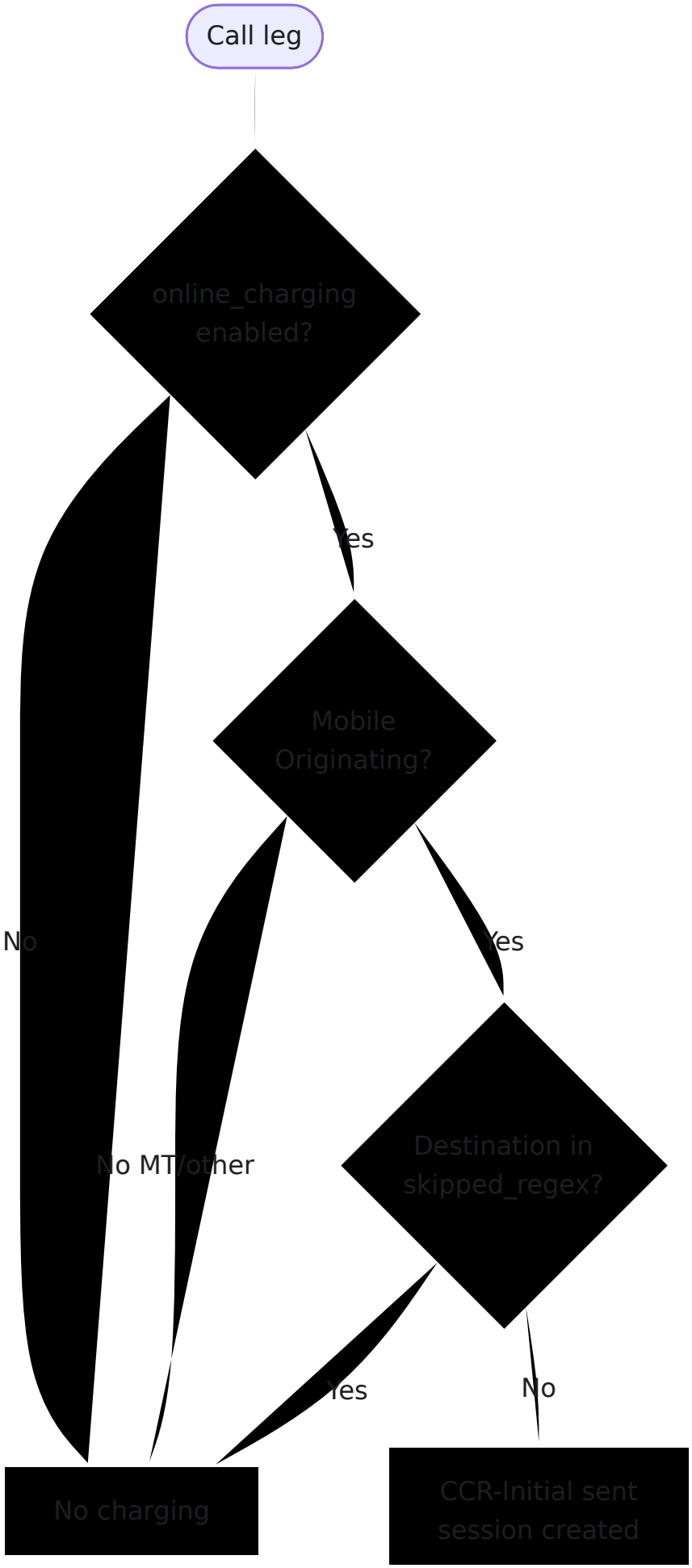
- Contains granted service units (time quota in seconds)
- Includes vendor-specific AVPs with additional charging data
- Provides routing information, charged party details, and service identifiers

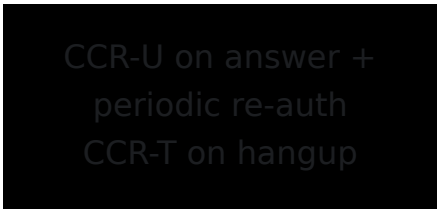
What Controls Whether a Call Is Charged

A call produces Diameter CCRs **only if all three** of the following conditions hold. They are evaluated when OmniTAS builds the Mobile Originating (MO) dialplan for the call:

1. **Online charging is enabled** — `online_charging.enabled` is `true`. When `false`, OmniTAS authorizes every call locally and never contacts the OCS.
2. **The call is Mobile Originating (MO)** — only the MO leg is charged. **Mobile Terminating (MT) calls are never charged**: the MT path does not perform OCS authorization, so no credit-control session is created for it.
3. **The destination is not exempt** — the dialled number does not match any pattern in `skipped_regex` (e.g. emergency numbers, service codes).

If all three hold, OmniTAS sends a **CCR-Initial** and opens a credit-control session keyed by the SIP Call-ID. From that point, **CCR-Update** and **CCR-Terminate** are driven by answer/hangup events (see Answer & Hangup Notifications) and are only generated for call legs that have an open session. MT legs, bridged B-legs, and exempt calls therefore never produce CCRs.

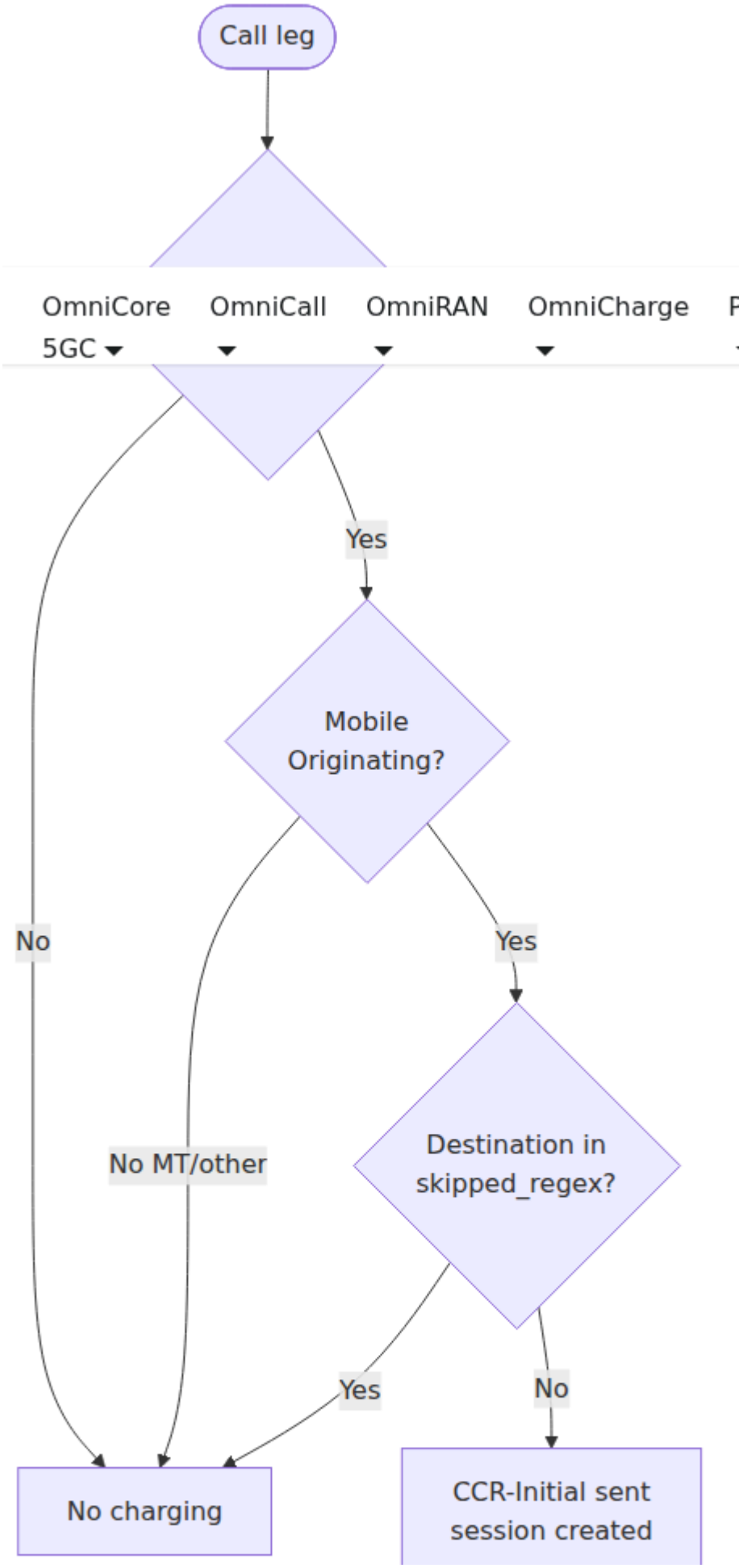

Call leg
online_charging enabled?
Yes
No
Mobile Originating?
Yes
No MT/other
Destination in skipped_regex?
Yes
No
No charging
CCR-Initial sent session created

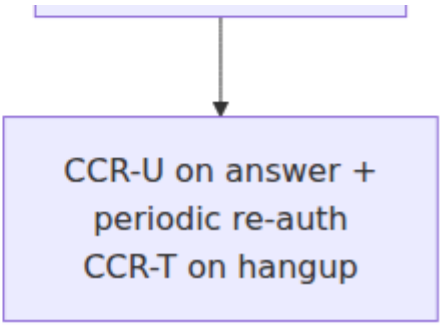


> **Note:** Reaching the OCS does not guarantee the call connects. A successful authorization with **zero** granted units bars the call — see Credit Exhaustion and Call Barring. Barring happens *after* a CCR is sent; it is distinct from the conditions above, which decide whether a CCR is sent at all.

Credit Control Flow

Call Authorization Sequence


Call leg
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Yes
Mobile Originating?
No
Yes
No MT/other
Destination in skipped_regex?
Yes
No
No charging
CCR-Initial sent session created



Credit Exhaustion and Call Barring

Call barring at setup. If the OCS returns a successful CCA (Result-Code `2001`) but with **zero** granted units — or an explicit denial such as `4012` (DIAMETER_CREDIT_LIMIT_REACHED) or `4010` (DIAMETER_END_USER_SERVICE_DENIED) — OmniTAS treats the call as having no credit and bars it with hangup cause `OUTGOING_CALL_BARRED`. The subscriber never connects. This is the expected outcome for an unprovisioned or out-of-credit subscriber.

Mid-call exhaustion. If credit is granted at setup but later runs out (a periodic CCR-Update returns zero units or a credit-limit error), OmniTAS terminates the in-progress call. With `credit_exhaustion_announcement` configured it transfers the caller to an announcement before hangup; otherwise it hangs up directly.

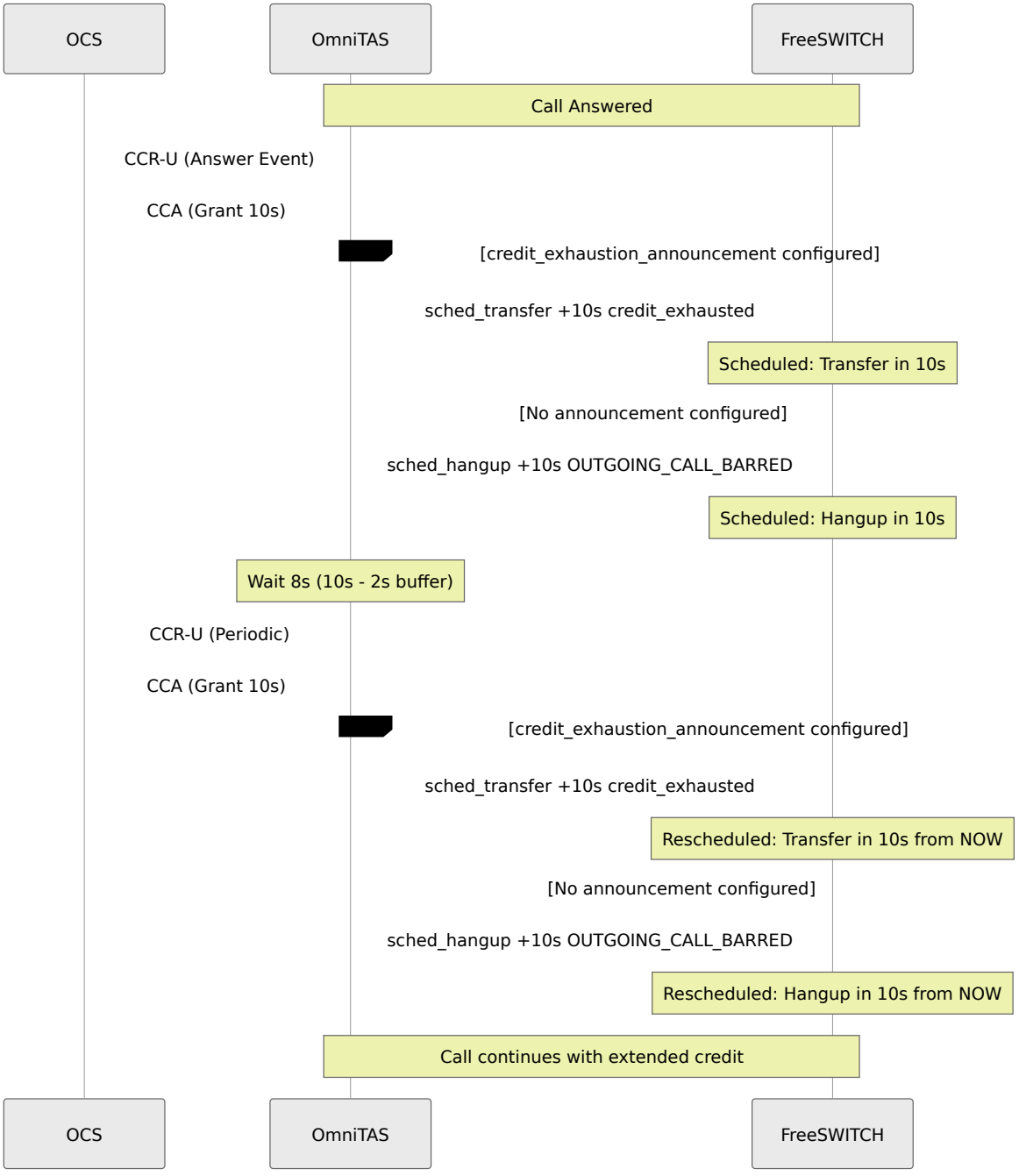
CCR-Terminate fires at the exhaustion moment, before any cleanup. When the balance is exhausted, OmniTAS sends the CCR-Terminate **immediately** (charged answer→exhaustion), *then* plays the out-of-credit announcement or hangs up. This is deliberate: the time the subscriber spends listening to the "you're out of balance" prompt is **not chargeable**, so the CDR must end when credit ran out — not when the user hangs up from the recording. The session terminates at that point, so the eventual FreeSWITCH hangup for the call finds no session and sends nothing (idempotent — no second CCR-T). This holds whether cleanup is a transfer-to-announcement or a direct hangup, so it does not depend on the FreeSWITCH hangup cause.

OmniTAS supports multiple mechanisms for handling credit exhaustion, with automatic integration between scheduled hangups and credit exhaustion announcements.

Scheduled Hangup with Dynamic Rescheduling

When `schedule_hangup_auth` is enabled, OmniTAS schedules a FreeSWITCH timer that automatically terminates calls when granted credit expires. This timer is **dynamically rescheduled** every time new credit is granted via CCR-Update responses.

How it works:



Buffer Logic:

OmniTAS sends CCR-Update messages **before** the granted credit expires to ensure continuous service. The buffer time is configurable via

`ccr_update_buffer_seconds` (default: 2 seconds).

Example timeline:

- **T+0s**: Call answered, OCS grants 10s, timer scheduled for T+10s
- **T+8s**: CCR-U sent (10s - 2s buffer)
- **T+8.1s**: OCS grants 10s, timer rescheduled to T+18.1s (10s from now)
- **T+16.1s**: CCR-U sent
- **T+16.2s**: OCS grants 10s, timer rescheduled to T+26.2s
- Call continues as long as OCS keeps granting credit

Logs to watch:

```
[OCS HANGUP RESCHEDULE] Found UUID <uuid> for call <id> -
rescheduling timer to 10s from now
[SCHED TRANSFER] Scheduling transfer to credit_exhausted dialplan
for <uuid> in 10s
[OCS HANGUP RESCHEDULE] Successfully rescheduled timer for call
<id> (UUID: <uuid>)
```

Integration: schedule_hangup_auth + credit_exhaustion_announcement

When **both** features are enabled, OmniTAS automatically uses scheduled **transfers** instead of direct hangups, allowing the caller to hear an announcement before call termination.

Without announcement configured:

```
config :tas, :online_charging,
  schedule_hangup_auth: true,
  credit_exhaustion_announcement: nil
```

→ Uses `sched_hangup` - direct hangup when credit expires

With announcement configured:

```
config :tas, :online_charging,
  schedule_hangup_auth: true,
  credit_exhaustion_announcement:
"${base_dir}/sounds/en/us/callie/misc/8000/credit_exhausted.wav"
```

→ Uses `sched_transfer` - transfers to `credit_exhausted` dialplan which plays announcement then hangs up

How the transfer works:

1. OmniTAS sets `tas_call_reason=credit_exhausted` channel variable
2. Schedules transfer to `credit_exhausted` extension in `ims_as` dialplan context
3. When timer fires:
   - FreeSWITCH transfers A-leg to credit_exhausted dialplan
   - Bridge breaks automatically, B-leg receives BYE
   - Dialplan plays announcement to A-leg
   - Call terminates after announcement

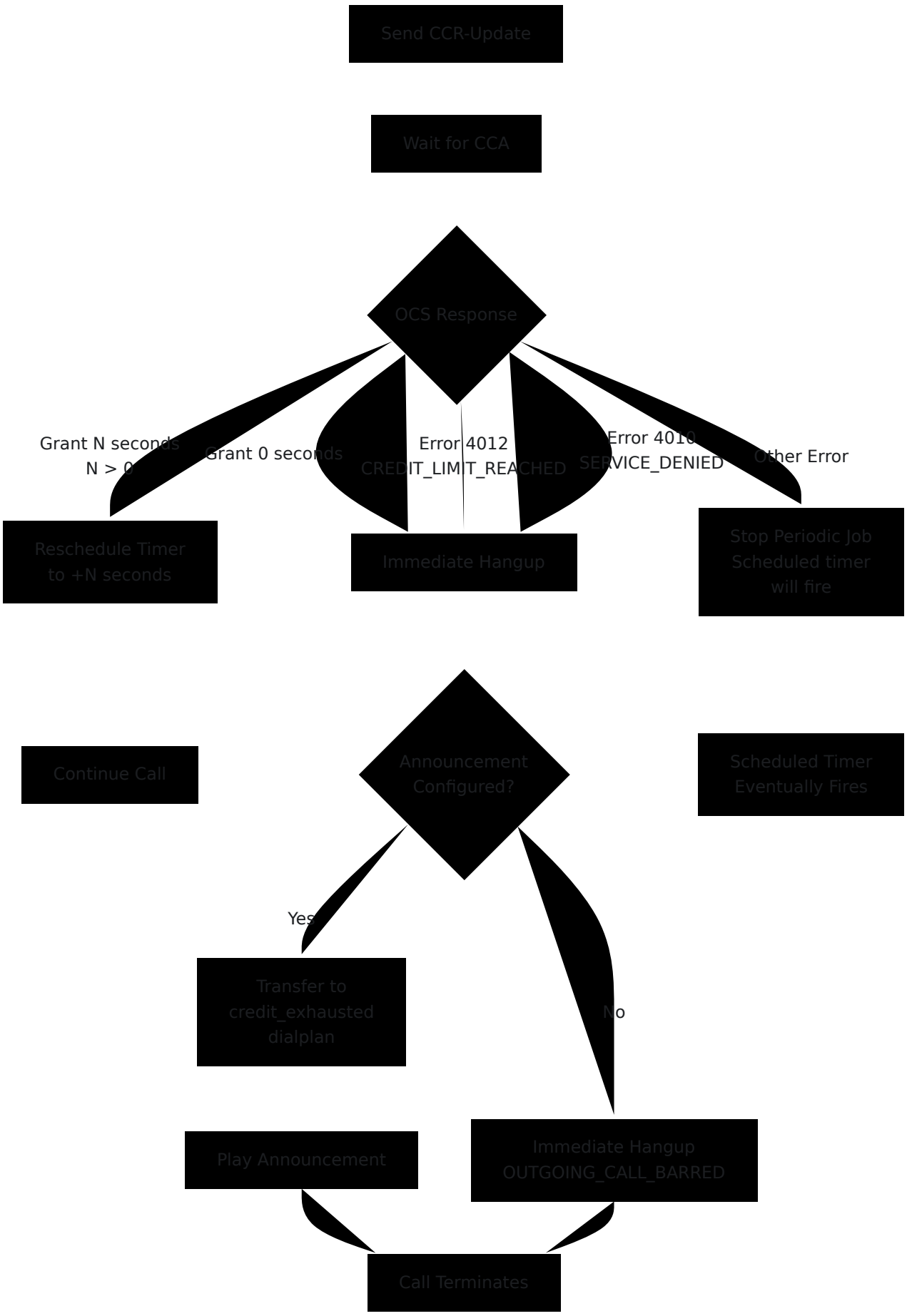
Benefits:

- Caller hears professional announcement instead of abrupt disconnect
- B-leg (called party) doesn't hear announcement
- CCR-T still sent with actual usage
- Announcement path: Must be relative to FreeSWITCH base directory (use `${base_dir}` variable)

Immediate Credit Exhaustion During CCR-Update

If the OCS **denies credit** or returns **zero seconds** during a CCR-Update, OmniTAS immediately triggers credit exhaustion handling, overriding any scheduled timer.

OCS Response Scenarios:



Handled Error Codes:

OCS Response	Action	Logs
`{:ok, 0}` (Zero seconds)	Immediate credit exhaustion hangup	`Credit exhausted (zero seconds allocated) - triggering immediate hangup`
`{:error, 4012}` (CREDIT_LIMIT_REACHED)	Immediate credit exhaustion hangup	`Credit exhausted (4012 CREDIT_LIMIT_REACHED) - triggering immediate hangup`
`{:error, 4010}` (END_USER_SERVICE_DENIED)	Immediate credit exhaustion hangup	`Service denied (4010 END_USER_SERVICE_DENIED) - triggering immediate hangup`
`{:error, reason}` (Other errors)	Stop periodic CCR job, scheduled timer fires	`Periodic CCR failed with error <reason> - Stopping job`
`{:ok, N}` where N > 0	Reschedule timer to +N seconds	`Periodic CCA allocated Ns, will send next CCR-U in (N-buffer)s`

Priority: Immediate credit exhaustion handling **wins** over scheduled timer. If OCS denies credit at T+8s but timer was scheduled for T+10s, the immediate hangup at T+8s occurs and the scheduled timer becomes irrelevant.

Example timeline with mid-call credit denial:

```
T+0s:   Call answered
T+0.1s: OCS grants 10s → Timer scheduled for T+10.1s
T+8s:   CCR-U sent (buffer = 2s)
T+8.1s: OCS returns 0 seconds → Immediate transfer to
credit_exhausted dialplan
T+8.2s: Announcement plays to caller
T+10s:  Call terminated (scheduled timer irrelevant)
```

Logs for immediate credit exhaustion:

```
[warning] Credit exhausted (zero seconds allocated) - triggering
immediate hangup
[warning] Hanging up call <id> (UUID: <uuid>) due to credit
exhaustion
[info] Credit exhaustion announcement config:
"${base_dir}/sounds/..."
[info] Playing announcement before hangup: ...
[info] Setting tas_call_reason=credit_exhausted for <uuid>
[info] Transferring to credit exhausted dialplan: uuid_transfer
<uuid> credit_exhausted XML ims_as
```

Summary: Credit Exhaustion Mechanisms

OmniTAS provides two complementary mechanisms:

1. **Scheduled Timer** (`schedule_hangup_auth`):
    - Automatic hangup/transfer when granted credit expires
    - Dynamically rescheduled on each CCR-U response
    - Uses buffer logic to send CCR-U before expiration
    - Integrates with announcement feature
2. **Immediate Exhaustion Handling**:
    - Triggered when OCS denies credit during CCR-U
    - Overrides scheduled timer
    - Supports announcement playback
    - Handles specific Diameter error codes

Both mechanisms respect the `credit_exhaustion_announcement` configuration and will play the configured audio before terminating calls when configured.

AVP Parsing and Variable Mapping

Overview

OmniTAS automatically extracts Attribute-Value Pairs (AVPs) from Credit-Control-Answer messages and makes them available to FreeSWITCH as channel variables. This enables dialplan logic to use OCS-provided data for routing decisions, billing purposes, or call treatment.

Supported AVP Types:

- Simple values (UTF8String, Unsigned32, Integer32)
- Grouped AVPs with nested structures
- Vendor-specific AVPs (e.g., 3GPP Service-Information)

Variable Naming Convention: AVPs are flattened into dot-notation channel variables with the prefix `CCA`:

```
CCA.<AVP-Name>.<Nested-AVP-Name>.<Value-AVP-Name> = "value"
```

Common AVP Mappings

Service-Information AVP (3GPP)

The Service-Information grouped AVP (AVP Code 873, Vendor-ID 10415) contains IMS-specific charging details:

Example OCS Response:

```
Service-Information
  ├── IMS-Information
  │     ├── Carrier-Select-Routing-Information: "1408"
  │     └── Node-Functionality: 6
  └── Alternate-Charged-Party-Address: "NickTest"
```

Resulting FreeSWITCH Variables:

```
CCA.Service-Information.Carrier-Select-Routing-Information =
"1408"
CCA.Service-Information.Alternate-Charged-Party-Address =
"NickTest"
```

Accessing in Dialplan: Variables use dot notation and hyphens as shown above:

```
<action application="log" data="INFO Carrier: ${CCA.Service-
Information.Carrier-Select-Routing-Information}"/>
```

Viewing with uuid_dump: In FreeSWITCH console or ESL, variables appear with the `variable_` prefix:

```
variable_CCA.Service-Information.Carrier-Select-Routing-
Information: 1408
variable_CCA.Service-Information.Alternate-Charged-Party-Address:
NickTest
```

Note: FreeSWITCH preserves dots and hyphens in variable names. The variables work in all dialplan contexts and applications.

Granted-Service-Unit AVP

Time quotas are extracted and made available:

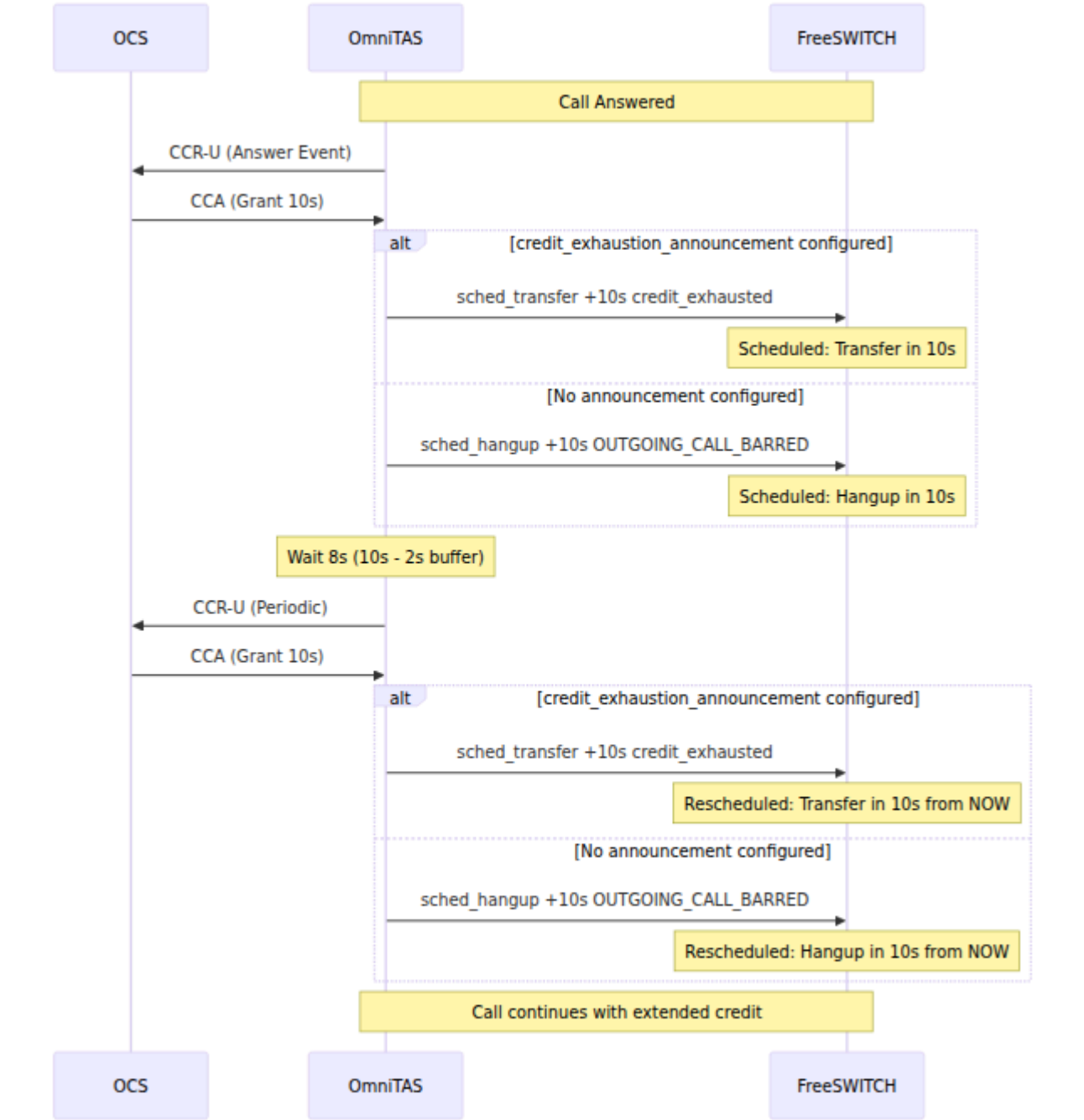
OCS Response:

```
Granted-Service-Unit
  └── CC-Time: 600
```

Variable:

```
allocated_time = 600
```

AVP Processing Logic



Processing Rules:

1. **Grouped AVPs** add a level to the variable name hierarchy but have no value themselves
2. **Simple AVPs** are mapped to variables with their full dotted path
3. **Vendor-Specific AVPs** are processed identically to standard AVPs
4. **Unknown AVPs** are safely skipped without errors

Example: Multi-Level Nesting

OCS CCA Structure:

```
Service-Information (Grouped)
  ├── IMS-Information (Grouped)
  │     ├── Node-Functionality: 6
  │     ├── Role-Of-Node: 1
  │     ├── Calling-Party-Address: "tel:+313380000000670"
  │     └── Time-Stamps (Grouped)
  │           ├── SIP-Request-Timestamp: "2026-01-24T22:40:18Z"
  │           └── SIP-Response-Timestamp: "2026-01-24T22:40:18Z"
  └── IN-Information (Grouped)
        └── Real-Called-Number: "24724741234"
```

FreeSWITCH Variables Created:

```
CCA.Service-Information.IMS-Information.Node-Functionality = "6"
CCA.Service-Information.IMS-Information.Role-Of-Node = "1"
CCA.Service-Information.IMS-Information.Calling-Party-Address =
"tel:+313380000000670"
CCA.Service-Information.IMS-Information.Time-Stamps.SIP-Request-
Timestamp = "2026-01-24T22:40:18Z"
CCA.Service-Information.IMS-Information.Time-Stamps.SIP-Response-
Timestamp = "2026-01-24T22:40:18Z"
CCA.Service-Information.IN-Information.Real-Called-Number =
"24724741234"
```

Configuration

Online Charging Parameters

Parameter	Type	Required	Default	
`enabled`	Boolean	No	`false`	M W au Se Is
`requested_units_seconds`	Integer	No	`0`	Ta re Se In **ar** bu O(m Vi re ov re 4(ra
`report_and_reserve`	Boolean	No	`false`	3(m `t` Te Ur cc (ir Re `R/`

Parameter	Type	Required	Default	

				th
				`QU`
				up
				Th
				ar
				la
				`fa`
				(c
				re
				`C(`
				ar

`service_identifier`	Integer	No	`1`	St
				`Id`
				ev
				`{S`
				`Id`
				fix

`periodic_ccr_time_seconds`	Integer	No	`10`	Fa be w us th dy cr `cc`
`ccr_update_buffer_seconds`	Integer	No	`2`	Sa gr se `(a` cr ou

Parameter	Type	Required	Default	
`schedule_hangup_auth`	Boolean	No	`false`	En ha cr On tir fr or w c

`credit_exhaustion_announcement`	String	No	`nil`	Au
				ex
				co
				`s`
				sc
				ar
				W
				ar
				cr
				us
				"
				`n`
				ar

`skipped_regex`	List[String]	No	`[]`	Li ag M er Us se "

Diameter Connection Parameters

Parameter	Type	Required	Default	Des
`origin_host`	String	Yes	-	OmniTAS Diameter Id unique across your Di Example: `"tas01.epc.mnc123.m`
`origin_realm`	String	Yes	-	OmniTAS Diameter Re decisions. Example: `"epc.mnc123.mcc456.`
`destination_realm`	String	Yes	-	OCS Diameter Realm. peers in this realm.
`destination_host`	String	No	`nil`	Specific OCS Diamete routing based on `des` when direct routing to required.

Configuration Example

```
config :tas, :online_charging,
  # Master switch
  enabled: true,

  # 0 = empty RSU, let the OCS decide the grant (matches Nokia TAS
/ CGRateS).
  # Set a positive value only for a standards OCS that rates on
the requested units.
  requested_units_seconds: 0,

  # Stable provisioned service identifier
  service_identifier: 1,

  # Fallback re-auth interval (normal timing is dynamic from the
grant)
  periodic_ccr_time_seconds: 10,

  # Re-authorize 2s before granted credit expires
  ccr_update_buffer_seconds: 2,

  # Schedule hangup based on granted credit
  schedule_hangup_auth: true,

  # Play announcement before credit exhaustion hangup
  credit_exhaustion_announcement: "ivr/ivr-
account_balance_low.wav",

  # Skip OCS for emergency calls and voicemail
  skipped_regex: [
    "^911$",      # Emergency (US)
    "^000$",      # Emergency (AU)
    "^\*86$"     # Voicemail access
  ]

config :tas, :diameter,
  # Service identity
  origin_host: "tas01.epc.mnc001.mcc001.3gppnetwork.org",
  origin_realm: "epc.mnc001.mcc001.3gppnetwork.org",

  # OCS routing
```

```
    destination_realm: "epc.mnc001.mcc001.3gppnetwork.org",
    destination_host: nil  # Realm-based routing
```

How it works:

When a call is received:

1. Destination number is checked against `skipped_regex` patterns
2. If matched, call bypasses OCS (useful for emergency services)
3. If not matched, CCR-Initial sent to OCS at `destination_realm`
4. CCA response is parsed for granted units and AVPs
5. AVPs are mapped to FreeSWITCH variables (see AVP Mapping)
6. Call proceeds with `allocated_time` and AVP data available
7. CCR-Update sent every `periodic_ccr_time_seconds` during call
8. If `schedule_hangup_auth` enabled, automatic hangup when credit expires
9. CCR-Terminate sent on call completion

Use cases:

- **Basic OCS**: Enable with defaults for standard credit control
- **High-value calls**: Reduce `periodic_ccr_time_seconds` to 30s for frequent re-auth
- **Prepaid service**: Enable `schedule_hangup_auth` and set `credit_exhaustion_announcement`
- **Emergency compliance**: Add emergency numbers to `skipped_regex` to ensure always connected

FreeSWITCH Integration

Accessing AVP Variables in Dialplan

AVP data extracted from CCA messages is available as channel variables in FreeSWITCH dialplan:

```
<extension name="Route_with_OCS_Data">
  <condition field="destination_number" expression="^(.+)$">

    <!-- Access carrier routing info from OCS -->
    <action application="log"
            data="INFO Carrier Code: ${CCA.Service-
Information.Carrier-Select-Routing-Information}"/>

    <!-- Access charged party from OCS -->
    <action application="log"
            data="INFO Charged Party: ${CCA.Service-
Information.Alternate-Charged-Party-Address}"/>

    <!-- Access granted time -->
    <action application="log"
            data="INFO Allocated Time: ${allocated_time}
seconds"/>

    <!-- Route based on carrier code -->
    <action application="set"
            data="carrier_code=${CCA.Service-Information.Carrier-
Select-Routing-Information}"/>
    <action application="bridge"

data="sofia/external/$1@carrier-${carrier_code}.sip.example.com"/>

  </condition>
</extension>
```

Variable Availability

Timing:

- Variables are set **before** FreeSWITCH call setup
- Available throughout entire call duration
- Persist across call transfers and updates

Scope:

- Channel-scoped (specific to individual call leg)

- Not inherited by bridged/transferred legs
- Safe to use in all dialplan applications

Example Use Cases

1. Carrier Selection Based on OCS Data

Use OCS-provided carrier code to route calls:

```
<extension name="Carrier_Selection">
  <condition field="${CCA.Service-Information.Carrier-Select-
Routing-Information}" expression="^(.+)$">
    <action application="bridge"

data="sofia/external/${destination_number}@carrier-$1.example.com"/>
  </condition>

  <!-- Fallback if no carrier specified -->
  <condition field="${CCA.Service-Information.Carrier-Select-
Routing-Information}" expression="^$">
    <action application="bridge"
            data="sofia/external/${destination_number}@default-
carrier.example.com"/>
  </condition>
</extension>
```

How it works: OCS returns carrier code "1408" in Service-Information AVP. FreeSWITCH routes call to `carrier-1408.example.com` gateway based on this data.

2. Alternate Billing Party

Route billing to a different party based on OCS response:

```
<extension name="Alternate_Billing">
  <condition field="${CCA.Service-Information.Alternate-Charged-
Party-Address}" expression="^(.+)$">

    <!-- Log billing party for CDRs -->
    <action application="set"
            data="billed_party=$1"/>
    <action application="export"
            data="billed_party=$1"/>

    <!-- Include in SIP headers -->
    <action application="set"
            data="sip_h_X-Billed-Party=$1"/>

    <action application="bridge"

data="sofia/external/${destination_number}@trunk.example.com"/>
  </condition>
</extension>
```

How it works: OCS specifies alternate charged party (e.g., corporate account). OmniTAS extracts "NickTest" from AVP and makes it available to dialplan for CDR recording and SIP header insertion.

3. Time-Limited Calls with Warnings

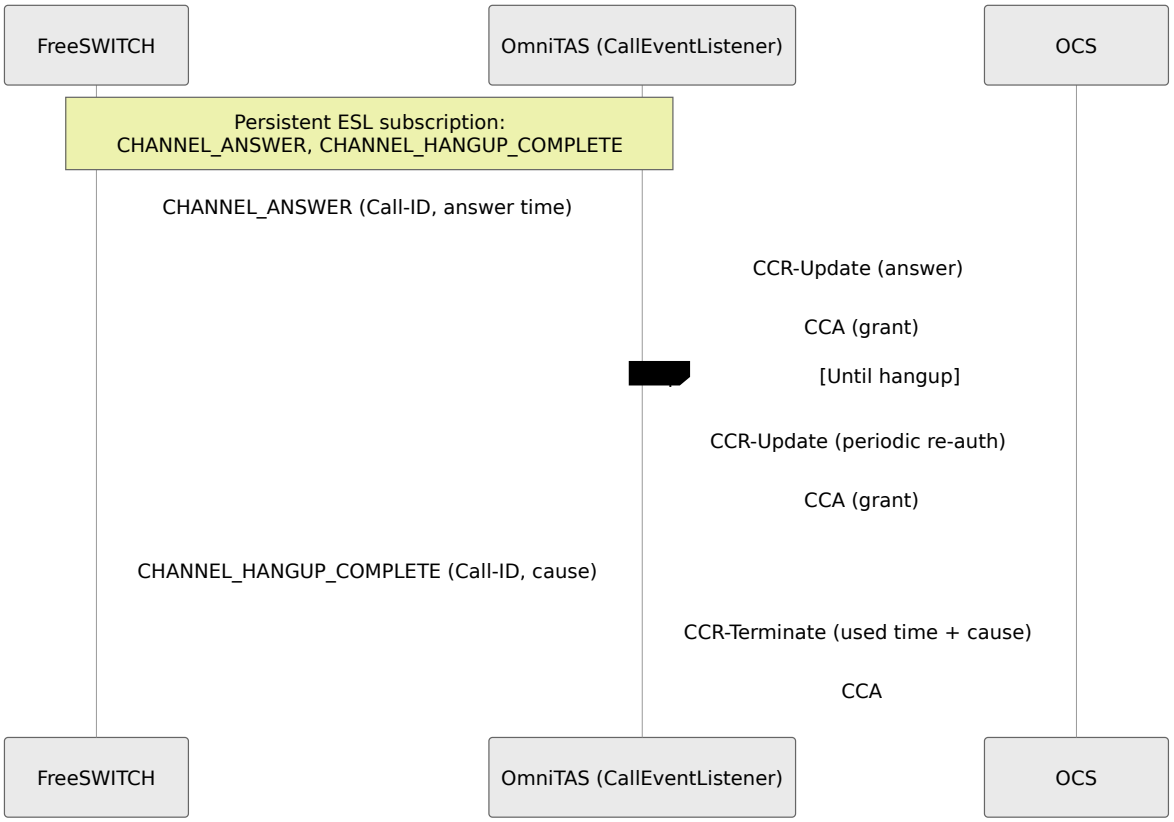
Provide warnings before credit expires:

```
<extension name="Credit_Warnings">
  <condition field="destination_number" expression="^(.+)$">

    <!-- Schedule warning 30 seconds before hangup -->
    <action application="set"
            data="warning_time=${expr(${allocated_time} - 30)}"/>

    <action application="sched_hangup"
            data="+${allocated_time} ALLOTTED_TIMEOUT"/>

    <action application="sched_broadcast"
            data="+${warning_time} playback::ivr/ivr-
account_balance_low.wav"/>

    <action application="bridge"
            data="sofia/external/$1@trunk.example.com"/>
  </condition>
</extension>
```

How it works: Uses `allocated_time` from OCS to schedule automatic hangup and plays warning announcement 30 seconds before disconnection.

Answer & Hangup Notifications

After the CCR-Initial authorizes a call, OmniTAS still needs to know **when the call is answered** (to begin metering and send the CCR-Update) and **when it ends** (to send the CCR-Terminate). These events are sourced from FreeSWITCH over the Event Socket Layer (ESL).



How it works:

- OmniTAS holds a persistent inbound ESL connection to the local FreeSWITCH and subscribes to `CHANNEL_ANSWER` and `CHANNEL_HANGUP_COMPLETE`.
- On `CHANNEL_ANSWER`, it reads the SIP Call-ID and the real answer timestamp and sends the CCR-Update that starts metering.
- On `CHANNEL_HANGUP_COMPLETE`, it reads the SIP Call-ID and the FreeSWITCH hangup cause and sends the CCR-Terminate with the final used time.
- Events are matched to a credit-control session by SIP Call-ID. Events for call legs without a session (MT legs, bridged B-legs, exempt calls) are ignored — this is what keeps non-charged calls from producing CCRs.
- **Duplicate answer is idempotent.** Both legs of a bridged call fire `CHANNEL_ANSWER` and can resolve to the same Call-ID; only the first sends a CCR-Update and starts the periodic sender, the rest are ignored (otherwise you get duplicate CCR-Us and double periodic senders).
- **A transfer ends the charged session.** A `BLIND_TRANSFER`/`ATTENDED_TRANSFER` hangup is treated as a normal hangup → CCR-Terminate (the charged caller has left the call, or the call was transferred to the out-of-credit announcement). Once a session has

terminated, any later hangup for the same Call-ID finds no session and sends nothing.

The `CallEventListener` always runs (a real deployment always has the co-located FreeSWITCH; it is skipped only under `test_mode`). It is **not** gated by a separate switch — whether a CCR is sent is decided per event by `online_charging.enabled`. With charging **off** the listener still receives and logs `CHANNEL_ANSWER`/`CHANNEL_HANGUP_COMPLETE` at debug but sends no CCR. (Earlier releases had a separate `esl_notifications` toggle; it was removed because it had to agree with `online_charging.enabled` and silently broke charging — only CCR-Initial, no CCR-U/CCR-T — when charging was on but the toggle was absent.)

Deprecated: `/call_event` HTTP endpoint

Earlier releases detected answer/hangup via FreeSWITCH dialplan `curl` hooks that POSTed to a `/call_event` HTTP endpoint on OmniTAS. **This mechanism is deprecated.** The dialplan hooks have been removed and ESL is now the notification source.

The `/call_event` endpoint still exists and returns `200 OK` so that any lingering dialplan references do not error, but it performs **no charging action**. Operators upgrading from an older release should ensure their dialplans no longer rely on it.

Diameter Messages

CCR-Initial (Request Type 1)

Sent before call setup to request authorization and initial credit allocation.

Key AVPs Sent:

AVP	Code	Type	Description
Session-Id	263	UTF8String	Session identifier, **constant for the whole credit-control session** (CCR-I/U/T share it). Derived deterministically from the SIP Call-ID.
Auth-Application-Id	258	Unsigned32	Value `4` for Diameter Credit Control Application per RFC 4006
Service-Context-Id	461	UTF8String	`"000.000.12.32260@3gpp.org"` for IMS charging per TS 32.299
CC-Request-Type	416	Enumerated	Value `1` (INITIAL_REQUEST)
CC-Request-Number	415	Unsigned32	Sequence number — **`0` for INITIAL_REQUEST**, then `1`, `2`, ... for subsequent requests in the session per RFC 4006 §8.2
Service-Identifier	439	Unsigned32	Stable provisioned service identifier (`service_identifier` config)
Subscription-Id	443	Grouped	Subscriber MSISDN or IMSI
Requested-Service-Unit	437	Grouped	Reservation request — `CC-Time` = `requested_units_seconds`
Service-Information	873	Grouped	IMS-specific call details (calling/called party, Role-Of-Node, timestamps)

Example CCR-I:

```
Session-Id: "tas01.example.org;1463927445;1744753804"
Auth-Application-Id: 4
CC-Request-Type: 1 (INITIAL_REQUEST)
CC-Request-Number: 0
Subscription-Id:
  - Subscription-ID-Type: 0 (END_USER_E164)
    Subscription-ID-Data: "313380000000670"
Multiple-Services-Credit-Control:
  - Service-Identifier: 1
    Requested-Service-Unit:
      - CC-Time: 300 (requested reservation)
Service-Information:
  - IMS-Information:
      - Role-Of-Node: 0 (ORIGINATING_ROLE)
      - Node-Functionality: 6 (AS)
      - Calling-Party-Address: "tel:+313380000000670"
      - Called-Party-Address: "tel:+24724741234"
```

> The `Requested-Service-Unit` carries a real reservation (not zero). The OCS decides the actual grant and returns it in the CCA's `Granted-Service-Unit`.

CCA (Credit-Control-Answer)

Response from OCS with authorization decision and granted credit.

Key AVPs Received:

AVP	Code	Type	Description
Result-Code	268	Unsigned32	`2001` for success. See Result Codes for error values.
Granted-Service-Unit	431	Grouped	Allocated credit (time in seconds)
Service-Information	873	Grouped	Additional charging data (carrier info, charged party, etc.)

Example CCA with AVPs:

```
Session-Id: "tas01.example.org;1769294418268;8a078232"
Result-Code: 2001 (DIAMETER_SUCCESS)
CC-Request-Type: 1
CC-Request-Number: 1
Granted-Service-Unit:
  - CC-Time: 600 (10 minutes granted)
Service-Information:
  - IMS-Information:
      - Carrier-Select-Routing-Information: "1408"
  - Alternate-Charged-Party-Address: "NickTest"
```

Resulting Variables:

```
allocated_time = 600
CCA.Service-Information.Carrier-Select-Routing-Information =
"1408"
CCA.Service-Information.Alternate-Charged-Party-Address =
"NickTest"
```

CCR-Update (Request Type 2)

Sent during active calls for periodic re-authorization or interim usage reporting.

When Sent:

- **On call answer** (`CHANNEL_ANSWER` from ESL) — `CC-Request-Number` = `1`, with the real answer time in the `SIP-Response-Timestamp`
- **Periodically** during the active call, timed dynamically as `allocated_time - ccr_update_buffer_seconds` from each grant (falling back to `periodic_ccr_time_seconds`)

Key Differences from CCR-I:

- `CC-Request-Type`: `2` (UPDATE_REQUEST)
- `CC-Request-Number`: `1` on answer, then increments on each periodic re-auth
- `Requested-Service-Unit`: the next reservation (`requested_units_seconds`)

- Same constant `Session-Id` as the CCR-Initial

> Consumed time is reported in the CCR-Terminate (`Used-Service-Unit`), not in interim CCR-Updates. Interim updates extend the reservation; final usage is settled at termination.

CCR-Terminate (Request Type 3)

Sent on call hangup (`CHANNEL_HANGUP_COMPLETE` from ESL) with final usage reporting.

Key AVPs:

- `CC-Request-Type`: `3` (TERMINATION_REQUEST)
- `CC-Request-Number`: next in sequence (e.g. `2`)
- `Used-Service-Unit`: total consumed talk time (`CC-Time`, seconds since answer)
- `Termination-Cause`: `1` (DIAMETER_LOGOUT) per RFC 6733 §8.15
- `Cause-Code` (AVP 861, IMS-Information): `0` for a normally answered-and-released call, `2` for an unsuccessful session setup, per TS 32.299

Result Codes

Code	Name	Description	OmniTA
2001	DIAMETER_SUCCESS	Request approved	Parse AVPs
4010	DIAMETER_END_USER_SERVICE_DENIED	Service denied for subscriber	Reject call `CALL_REJE`
4012	DIAMETER_CREDIT_LIMIT_REACHED	Insufficient credit	Reject call `OUTGOING_`
5003	DIAMETER_AUTHORIZATION_REJECTED	OCS policy denied	Reject call `CALL_REJE`
5xxx	Permanent failures	OCS configuration or system error	Reject call

Reference: RFC 6733 §7.1 and 3GPP TS 32.299

Metrics

> See doc/metrics.md for the full metric catalogue. The metrics below are the ones relevant to online charging.

Diameter Request / Response Metrics

CCRs are split by request type via the `command` label so CCR-Initial (setup), CCR-Update (interim re-auth) and CCR-Terminate (teardown) can be tracked independently — a spike in failed interim updates is then visible on its own.

Metric: `diameter_requests_total` (Counter) — Diameter requests sent
Metric: `diameter_responses_total` (Counter) — Diameter responses received
Metric: `diameter_response_duration_milliseconds` (Histogram) — request round-trip time

Labels:

- `application` - `ro` (online charging) or `sh` (subscriber data)
- `command` - `ccr_i`, `ccr_u`, `ccr_t` (Ro) or `udr` (Sh)
- `result_code` - (responses only) Diameter Result-Code: `2001`, `4012`, etc.; `0` = timeout / unparseable reply
- `result` - (duration only) `success`, `nocredit`, `error`

Example queries:

```
# CCR rate by request type
sum by (command) (rate(diameter_requests_total{application="ro"}[5m]))

# CCR error rate, per request type (anything that is not 2001)
sum by (command) (rate(diameter_responses_total{application="ro",
result_code!="2001"}[5m]))

# Credit limit rejections (4012)
rate(diameter_responses_total{application="ro", result_code="4012"}
[5m])

# 95th percentile CCR latency by request type
histogram_quantile(0.95,
  sum by (le, command)
(rate(diameter_response_duration_milliseconds_bucket{application="ro"
[5m]))
)
```

Credit-Control Quota Metrics

Metric: `ro_charging_quota_seconds` **Type**: Histogram **Description**: Quota (in seconds) observed on each CCR, for verifying charging correctness **Labels**:

- `request_type` - `ccr_i`, `ccr_u`, `ccr_t`

- `kind` - `requested` (CC-Time in Requested-Service-Unit), `granted` (Granted-Service-Unit, `0` = no credit), `used` (Used-Service-Unit on terminate)

Example queries:

```
# Median granted quota at call setup
histogram_quantile(0.5,
rate(ro_charging_quota_seconds_bucket{request_type="ccr_i",
kind="granted"}[5m]))

# No-credit grants (zero-second allocations) per second
rate(ro_charging_quota_seconds_bucket{kind="granted", le="0"}[5m])

# Total used seconds reported on termination per second (sanity-
check against billed minutes)
rate(ro_charging_quota_seconds_sum{kind="used"}[5m])
```

OCS Authorization & Event Metrics

Metric: `ocs_authorization_attempts_total` (Counter) **Labels**: `result` (`success`, `nocredit`, `timeout`, `error`), `skipped` (`true` if bypassed via regex, else `false`)

Metric: `online_charging_events_total` (Counter) — lifecycle events **Labels**: `event_type` (`authorize`, `answer`, `reauth`, `hangup`, `credit_exhaustion_hangup`, `hangup_rescheduled`), `result` (`success`, `nocredit`, `timeout`, `error`, `triggered`)

Example queries:

```
# Authorization success rate (excluding skipped)
sum(rate(ocs_authorization_attempts_total{result="success",
skipped="false"}[5m]))
  / sum(rate(ocs_authorization_attempts_total{skipped="false"}[5m]))

# Calls released mid-call due to credit exhaustion
rate(online_charging_events_total{event_type="credit_exhaustion_hangu
[5m])
```

Troubleshooting

AVP Variables Not Available in FreeSWITCH

Symptoms:

- FreeSWITCH dialplan cannot access `${CCA.Service-Information.*}` variables
- Variables show as empty or undefined

Possible causes:

1. OCS not returning Service-Information AVPs in CCA
2. AVP parsing failed due to unexpected structure
3. Variables not exported to FreeSWITCH channel

Resolution:

1. **Verify OCS Response Contains AVPs**

   Check OmniTAS logs for CCA message:

   ```
   [debug] Credit Control Answer: {:diameter_packet, ...}
   [debug] Parsed AVP variables: %{
     "CCA.Service-Information.Carrier-Select-Routing-Information"
   => "1408",
     "CCA.Service-Information.Alternate-Charged-Party-Address" =>
   "NickTest"
   }
   ```

   If "Parsed AVP variables" is empty `%{}`, OCS is not returning the expected AVPs.

2. **Check for AVP Parsing Errors**

   Look for warnings in logs:

   ```
   [warning] got back another type of reply: {...}
   ```

This indicates AVP structure doesn't match expected format. Check Diameter packet structure.

3. **Verify FreeSWITCH Variable Export**

   In FreeSWITCH console or ESL:

   ```
   freeswitch> uuid_dump <call-uuid>
   ```

   Look for variables with the `variable_` prefix and `CCA.` in the name:

   ```
   variable_CCA.Service-Information.Carrier-Select-Routing-
   Information: 1408
   variable_CCA.Service-Information.Alternate-Charged-Party-
   Address: NickTest
   variable_CCA.Auth-Application-Id: 4
   variable_CCA.Result-Code: 2001
   ```

   Note: FreeSWITCH preserves dots and hyphens in variable names. They work correctly in dialplan:

   ```
   <action application="log" data="Carrier: ${CCA.Service-
   Information.Carrier-Select-Routing-Information}"/>
   ```

Call Rejected with "unhandled" Error

Symptoms:

- Logs show: `[warning] Could not authorize call: :unhandled`
- Valid CCA responses (Result-Code 2001) are rejected
- Calls fail despite OCS approving them

Possible causes:

- CCA message structure doesn't match expected pattern
- Vendor-specific AVPs in unexpected positions
- AVP position index mismatch

Resolution:

This was a known issue fixed in recent releases. Ensure you're running current version.

Previous behavior: Pattern matching required:

- Granted-Service-Unit AVP at position 7 exactly
- Empty vendor-specific AVP list `[]`

Current behavior: Pattern matching accepts:

- Granted-Service-Unit AVP at any position
- Non-empty vendor-specific AVP lists

If issue persists:

1. Capture CCA packet structure from logs
2. Check if AVPs are in expected Diameter format
3. Verify Result-Code is 2001

OCS Timeout on All Requests

Symptoms:

- All CCR requests timeout
- Logs show: `[debug] Got back response for authorize: {:error, :timeout}`
- No CCA received within 5 seconds

Possible causes:

- Network connectivity to OCS/DRA
- Firewall blocking Diameter port (3868)
- Incorrect `destination_realm` or `destination_host`
- OCS not responding to requests

Resolution:

1. **Verify Network Connectivity**

   Test TCP connection to OCS:

   ```
   telnet ocs.example.com 3868
   ```

   Should connect successfully. If connection refused or timeout, check firewall rules.

2. **Check Diameter Configuration**

   Verify `destination_realm` matches OCS configuration:

   ```
   config :tas, :diameter,
     destination_realm: "epc.mnc001.mcc001.3gppnetwork.org"  #
   Must match OCS realm
   ```

3. **Review OCS Logs**

   Check OCS for incoming CCR messages. If OCS receives requests but doesn't respond:

   - Verify OmniTAS `origin_host` is recognized by OCS
   - Check OCS peer configuration allows connections from OmniTAS
   - Verify Service-Context-Id and Application-Id match OCS expectations

Credit Exhaustion Not Hanging Up Calls

Symptoms:

- Calls continue beyond granted credit time
- No automatic hangup when `allocated_time` expires
- `schedule_hangup_auth` enabled but not working

Possible causes:

- FreeSWITCH scheduled hangup not configured

- `schedule_hangup_auth` is `false`
- Call state not tracked properly

Resolution:

1. **Verify Configuration**

   Ensure `schedule_hangup_auth` is enabled:

   ```
   config :tas, :online_charging,
     schedule_hangup_auth: true
   ```

2. **Check FreeSWITCH ESL Connection**

   Verify OmniTAS can send commands to FreeSWITCH:

   ```
   [debug] Schedule Hangup Response: {:ok, "+OK"}
   ```

   If error or no response, check FreeSWITCH Event Socket configuration.

3. **Monitor Call State**

   Check that call UUID is tracked in call state:

   ```
   [debug] Setting Scheduled Hangup for call in 600 seconds
   ```

   If UUID not found, call state tracking may have issues.

Skipped Regex Not Bypassing OCS

Symptoms:

- Emergency calls (911, 000) still go through OCS authorization
- Numbers matching `skipped_regex` patterns are not bypassed
- Delays on emergency calls

Possible causes:

- Regex pattern syntax error
- Destination number format mismatch
- Regex not properly escaped

Resolution:

1. **Verify Regex Patterns**

   Test regex compilation:

   ```
   Regex.compile("^911$")  # Should return {:ok, ~r/^911$/}
   ```

   Common mistakes:

   - Missing anchors: Use `^911$` not `911`
   - Escaping: Use `\*` for literal asterisk, not `\*`

2. **Check Number Format**

   Verify destination number format matches pattern:

   ```
   [debug] Checking if dialled number "911" matches skipped
   regex...
   ```

   If number is formatted as "+1911" but pattern is "^911$", it won't match.

3. **Example Patterns**

   ```
   config :tas, :online_charging,
     skipped_regex: [
       "^911$",           # US Emergency
       "^000$",           # AU Emergency
       "^112$",           # International Emergency
       "^\*86$",         # Voicemail (escaped asterisk)
       "^1?800\d{7}$"    # Toll-free numbers
     ]
   ```

Reference

3GPP Specifications

Specification	Title	Relevant Sections
TS 32.299	Diameter charging applications	§6.3 (Ro interface), §7.2 (AVP definitions)
TS 32.240	Charging architecture and principles	§5 (Online charging)
TS 29.229	Cx and Dx interfaces	Service-Information AVP usage in IMS

IETF RFCs

RFC	Title	Relevant Sections
RFC 6733	Diameter Base Protocol	§3 (Protocol overview), §7 (Error handling)
RFC 4006	Diameter Credit-Control Application	§8 (Credit-Control messages)

AVP Codes Reference

Vendor-ID `0` = IETF base (RFC 6733 / RFC 4006); Vendor-ID `10415` = 3GPP (TS 32.299).

Complete CCR structure (voice call)

Every AVP OmniTAS includes in a call Credit-Control-Request, with grouping shown:

```
Credit-Control-Request (Command 272, App 4)
├─ Session-Id (263)
├─ Origin-Host (264)
├─ Origin-Realm (296)
├─ Destination-Realm (283)
├─ Destination-Host (293)                      [only if
configured]
├─ Auth-Application-Id (258)           = 4
├─ Service-Context-Id (461)            =
000.000.12.32260@3gpp.org
├─ CC-Request-Type (416)
├─ CC-Request-Number (415)
├─ Event-Timestamp (55)
├─ User-Name (1)                               [only if username
set]
├─ Termination-Cause (295)                     [CCR-T only]
├─ Subscription-Id (443)
│   ├─ Subscription-Id-Type (450)      = 0 (END_USER_E164)
│   └─ Subscription-Id-Data (444)      = subscriber MSISDN
├─ Multiple-Services-Credit-Control (456)
│   ├─ Service-Identifier (439)
│   ├─ Requested-Service-Unit (437)            [CCR-I, CCR-U]
│   │   └─ CC-Time (420)
│   └─ Used-Service-Unit (446)                 [CCR-T]
│       └─ CC-Time (420)
└─ Service-Information (873, v10415)
    ├─ IN-Information                           [operator
extension]
    │   └─ Real-Called-Number
    └─ IMS-Information (876, v10415)
        ├─ Role-Of-Node (829)          = 0 (ORIGINATING_ROLE)
        ├─ Node-Functionality (862)    = 6 (AS)
        ├─ User-Session-Id (830)       = SIP Call-ID
        ├─ Calling-Party-Address (831)
        ├─ Called-Party-Address (832)
        ├─ Requested-Party-Address (1251)
        ├─ Time-Stamps (833)
        │   ├─ SIP-Request-Timestamp (834)
        │   ├─ SIP-Request-Timestamp-Fraction (2301)
        │   ├─ SIP-Response-Timestamp (835)              [once
answered]
        │   └─ SIP-Response-Timestamp-Fraction (2302)    [once
```

```
answered]
        └─ Cause-Code (861)                          [CCR-T]
```

AVPs sent by OmniTAS (CCR)

The **In** column shows which request types carry the AVP: **I** = CCR-Initial, **U** = CCR-Update, **T** = CCR-Terminate.

AVP	Code	Vendor	Type	In	Value / Sc
Session-Id	263	0	UTF8String	I U T	Constant per call from SIP Call-ID); across I/U/T
Origin-Host	264	0	DiameterIdentity	I U T	OmniTAS Diamet (`origin_host`.`or`
Origin-Realm	296	0	DiameterIdentity	I U T	`origin_realm`
Destination-Realm	283	0	DiameterIdentity	I U T	OCS realm
Destination-Host	293	0	DiameterIdentity	I U T	Only present if a host is configured
Auth-Application-Id	258	0	Unsigned32	I U T	`4` (Diameter Cred Application)
Service-Context-Id	461	0	UTF8String	I U T	`000.000.12.3226`
CC-Request-Type	416	0	Enumerated	I U T	`1`=Initial, `2`=Upd `3`=Terminate
CC-Request-Number	415	0	Unsigned32	I U T	`0` for Initial, then

AVP	Code	Vendor	Type	In	Value / Sc
Event-Timestamp	55	0	Time	I U T	Time the request generated
User-Name	1	0	UTF8String	I U T	Optional; only if a username/IMSI is
Termination-Cause	295	0	Enumerated	T	`1` (DIAMETER_LO
Subscription-Id	443	0	Grouped	I U T	Subscriber identit children)
→ Subscription-Id-Type	450	0	Enumerated	I U T	`0` (END_USER_E1
→ Subscription-Id-Data	444	0	UTF8String	I U T	Subscriber MSISD
Multiple-Services-Credit-Control	456	0	Grouped	I U T	Credit-control cor children)
→ Service-Identifier	439	0	Unsigned32	I U T	`service_identif` provisioned)
→ Requested-Service-Unit	437	0	Grouped	I U	Reservation requ

AVP	Code	Vendor	Type	In	Value / Sc
→ → CC-Time	420	0	Unsigned32	I U	`requested_units`
→ Used-Service-Unit	446	0	Grouped	T	Final consumptio
→ → CC-Time	420	0	Unsigned32	T	Consumed secon answer
Service-Information	873	10415	Grouped	I U T	3GPP service con
→ IN-Information	—	10415	Grouped	I U T	Operator extensi the real dialled nu
→ → Real-Called-Number	—	10415	UTF8String	I U T	Dialled (called) nu
→ IMS-Information	876	10415	Grouped	I U T	IMS charging deta
→ → Role-Of-Node	829	10415	Enumerated	I U T	`0` (ORIGINATING_ MO is the charge
→ → Node-Functionality	862	10415	Enumerated	I U T	`6` (AS)
→ → User-Session-Id	830	10415	UTF8String	I U	SIP Call-ID (OCS c key)

AVP	Code	Vendor	Type	In	Value / Sc
				T	
→ → Calling-Party-Address	831	10415	UTF8String	I U T	`tel:+<calling M`
→ → Called-Party-Address	832	10415	UTF8String	I U T	`tel:+<called nu`
→ → Requested-Party-Address	1251	10415	UTF8String	I U T	`tel:+<called nu`
→ → Time-Stamps	833	10415	Grouped	I U T	SIP request/respo
→ → → SIP-Request-Timestamp	834	10415	Time	I U T	INVITE time (who
→ → → SIP-Request-Timestamp-Fraction	2301	10415	Unsigned32	I U T	INVITE time (milli
→ → → SIP-Response-Timestamp	835	10415	Time	U T	Answer (200 OK) **only once answ**

AVP	Code	Vendor	Type	In	Value / Sc
→ → → SIP-Response-Timestamp-Fraction	2302	10415	Unsigned32	U T	Answer time (mil only once answer
→ → Cause-Code	861	10415	Integer32	T	`0` normal release unsuccessful setu

> Codes shown as `–` are operator/vendor extensions carried under `Service-Information`; they have no 3GPP-assigned code and are defined in the OmniTAS Diameter dictionary.

AVPs received from the OCS (CCA)

AVP	Code	Vendor	Type	Description
Result-Code	268	0	Unsigned32	`2001` = success; see Result Codes
Granted-Service-Unit	431	0	Grouped	Allocated credit
→ CC-Time	420	0	Unsigned32	Granted seconds (`allocated_time`). `0` ⇒ no credit ⇒ bar/hangup
Service-Information	873	10415	Grouped	Optional charging data returned by the OCS
→ Carrier-Select-Routing-Information	2023	10415	UTF8String	Carrier routing code (mapped to a FreeSWITCH variable)
→ Alternate-Charged-Party-Address	1280	10415	UTF8String	Billing party identifier (mapped to a FreeSWITCH variable)

All `Service-Information` sub-AVPs returned in the CCA are flattened into FreeSWITCH channel variables — see AVP Parsing and Variable Mapping and FreeSWITCH Channel Variables.

FreeSWITCH Channel Variables

All extracted AVP data is available as FreeSWITCH channel variables:

Variable Name	Source	Example Value
`${allocated_time}`	Granted-Service-Unit / CC-Time	`600`
`${CCA.Session-Id}`	Session-Id AVP	`omni-as01.epc...;1769299669873;`
`${CCA.Result-Code}`	Result-Code AVP	`2001`
`${CCA.Auth-Application-Id}`	Auth-Application-Id AVP	`4`
`${CCA.CC-Request-Type}`	CC-Request-Type AVP	`1`
`${CCA.CC-Request-Number}`	CC-Request-Number AVP	`1`
`${CCA.CC-Time}`	CC-Time AVP (if present)	`600`
`${CCA.Origin-Host}`	Origin-Host AVP	`ocs01.epc.mnc001.mcc001.3gppnet`
`${CCA.Origin-Realm}`	Origin-Realm AVP	`epc.mnc001.mcc001.3gppnetwork.o`

Variable Name	Source	Example Value
`${CCA.Service-Information.Carrier-Select-Routing-Information}`	Service-Information → Carrier-Select-Routing-Information	`1408`
`${CCA.Service-Information.Alternate-Charged-Party-Address}`	Service-Information → Alternate-Charged-Party-Address	`NickTest`

Variable Format:

- All CCA AVPs use the prefix `CCA.`
- Nested AVPs use dot notation: `CCA.Parent.Child`
- Dots and hyphens are preserved in variable names
- In uuid_dump, variables appear with `variable_` prefix

Example uuid_dump output:

```
variable_allocated_time: 600
variable_CCA.Service-Information.Carrier-Select-Routing-
Information: 1408
variable_CCA.Service-Information.Alternate-Charged-Party-Address:
NickTest
variable_CCA.Result-Code: 2001
```

Operations Guide

 Back to Main Documentation

This document covers operational monitoring and management features available in the Control Panel.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - System configuration reference
- **Metrics Reference** - Prometheus metrics and monitoring

Monitoring & Testing Tools

- **HLR & Call Simulator** - Testing tools for HLR and call simulation
- **IMS Conference Server** - Conference management and monitoring
- **Dialplan Metrics** - Dialplan-specific metrics

Call Processing & Services

- **Dialplan Configuration** - Call routing and dialplan reference
- **Sh Interface** - Subscriber data testing
- **Online Charging** - OCS testing
- **Number Translation** - Number translation testing
- **Voicemail** - Voicemail management

Integration Interfaces

- **SS7 MAP** - HLR/MAP testing
- ⚙ **Supplementary Services** - Emergency calling, call forwarding

- **HOMER Integration** - SIP trace and log correlation

Operations

This section covers operational monitoring and management features available in the OmniTAS Control Panel.

Table of Contents

- Subscribers View
- Call Detail Records (CDR)
- Active Calls Monitoring
- IMS Conference Server
- Gateway Status
- Diameter Peer Status
- Logs Viewer
- Cell Tower Database
- Call Simulator
- HLR/MAP Testing
- Other Views

Subscribers View

The Subscribers view provides real-time monitoring of IMS subscriber registrations stored in the Sofia SIP registration database.

Access: Navigate to `/subscribers` in the Control Panel

Features

- **Registration List:** View all active subscriber registrations
- **Registration Details:** Click on any registration to view complete details including:
  - SIP User and Realm

- Contact URI
- Registration status and expiration
- Network information (IP, port, hostname)
- Authentication details
- Cell tower location (when available via P-Access-Network-Info)
  - MCC/MNC, Radio Type, TAC/LAC, Cell ID
  - Geographic coordinates and coverage range
  - Interactive map view powered by OpenStreetMap and OpenCellID data

Data Source

Registration data is queried directly from the Sofia registration database, providing real-time visibility into subscriber attachment status. Cell tower locations are resolved using the OpenCellID database when subscribers provide P-Access-Network-Info headers in their SIP REGISTER messages.

Use Cases

- Monitor active subscriber registrations
- Verify subscriber attachment status
- Troubleshoot registration issues
- Audit subscriber connectivity

Call Detail Records (CDR)

The CDR view provides access to call detail records stored by TAS for billing, troubleshooting, and analytics purposes.

Access: Navigate to `/cdr` in the Control Panel

Features

- **Paginated View:** Browse through call records (100 per page with Previous/Next controls)
- **Advanced Search:** Powerful search with support for exact match, inverse/exclude, and multiple terms
- **Column Selection:** Customize which fields to display
  - Click **"Columns"** button to open column picker modal
  - Select/deselect individual columns
  - **Select All** / **Deselect All** quick actions
  - Selection persists across sessions (saved to browser localStorage)
  - Shows "X / Y columns" counter
- **Sortable Columns:** Click any column header to sort (ascending/descending)

    - Visual indicators (▲ ascending, ▼ descending)
    - Sorted column highlighted in blue
    - Resets to page 1 when sorting changes
- **Multiple Filter Options:**
    - **Text Search:** Search across all fields with advanced operators
    - **Date Range Filter:** Filter by start/end date and time (datetime picker)
    - **Field-Specific Filter:** Filter by exact field value (hangup cause, caller ID, destination, context)
    - **Active Filter Display:** Visual chips show currently active filters
    - **Clear All:** One-click removal of all active filters
- **Detailed Information:** Click on any CDR row to expand and view all fields:
    - Call parties (caller ID name/number, destination number)
    - Timestamps (start, answer, end)
    - Duration and billed seconds
    - Hangup cause (color-coded: green=normal, yellow=cancel, red=error)
    - Call UUIDs (A-leg and B-leg)
    - Context and account code
    - All available database fields in alphabetical order
- **Color-Coded Hangup Causes:**
    - □ Green: `NORMAL_CLEARING`
    - □ Yellow: Cancelled calls
    - □ Red: Error conditions
- **Total Count:** Real-time display of total matching records
- **Responsive Layout:** Filters wrap appropriately on smaller screens

How to Use

1. **Basic Viewing:**
    - Page loads with latest 100 CDR records (sorted by `start_stamp` descending)
    - Total record count shown in top-right
    - Use **Previous** / **Next** buttons to navigate pages
    - Click any row to expand and see all fields

2. **Customize Columns:**

    - Click **"Columns"** button in top-right
    - Modal shows all available fields
    - Check/uncheck fields to show/hide columns
    - Use **"Select All"** or **"Deselect All"** for quick selection
    - Settings automatically saved to browser
    - Close modal to apply changes

3. **Sort Data:**

    - Click any column header to sort by that field
    - First click: Descending (▼)
    - Second click: Ascending (▲)
    - Third click: Back to descending
    - Sorted column highlighted in blue

4. **Search Records:**

    - Enter search query in **"Search"** box
    - Supports advanced operators (see Search Syntax below)
    - Searches across multiple fields: `caller_id_number`, `destination_number`, `uuid`, `caller_id_name`, `hangup_cause`
    - Click **"Apply"** to execute search

5. **Filter by Date Range:**

    - Use **"Start Date"** and **"End Date"** datetime pickers
    - Both dates required for date filtering
    - Supports date and time selection
    - Click **"Apply"** to filter

6. **Filter by Specific Field:**

    - Select field from **"Select Field to Filter"** dropdown:
        - Hangup Cause
        - Caller ID

        - Destination
        - Context
    - Enter exact value in **"Enter Filter Value"**
    - Click **"Apply"** to filter

7. **Combine Filters:**

    - All filters can be used simultaneously:
        - Text search + Date range + Field filter all work together
    - Active filters shown as chips below the filter form
    - Click **"Clear All"** to remove all filters at once

8. **View Details:**

    - Click any CDR row to expand
    - Shows all database fields in a grid layout
    - Fields displayed in alphabetical order
    - Hangup cause color-coded for quick identification
    - Click row again to collapse

Advanced Search Syntax

The search box supports powerful query syntax for precise record filtering across multiple fields simultaneously.

How Search Works:

The search engine checks **all searchable fields** in each CDR record. A record is included in results when it matches your search criteria in **any** of these fields:

- `caller_id_number`
- `destination_number`
- `uuid`
- `caller_id_name`
- `hangup_cause`

Search Operators (can be combined):

1. **Contains Search (default):**
    - Syntax: `term` (no quotes)
    - Matches: Records where **any field contains** the term anywhere within it
    - SQL: Uses `LIKE '%term%'` across all searchable fields joined with `OR`
    - Example: `61480` matches "61480123456", "55561480999", etc.
2. **Exact Match:**
    - Syntax: `"term"` (with double quotes)
    - Matches: Records where **any field exactly equals** the term
    - SQL: Uses `= 'term'` across all searchable fields joined with `OR`
    - Example: `"911"` matches only exactly "911", not "9115" or "1911"
3. **Inverse/Exclude:**
    - Syntax: `!term` (exclamation mark prefix, no quotes)
    - Matches: Records where **no field contains** the term
    - SQL: Uses `NOT LIKE '%term%'` across all searchable fields joined with `AND`
    - Example: `!NORMAL` excludes any record with "NORMAL" in any field
4. **Exact Inverse/Exclude:**
    - Syntax: `!"term"` (exclamation mark + double quotes)
    - Matches: Records where **no field exactly equals** the term
    - SQL: Uses `!= 'term'` across all searchable fields joined with `AND`
    - Example: `!"NORMAL_CLEARING"` excludes records where any field is exactly "NORMAL_CLEARING"
5. **Multiple Terms with AND:**
    - Syntax: `term1 AND term2` (case-insensitive AND)
    - Matches: Records matching **all terms** (each term can match different fields)
    - Each term is processed with its own operator (quotes, !, etc.)
    - Terms are combined with `AND` in SQL

- Example: `"911" AND "12345"` finds records with "911" in one field AND "12345" in another

Search Execution Logic:

```
For each CDR record:
  For normal search (no !):
    - Check if ANY field contains/equals the term → Include if
TRUE
    - SQL: field1 LIKE '%term%' OR field2 LIKE '%term%' OR ...

  For inverse search (!):
    - Check if ALL fields do NOT contain/equal the term → Include
if TRUE
    - SQL: field1 NOT LIKE '%term%' AND field2 NOT LIKE '%term%'
AND ...

  For AND searches:
    - Each term is evaluated separately
    - All term conditions must be TRUE → Include if TRUE
    - SQL: (term1_conditions) AND (term2_conditions) AND ...
```

Complex Search Examples:

Query	How It Works	Result
`61480`	Contains search across all fields	All records with "61480" anywhere (caller, destination, UUID, etc.)
`"911"`	Exact match across all fields	Records where any field is exactly "911"
`!NORMAL_CLEARING`	Inverse contains search	Excludes records with "NORMAL_CLEARING" in ANY field (failed calls)
`!"NORMAL_CLEARING"`	Exact inverse	Excludes records where any field exactly equals "NORMAL_CLEARING"
`"911" AND "12345"`	Exact "911" AND exact "12345"	Records with both values (e.g., caller="12345", destination="911")
`!NORMAL AND 61480`	Inverse contains "NORMAL" AND contains "61480"	Non-normal calls involving "61480"
`!"ANSWER" AND !NORMAL`	Exact inverse "ANSWER" AND inverse contains "NORMAL"	Exclude answered calls and anything with "NORMAL"
`61480 AND !NORMAL_CLEARING`	Contains "61480" AND inverse contains "NORMAL_CLEARING"	Failed calls involving "61480"

Practical Use Cases:

- **Find specific number:** `61480123456` - Contains search finds partial matches
- **Find exact emergency calls:** `"911"` - Only calls to exactly "911"
- **All failed calls:** `!NORMAL_CLEARING` - Exclude successful calls
- **Specific caller's failed calls:** `"61480123456" AND !NORMAL` - Combine exact caller with inverse
- **Exclude test numbers:** `!test AND !demo` - Multiple inverse searches
- **Complex debugging:** `61480 AND !"ANSWER" AND !CANCEL` - Contains one term, exclude exact and partial others

Data Source

CDR data is queried directly from the TAS CDR SQLite database.

The schema may vary between deployments based on specific requirements.

CDR Export Options

Important: CDR records can be exported in various formats to support integration with billing systems, analytics platforms, and reporting tools.

The CDR database schema and export formats are deployment-specific. When setting up your system, **please request the specific CDR output formats you need from your integration engineer**. Common export formats include:

- CSV (Comma-Separated Values)
- JSON (for API integration)
- XML
- Direct database access
- Custom formatted exports

Your integration engineer can configure CDR export mechanisms tailored to your operational and billing requirements.

Use Cases

- **Call Troubleshooting:** Search for specific calls by number or UUID to debug issues
- **Billing Reconciliation:** Filter by date range to match billing periods
- **Quality Analysis:** Filter by hangup cause to identify problem patterns
- **Emergency Call Auditing:** Search for "911" to verify emergency call handling
- **Customer Support:** Look up specific customer calls by caller ID or destination
- **Pattern Analysis:** Sort by duration or timestamps to identify anomalies
- **Compliance & Record Keeping:** Date range filters for regulatory reporting
- **Failed Call Analysis:** Use `!NORMAL_CLEARING` to find all failed calls
- **Context-Based Reports:** Filter by context to analyze specific call flows

Configuration

Default Visible Columns

You can configure which CDR fields are **shown by default** in the LiveView by setting `cdrs_field_list` in your `config/runtime.exs`:

```
config :tas,
  cdrs_field_list: [
    "caller_id_number",
    "destination_number",
    "start_stamp",
    "duration",
    "hangup_cause"
  ]
```

Behavior:

- If `cdrs_field_list` is **not set**: All available CDR fields are shown by default
- If `cdrs_field_list` **is set**: Only the specified fields are shown by default, but **all other fields remain available** in the column picker

- If a field in the list doesn't exist in the CDR data, it will be automatically skipped
- Field names can be specified as strings or atoms
- Users can manually select additional columns from the column picker at any time

Use Cases:

- Set a clean default view with only essential fields visible
- Reduce information overload for new users
- Standardize the initial column layout across all users
- Keep advanced fields hidden by default but still accessible

Example Configuration:

```
# Show only essential call information by default
cdrs_field_list: [
  "start_stamp",
  "caller_id_number",
  "destination_number",
  "duration",
  "billsec",
  "hangup_cause"
]
```

Note: This configuration sets the *default visible* columns. All CDR fields remain available in the "Columns" picker - users can manually show/hide any field they need.

Troubleshooting

No Results Found

1. Check for typos in search terms
2. Try removing quotes for broader search
3. Verify the term exists in searchable fields
4. Check date range isn't too restrictive

Too Many Results

1. Add more AND terms to narrow
2. Use exact match with quotes
3. Apply date range filters
4. Use field-specific filters

Unexpected Results

1. Remember search applies to ALL searchable fields
2. Check if term appears in unexpected field (like UUID)
3. Use exact match to avoid partial matches
4. Verify inverse logic (AND vs OR)

Tips

- **Column Selection:** Hide unused columns to focus on relevant data and improve performance
- **Combine Filters:** Use search + date range + field filter together for precise queries
- **Date Range Performance:** Narrow date ranges return results faster for large databases
- **Sort for Analysis:** Sort by duration to find long/short calls, or by timestamp to see call patterns
- **Active Filter Chips:** Use visual chips to verify which filters are currently active
- **Persistent Settings:** Column selections are saved per browser, useful for different analysis tasks
- **Color Coding:** Quickly scan hangup causes - green is good, red needs investigation
- **Expandable Details:** Click rows to see all fields without cluttering the main view
- **Search Operators:** Master the search syntax for powerful filtering:
  - Use quotes for exact matches: `"911"`
  - Use `!` to exclude: `!NORMAL_CLEARING`
  - Combine with AND: `"61480" AND !NORMAL`
- **Pagination:** Remember filters persist across pages - use pagination to review large result sets

Active Calls Monitoring

The Active Calls view shows real-time information about ongoing calls through the system.

Access: Navigate to `/calls` in the Control Panel

Features

- **Real-time Status:** Live view of active call sessions
- **Call Details:** View channel variables and call state information
- **UUID Tracking:** Monitor both A-leg and B-leg call identifiers

---

IMS Conference Server

The IMS Conference Server provides multi-party conferencing capabilities compliant with 3GPP IMS standards (RFC 4579, RFC 4575, TS 24.147).

Access: Navigate to `/conference` in the Control Panel

Documentation: See IMS Conference Server User Guide for detailed documentation

Features

- **Real-time Monitoring:** Live view of active conferences and participants
- **Conference Statistics Dashboard:**
  - Active conference count
  - Total participants across all conferences
  - Video conference count
  - Locked conference count
  - Server configuration details (domain, MNC/MCC, max participants)
- **Conference List:** View all active conferences with:
  - Conference ID and SIP URI
  - Current participant count

    - Conference creator identity
- **Conference Details:** Click any conference to expand and view:
    - Full conference information (state, video status, locked status, recording status)
    - Complete participant list with roles and states
    - Participant video status
- **Conference Control Actions:**
    - Lock/Unlock conferences to control access
    - Enable/Disable video for conferences
    - Real-time status updates with action feedback
- **Auto-Refresh:** Configurable auto-refresh (default: 5 seconds) for real-time monitoring

OmniTAS Console Management

All conference operations are also available through the OmniTAS console using the `ims_conference` command:

```
ims_conference list                    # List all active
conferences
ims_conference info <conf_id>          # Show conference details
ims_conference stats                   # Show server statistics
ims_conference lock <conf_id>          # Lock a conference
ims_conference unlock <conf_id>        # Unlock a conference
ims_conference video <conf_id> on|off  # Control video
ims_conference record <conf_id> start|stop  # Control recording
ims_conference add <conf_id> <sip_uri>     # Add participant
ims_conference remove <conf_id> <uuid>     # Remove participant
ims_conference destroy <conf_id>       # Terminate conference
```

Use Cases

- **Operational Monitoring:** Real-time visibility into active conferences and resource usage
- **Capacity Management:** Monitor participant counts and video usage to manage bandwidth
- **Troubleshooting:** Diagnose conference access issues, participant connection problems

- **Conference Control:** Lock conferences for privacy, manage video to control bandwidth
- **Compliance:** Monitor and record conferences for regulatory compliance

3GPP Compliance

The conference server implements key 3GPP IMS conferencing specifications:

- **TS 24.147**: Conferencing using IM Core Network subsystem
- **RFC 4579**: SIP Call Control - Conferencing for User Agents
- **RFC 4575**: SIP Event Package for Conference State
- **RFC 5239**: Framework for Centralized Conferencing

Gateway Status

Monitor the status and health of SIP gateways/trunks connected to the TAS.

Access: Navigate to `/gw` in the Control Panel

Features

- **Registration Status:** View gateway registration state
- **Call Statistics:** Track incoming/outgoing calls and failures
- **Ping Monitoring:** SIP OPTIONS ping times and reachability
- **Gateway Details:** Complete configuration and status information

Monitored Metrics

- SIP Registration status
- Ping time (average SIP OPTIONS response time)
- Uptime (seconds since profile restart)
- Calls In / Calls Out
- Failed Calls In / Failed Calls Out
- Last ping time and frequency

Diameter Peer Status

Monitor Diameter peer connectivity for Sh and Ro interfaces.

Access: Navigate to `/diameter` in the Control Panel

Features

- **Peer Status:** Connection state for each configured peer
- **Application Support:** View supported Diameter applications (Sh, Ro)
- **Watchdog Status:** Diameter watchdog monitoring

System Logs Viewer

Real-time unified log viewer for both TAS Backend (Elixir) and TAS Call Processing (FreeSWITCH) logs.

Access: Navigate to `/logs` in the Control Panel

Features

- **Unified Log Stream:** View logs from both TAS Backend and Call Processing in one interface
- **Real-time Updates:** Live streaming of log messages as they occur (auto-refresh every 1 second)
- **Color-Coded Log Levels:**
  - □ **Console** - Console messages (purple/magenta)
  - □ **Alert/Critical** - Urgent issues requiring immediate attention (red)

    - **Error** - Error conditions (light red)
    - **Warning** - Warning messages (yellow)
    - **Notice** - Notable informational messages (cyan)
    - **Info** - General informational messages (blue)
    - **Debug** - Debug/verbose logging (gray)
- **Source Badges:**
    - **TAS Backend** - Elixir application logs (blue badge)
    - **TAS Call Processing** - FreeSWITCH logs (purple badge)
- **Left Border Indicators:** Color-coded left border matching log level for quick visual scanning
- **Multiple Filters:**
    - **Source Filter:** All Sources / TAS Backend / TAS Call Processing
    - **Level Filter:** All / Console / Alert / Critical / Error / Warning / Notice / Info / Debug
    - **Text Search:** Real-time keyword search across log messages
- **Pause/Resume:** Freeze log streaming to analyze specific entries without losing context
- **Clear Logs:** Remove all current log entries from display
- **Log Counter:** Shows filtered logs vs total logs (e.g., "Showing 150 of 500 logs")
- **Tail Behavior:** Maintains last 500 log entries for performance
- **Metadata Display:** File name and line number for source code references (when available)
- **Scrollable View:** Fixed-height container with auto-scroll for latest logs

How to Use

1. **Basic Viewing:**
    - Page loads with latest 500 log entries from both sources
    - Logs appear in real-time as they're generated
    - Most recent logs appear at the top
    - Auto-refreshes every 1 second
2. **Filter by Source:**

- Select from **"Source"** dropdown:
  - **All Sources** - Show both TAS Backend and Call Processing logs
  - **TAS Backend** - Only Elixir application logs
  - **TAS Call Processing** - Only FreeSWITCH/dialplan logs
- Filter applies immediately

3. **Filter by Log Level:**

   - Select from **"Level"** dropdown:
     - **All** - Show all log levels
     - **Console** through **Debug** - Show only that specific level
   - Useful for focusing on errors or debugging specific issues

4. **Search for Keywords:**

   - Type in the **"Search logs..."** box
   - Case-insensitive search across log messages
   - Filters in real-time as you type
   - Combines with source and level filters

5. **Pause/Resume Stream:**

   - Click **"Pause"** button (orange) to freeze log updates
   - "PAUSED" indicator appears in header
   - Review specific log entries without new logs interrupting
   - Click **"Resume"** button (green) to restart live streaming

6. **Clear Logs:**

   - Click **"Clear"** button (red) to remove all displayed logs
   - Clears both TAS Backend and Call Processing logs
   - Fresh logs will appear as they're generated

7. **Read Log Entries:**

   - **Timestamp:** Shows time in HH:MM:SS.milliseconds format
   - **Source Badge:** Indicates TAS Backend (blue) or Call Processing (purple)

- **Log Level:** Color-coded level in brackets [ERROR], [INFO], etc.
- **File/Line:** Source code location (when available)
- **Message:** The actual log message content

Log Levels Explained

Level	Color	When Used	Example
Console	Purple	Console-specific messages	High-priority FreeSWITCH console output
Alert	Red	Immediate action required	System component failure
Critical	Red	Critical conditions	Database connection lost
Error	Light Red	Error conditions	Failed to process call, invalid configuration
Warning	Yellow	Warning conditions	Deprecated function used, retry attempt
Notice	Cyan	Notable normal events	Configuration reloaded, service started
Info	Blue	Informational messages	Call connected, Diameter request sent
Debug	Gray	Debug-level messages	Function entry/exit, variable values

Use Cases

- **Real-time Troubleshooting:** Monitor logs during active call to debug issues
- **Error Investigation:** Filter by Error/Critical levels to find problems
- **Call Flow Analysis:** Search for Call-ID or phone number to trace call path

- **Performance Monitoring:** Watch for warnings and errors during load testing
- **Integration Debugging:** Filter TAS Backend to see Diameter/Sh/Ro messages
- **Dialplan Debugging:** Filter TAS Call Processing to see FreeSWITCH call routing
- **System Health Monitoring:** Keep logs open to watch for anomalies
- **Development & Testing:** Use Debug level to see verbose application behavior

Tips

- **Combine Filters:** Use Source + Level + Search together for precise filtering
  - Example: Source="TAS Backend" + Level="Error" + Search="Diameter" → Find Diameter errors
- **Pause Before Searching:** Pause stream before typing search query to avoid logs scrolling
- **Use Debug Wisely:** Debug level is verbose - filter to specific source to reduce noise
- **Color Scanning:** Quickly scan left borders - red borders indicate problems
- **Source Badges:** Blue badges (Backend) for app logic, Purple badges (Call Processing) for calls
- **Timestamp Precision:** Millisecond timestamps help correlate events across systems
- **File References:** Click/note file:line references to jump to source code
- **Clear Regularly:** Clear logs when switching investigation contexts for clarity
- **Search for UUIDs:** Search for Call-ID/UUID to follow a specific call through entire system
- **Emergency Search:** Search "911" or "emergency" to quickly find emergency call handling

Technical Details

- **Log Limit:** Maximum 500 logs displayed (oldest discarded when limit reached)
- **Refresh Rate:** Auto-refresh every 1000ms (1 second)
- **Search:** Case-insensitive substring matching on message field only
- **Empty Filtering:** Automatically filters out empty/placeholder log messages
- **Source Detection:** Logs tagged with `:elixir` or `:freeswitch` source
- **Sorting:** Logs sorted by timestamp descending (newest first)
- **PubSub:** Elixir logs delivered via Phoenix PubSub for real-time updates
- **FreeSWITCH Logs:** Collected via Event Socket Interface (ESI) log listener

---

Cell Tower Database

Manage and query the OpenCellID cell tower location database for emergency services and location-based features.

Access: Navigate to `/cell_towers` in the Control Panel

Features

- **Database Statistics:** View total records, coverage by country/network
- **Search & Query:**
  - Search by MCC (Mobile Country Code)
  - Search by MNC (Mobile Network Code)
  - Search by radio type (GSM, UMTS, LTE)
  - Search by location string

- **Database Management:**
  - Import cell tower data
  - Re-download latest dataset from OpenCellID
  - View import status and progress
- **Location Resolution:** Resolve cell IDs to geographic coordinates

Use Cases

- Emergency call location determination
- Subscriber location tracking (with consent)
- Network coverage analysis
- Troubleshooting roaming location issues
- Cell tower database maintenance

Data Source

Cell tower data is sourced from OpenCellID (https://opencellid.org/), a collaborative community project to create a free database of cell tower locations worldwide.

Call Simulator

Interactive call simulation tool for testing dialplan logic without making real calls.

Access: Navigate to `/simulator` in the Control Panel

Detailed Documentation: See HLR and Call Simulator Guide

Features

- **Simulate Call Types:** Test MO, MT, and Emergency calls
- **Configurable Parameters:**
  - Source and destination numbers
  - Source IP address (to simulate SBC/CSCF)
  - Force specific call disposition
  - Skip OCS authorization for faster testing

- **Comprehensive Results:**
  - Complete dialplan variable output
  - Sh/HLR lookup results
  - OCS authorization result
  - SS7 MAP query results (if applicable)
  - Generated dialplan XML
- **Step-by-Step Processing:** View each stage of call processing

Use Cases

- Test dialplan changes before deployment
- Verify subscriber provisioning
- Debug call routing issues
- Train staff on call flow
- Validate OCS/HLR integration
- Test emergency call handling

HLR/MAP Testing

Test SS7 MAP operations including Send Routing Info (SRI) and Provide Roaming Number (PRN) queries.

Access: Navigate to `/hlr` in the Control Panel

Detailed Documentation: See HLR and Call Simulator Guide

Features

- **SRI Query:** Test Send Routing Info for call routing
- **PRN Query:** Test Provide Roaming Number for roaming subscribers
- **Real Results:** Actual queries to configured MAP gateway
- **Response Display:** View MSRN, MSC address, and forwarding status
- **Error Handling:** Clear display of MAP errors and timeouts

Use Cases

- Verify HLR connectivity
- Test roaming number allocation
- Debug call routing to roaming subscribers
- Validate MAP gateway configuration
- Troubleshoot call forwarding issues

OCS Testing

Test Diameter Ro (Online Charging) Credit-Control-Request (CCR) operations directly against your OCS.

Access: Navigate to `/ocs_test` in the Control Panel

Features

- **Flexible CCR Types:** Send INITIAL, UPDATE, TERMINATION, or EVENT requests
- **Session Simulation:** Reuse the same Call ID to simulate a complete session lifecycle
- **Event Type Selection:** Test both SMS (event-based) and Call (session-based) charging
- **Direction Control:** Test both outgoing (MO) and incoming (MT) scenarios
- **Optional Parameters:** Specify Destination-Host and Username for advanced testing
- **Real-time Results:** View complete CCA (Credit-Control-Answer) responses

How to Use

1. **Enter Test Parameters:**
    - **Called MSISDN:** The destination number (e.g., `61400123456`)
    - **Calling MSISDN:** The originating number (e.g., `61400987654`)
    - **Event Type:** Choose `sms` or `call`
        - SMS defaults to EVENT_REQUEST (type 4)
        - Call defaults to INITIAL_REQUEST (type 1)
    - **Direction:** `out` for MO or `in` for MT
2. **Configure CCR Type:**
    - **Request-Type:** Select the CCR type:
        - `1 — INITIAL_REQUEST` - Start a new session
        - `2 — UPDATE_REQUEST` - Mid-session re-authorization
        - `3 — TERMINATION_REQUEST` - End session and report usage
        - `4 — EVENT_REQUEST` - One-time event (SMS, immediate event)
    - **Request-Number:** Starts at 1, increment for each request in the same session
3. **Session Testing:**
    - **Call ID:** Auto-generated unique identifier for correlation
    - Click **"New ID"** to generate a fresh Call ID for a new test session
    - **Keep the same Call ID** to simulate a complete session:
        - First request: INITIAL_REQUEST (type 1, number 1)
        - Mid-session: UPDATE_REQUEST (type 2, number 2, 3, 4...)
        - Final request: TERMINATION_REQUEST (type 3, number N+1)
4. **Advanced Options:**
    - **Destination-Host:** Target a specific OCS node (optional)
    - **Username:** Override the subscriber identifier (optional)
5. **Run and Review:**
    - Click **"Run CCR"** to send the request
    - View the complete CCA response with all AVPs
    - Check result code, granted units, and validity time

    - Last run timestamp shown in top-right corner

Use Cases

- **OCS Connectivity Testing:** Verify Diameter Ro connection and authentication
- **Credit Control Logic:** Test credit allocation, consumption, and exhaustion scenarios
- **Session Flow Testing:** Simulate complete call lifecycle (INITIAL → UPDATE → TERMINATION)
- **Rating Validation:** Verify correct charging rates for different number ranges
- **Failover Testing:** Test OCS redundancy by targeting specific Destination-Host
- **Integration Debugging:** Troubleshoot OCS integration issues with detailed AVP inspection
- **Load Testing Preparation:** Validate OCS behavior before load testing
- **Emergency Number Bypass:** Verify that emergency numbers bypass charging correctly

Tips

- Use the same Call ID with incrementing Request-Numbers to test session continuity
- Monitor OCS logs simultaneously to correlate test requests
- Test UPDATE requests to verify mid-session re-authorization logic
- Verify that TERMINATION requests properly close sessions and prevent leaks
- Test credit exhaustion by sending UPDATE requests after consuming granted units

---

Sh Interface Testing

Test Diameter Sh User-Data-Request (UDR) operations to retrieve subscriber profile data from the HSS.

Access: Navigate to `/sh_test` in the Control Panel

Features

- **Multiple Data References:** Query over 20 different subscriber data types
- **Real HSS Queries:** Live Diameter Sh requests to your configured HSS
- **Complete Response Display:** View full XML subscriber data and AVPs
- **Session Tracking:** Shows HSS hostname, realm, and session ID
- **Error Handling:** Clear display of Diameter result codes and error conditions

How to Use

1. **Enter Public Identity:**
    - **Public Identity:** The subscriber's IMS Public Identity
    - Format: `sip:61400123456@ims.mncXXX.mccXXX.3gppnetwork.org`
    - Can also use `tel:+61400123456` format
2. **Select Data Reference:** Choose the type of subscriber data to retrieve:
    - **RepositoryData (0):** Complete subscriber profile
    - **IMSPublicIdentity (10):** List of public identities
    - **IMSUserState (11):** Registration state
    - **S-CSCFName (12):** Assigned S-CSCF
    - **InitialFilterCriteria (13):** iFC triggers for application servers
    - **LocationInformation (14):** Current location
    - **ChargingInformation (16):** P-Charging addresses

- **MSISDN (17):** Phone number
- **IMSI (32):** International Mobile Subscriber Identity
- **IMSPrivateUserIdentity (33):** Private user identity
- And many more...

3. **Run and Review:**
   - Click **"Fetch SH Data"** to send the UDR request
   - View the complete User-Data-Answer (UDA) response
   - Check subscriber profile XML, service data, and iFC rules
   - Session metadata shows which HSS responded

Use Cases

- **Subscriber Verification:** Confirm subscriber is provisioned in HSS
- **iFC Debugging:** Review Initial Filter Criteria and trigger points
- **Registration Troubleshooting:** Check user state and S-CSCF assignment
- **Charging Configuration:** Verify P-Charging-Function-Addresses
- **HSS Connectivity Testing:** Validate Diameter Sh connection
- **Profile Validation:** Ensure correct service profile is assigned
- **Integration Testing:** Test HSS integration after provisioning changes
- **Roaming Analysis:** Check location information and serving network

Tips

- Use **IMSPublicIdentity (10)** to see all aliases for a subscriber
- Use **RepositoryData (0)** to get the complete subscriber profile in one query
- Check **IMSUserState (11)** to verify if a subscriber is registered
- **InitialFilterCriteria (13)** shows which application servers will be triggered
- The session ID can be used to correlate queries in HSS logs
- Error responses include Diameter result codes (e.g., 5001 = User Unknown)

Number Translation Testing

Test number translation rules and formatting without making actual calls.

Access: Navigate to `/translate` in the Control Panel

Features

- **Real-time Translation:** Auto-translates as you type
- **Country Code Support:** Test different country code contexts
- **Disposition-Aware:** Apply different rules based on call disposition
- **Live Results:** Immediate feedback with translated number
- **Debug Information:** View raw return values for troubleshooting

How to Use

1. **Configure Parameters:**
    - **Country Code:** The dialing context (e.g., `AU`, `US`, `NZ`)
        - Defaults to the configured country code in `config/runtime.exs`
        - Accepts formats: `AU`, `:AU`, `au`
    - **Phone Number:** The number to translate
        - Examples: `+61400111222`, `0400111222`, `61400111222`
    - **Disposition:** (Optional) Call context for conditional rules
        - Examples: `originate`, `route`, `emergency`
2. **Test Translation:**
    - Enter values in the form

- Translation runs automatically as you type
- Or click **"Translate"** to manually trigger
- View the translated result immediately

3. **Review Results:**
   - **Translated:** Shows the formatted output number
   - **Error:** Displays validation errors or translation failures
   - **Raw return (debug):** Shows the complete Elixir tuple for debugging

Use Cases

- **Dialplan Development:** Test number formatting rules before deployment
- **Format Validation:** Verify E.164 conversion is working correctly
- **Country Code Testing:** Ensure correct handling of international prefixes
- **Emergency Number Detection:** Verify emergency numbers are properly identified
- **Short Code Handling:** Test special service codes (voicemail, etc.)
- **Trunk Preparation:** Format numbers correctly for SIP trunk requirements
- **Disposition Logic:** Test different rules for MO vs MT scenarios
- **Debugging Translation Issues:** Troubleshoot why specific numbers fail routing

Tips

- Test both local format (`0400111222`) and international format (`+61400111222`)
- Verify emergency numbers (`000`, `112`) are detected correctly
- Use disposition field to test different call scenarios (MO, MT, emergency)
- Check that short codes and internal numbers are handled appropriately
- The debug output shows the raw return value - useful for investigating issues
- Test edge cases like leading zeros, international prefixes, and special characters

---

Voicemail Management

Manage and listen to voicemail messages stored in the system.

Access: Navigate to `/voicemail` in the Control Panel

Features

- **Complete Voicemail List:** View all voicemail messages across all mailboxes
- **In-Browser Playback:** Listen to voicemail recordings directly in the web interface
- **Message Details:** View username, UUID, timestamps, file paths, and metadata
- **Delete Functionality:** Remove individual voicemail messages
- **Auto-Refresh:** Refresh button to reload latest voicemail data
- **Dynamic Columns:** Automatically displays all available database fields

How to Use

1. **View Voicemail List:**
    - Page loads automatically with all voicemail records
    - Table shows all fields from the voicemail database
    - Timestamps are automatically formatted from epoch values
    - File paths are shortened for readability
2. **Listen to Messages:**
    - Click **"▶ Play"** button next to any voicemail
    - Audio player appears with controls (play, pause, seek, volume)

- Supports WAV, MP3, and OGG formats
- Click **"Stop"** to close the audio player

3. **Delete Messages:**
   - Click **"Delete"** button to remove a voicemail
   - Confirmation prompt prevents accidental deletion
   - Page automatically refreshes after successful deletion
4. **Refresh Data:**
   - Click **"Refresh"** button in top-right to reload voicemail list
   - Useful after new voicemails are left

Message Details Displayed

The table dynamically shows all available fields, typically including:

- **Username:** Mailbox owner
- **UUID:** Unique message identifier
- **Created Epoch:** When the message was left (auto-formatted to readable date/time)
- **Read Epoch:** When the message was accessed (if applicable)
- **File Path:** Location of the audio file
- Additional metadata from the voicemail database

Use Cases

- **Subscriber Support:** Listen to voicemail messages for troubleshooting
- **Testing Voicemail Delivery:** Verify voicemails are being stored correctly
- **Message Management:** Clean up old or test voicemail messages
- **Troubleshooting Recording Issues:** Check file paths and verify audio files exist
- **Mailbox Maintenance:** Monitor voicemail storage and usage
- **Quality Assurance:** Review recorded messages for audio quality

Tips

- File paths are automatically shortened to show only the relevant portion
- Epoch timestamps are automatically converted to human-readable format
- Empty voicemail database shows "No voicemail records found"
- Audio playback uses HTML5 audio element - supported in all modern browsers
- Delete confirmation prevents accidental removal of important messages

---

TTS Prompt Management

Manage Text-to-Speech (TTS) generated audio prompts used throughout the system.

Access: Navigate to `/prompts` in the Control Panel

Features

- **Prompt Settings Display:** View current TTS voice, response format, and instructions
- **Recording Status:** See which prompts exist and which are missing
- **File Details:** View file size, modification time, and path for each prompt
- **In-Browser Playback:** Listen to prompts directly in the web interface
- **Generate Missing:** Automatically create all missing prompt files
- **Re-record Individual:** Regenerate a specific prompt with updated settings
- **Re-record All:** Regenerate all prompts (useful after changing voice or settings)

How to Use

1. **Review Prompt Settings:**
    - **Voice:** TTS voice being used (e.g., `alloy`, `nova`, `shimmer`)
    - **Response Format:** Audio format (e.g., `wav`, `mp3`, `opus`)
    - **Instructions:** Special instructions passed to TTS engine
2. **Check Recording Status:**
    - **Text:** The prompt text to be spoken
    - **Relative Path:** Where the audio file is stored
    - **Exists:** Green "Yes" if file exists, Yellow "No" if missing
    - **Size:** File size in bytes/KiB/MiB
    - **Modified:** Last modification timestamp
3. **Generate Prompts:**
    - **Generate Missing:** Creates only prompts that don't exist yet
        - Useful for initial setup or after adding new prompts
    - **Re-record All:** Regenerates all prompts regardless of existence
        - Useful after changing voice, format, or instructions
        - Use with caution as it regenerates everything
4. **Manage Individual Prompts:**
    - ▶ **Play:** Listen to the prompt (only enabled if file exists)
    - □ **Re-record:** Regenerate just this one prompt
        - Useful if one prompt sounds incorrect
        - Uses current voice and settings
5. **Listen to Prompts:**
    - Click **"▶ Play"** to hear the prompt
    - Audio player appears at bottom with full controls
    - Click **"Stop"** to close the player

Prompt Configuration

Prompts are configured in your application config:

```
config :tas, :prompts,
  voice: "nova",
  response_format: "wav",
  instructions: "Speak clearly and professionally.",
  recordings: [
    %{path: "/sounds/en/us/callie/voicemail/vm-enter_id.wav",
      text: "Please enter your mailbox ID followed by pound"},
    # ... more prompts
  ]
```

Use Cases

- **Initial Setup:** Generate all prompts after system installation
- **Voice Changes:** Re-record all prompts with a different TTS voice
- **Quality Improvement:** Fix individual prompts that don't sound right
- **Format Updates:** Regenerate prompts in different audio format (wav → mp3)
- **Text Updates:** Re-record after changing prompt text in config
- **Testing TTS:** Preview how prompts will sound before deployment
- **Troubleshooting Playback:** Verify prompt files exist and are accessible
- **Storage Management:** Check file sizes and manage disk usage

Tips

- Use **"Generate Missing"** for initial setup - it won't overwrite existing prompts
- Use **"Re-record All"** after changing voice or format in config
- Individual **"Re-record"** is useful for iterating on specific prompts
- Listen to prompts before deployment to ensure quality
- Larger response formats (wav) have better quality but use more disk space
- The instructions field can guide TTS engine for tone and pacing
- Re-recording can take time if you have many prompts - be patient
- Prompts are stored in FreeSWITCH sounds directory for easy access

---

Dialplan XML Templates

View and inspect FreeSWITCH dialplan XML templates used for call routing.

Access: Navigate to `/routing` in the Control Panel

Features

- **Template List:** View all XML dialplan templates from `priv/templates/` directory
- **File Details:** See filename and last modified timestamp for each template
- **Syntax Highlighting:** Color-coded XML display for easy reading
  - Tags in teal
  - Attributes in light blue
  - Values in orange/tan
  - Comments in green
- **Expandable View:** Click any template to view its full XML content
- **Read-Only View:** Safe inspection without risk of accidental modification
- **Scrollable Content:** Large templates scroll within fixed-height container (max 600px)

How to Use

1. **View Template List:**
   - Page loads with all `.xml` files from the templates directory
   - Sorted alphabetically by filename
   - Shows modification timestamp for each file

2. **Inspect Template:**
    - Click any row to expand and view the XML content
    - Template displays with syntax highlighting
    - Click again to collapse
3. **Read XML Content:**
    - **Tags** (teal): XML element names like `<extension>`, `<condition>`
    - **Attributes** (light blue): Attribute names like `name=`, `field=`
    - **Values** (orange): Attribute values like `"public"`, `"destination_number"`
    - **Comments** (green): XML comments `<!-- ... -->`

Use Cases

- **Review Dialplan Logic:** Inspect routing rules and call flow templates
- **Troubleshoot Call Routing:** Understand which templates are used for different call types
- **Verify Template Syntax:** Check XML structure before deployment
- **Training & Documentation:** Share template contents with team members
- **Change Auditing:** Compare modification timestamps to track updates
- **Template Development:** Reference existing templates when creating new ones

Tips

- Templates are loaded from `priv/templates/` within the TAS application
- Only `.xml` files are displayed
- Templates are read-only through the web interface
- Modification timestamps help identify recent changes
- Use this view to verify templates match your dialplan expectations
- Syntax highlighting makes complex XML easier to parse visually
- Combine with `/logs` view to correlate routing behavior with templates

Technical Details

- **Location:** Templates stored in `priv/templates/` directory
- **Format:** FreeSWITCH XML dialplan format
- **File Extension:** Only `.xml` files listed
- **Sorting:** Alphabetical by filename
- **Syntax Highlighting:** Client-side colorization using regex patterns
- **Max Display Height:** 600px with scroll for large files

ESL Command Runner

Execute FreeSWITCH Event Socket Layer (ESL) commands directly from the web interface.

Access: Navigate to `/command` in the Control Panel

Features

- **Command Execution:** Run any ESL/FreeSWITCH API command
- **Live Output:** See command results in real-time
- **Command History:** Recent commands dropdown (last 10 commands)
- **Auto-Complete Ready:** Monospace input for precise command entry
- **Error Handling:** Clear display of command errors and exceptions
- **No Auto-Execute:** Selecting history fills input but requires explicit "Run" click

How to Use

1. **Enter Command:**

- Type ESL command in the input box
- Examples:
  - `status` - Show FreeSWITCH status
  - `show channels` - List active calls
  - `uuid_dump <uuid>` - Dump all variables for a call
  - `sofia status` - Show SIP profile status
  - `reloadxml` - Reload XML dialplan
  - `version` - Show FreeSWITCH version

2. **Run Command:**

   - Click **"Run"** button to execute
   - Button shows "Running..." while executing
   - Cannot run multiple commands simultaneously

3. **View Output:**

   - Results appear in the "Output" section below
   - Successful commands show raw response
   - Errors prefixed with "ERROR:"
   - Output is scrollable with max height of 600px
   - Monospace font for aligned data

4. **Use Command History:**

   - Recent commands appear in dropdown after first execution
   - Select from "Recent:" dropdown to fill input field
   - History maintains last 10 unique commands
   - Most recent command at top
   - Selecting history does NOT auto-execute (safety feature)

Common Commands

Command	Description	Example Output
`status`	System status and uptime	FreeSWITCH running info
`show channels`	List all active calls	Channel list or "0 total"
`show calls`	Summary of active calls	Call count summary
`uuid_dump <uuid>`	All variables for a call	Complete variable dump
`uuid_kill <uuid>`	Hangup specific call	"+OK" or error
`sofia status`	SIP profile status	Profile list and states
`sofia status profile <name>`	Specific profile details	Registration count, etc
`reloadxml`	Reload dialplan XML	"+OK" confirmation
`version`	FreeSWITCH version info	Version string
`global_getvar <var>`	Get global variable	Variable value
`api help`	List available commands	Command reference

Use Cases

- **Call Debugging:** Get detailed info about active call with `uuid_dump`
- **System Status:** Check FreeSWITCH health with `status` and `show calls`
- **SIP Troubleshooting:** Inspect SIP profiles with `sofia status`
- **Dialplan Reload:** Apply config changes with `reloadxml`

- **Emergency Actions:** Kill stuck calls with `uuid_kill`
- **Variable Inspection:** Check global or channel variables

Troubleshooting

Subscribers Not Showing

- Verify OmniTAS is running
- Check Sofia profile is active: `sofia status profile internal`
- Verify database path in configuration matches actual database location

CDR Records Not Appearing

- Confirm OmniTAS CDR module is loaded
- Check CDR database exists at configured path
- Verify CDR module configuration in OmniTAS

Performance Considerations

- Large CDR databases (>1M records) may require additional indexing for optimal performance
- Consider archiving old CDR records periodically
- Subscriber registration queries are typically fast as the registration database is small

---

Configuration

Access Control

The Control Panel should be deployed behind appropriate access controls (firewall, VPN, authentication) as it provides visibility into subscriber activity and call records.

TTS Prompt Configuration

Back to Main Documentation

Configuration for Text-to-Speech (TTS) prompts using OpenAI's TTS engine.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - TTS prompts configuration (voice, instructions, recordings)
- **Operations Guide** - TTS prompt management in Control Panel

Integration & Usage

- **Dialplan Configuration** - Using prompts in dialplan with playback application
- **Voicemail** - Voicemail greeting and instruction prompts
- ⚙ **Supplementary Services** - Service announcement prompts
- **Online Charging** - Out-of-credit prompts

Prompt Configuation

You can define prompts in the config that are then generated with Text to Speech.

You can then use these in your dialplan with the `playback` commands.

For the prompts we can define "instructions" for tone, language, accent, etc, and pick the voice. The TTS engine uses OpenAI's text to speech engine, which you can test from openai.fm

```
config :tas,
  ...
  prompts: %{
    voice: "alloy",
    instructions: "Speak with a prim, British accent.",
    response_format: "wav",
    recordings: [
      %{
        text:
          "You do not have sufficient credit to make that call,
please topup your service and then try again  ",
        path: "/sounds/en/us/callie/misc/8000/out_of_credit.wav"
      },
      %{
        text: "The destination you have called is unabled to be
reached",
        path:
"/sounds/en/us/callie/misc/8000/unable_to_be_reached.wav"
      },
      %{
        text: "Your call is being transferred to emergency
services",
        path:
"/sounds/en/us/callie/misc/8000/emergency_services_transfer.wav"
      }
    ]
  }
```

Sh Interface (Subscriber Data Retrieval)

Back to Main Documentation

The Sh interface provides access to subscriber profile data from the HSS/Repository via Diameter.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - Diameter peer configuration
- **Operations Guide** - Sh interface testing in Control Panel

Call Processing Integration

- **Dialplan Configuration** - Using Sh data in dialplan variables
- ⚙ **Supplementary Services** - MMTel-Config for call forwarding
- **SS7 MAP** - HLR data vs Sh data priority

Related Interfaces

- **Online Charging** - Ro interface (also uses Diameter)
- **Number Translation** - Number normalization before Sh lookup

Monitoring

- **Metrics Reference** - Sh interface metrics and monitoring

Sh Interface (Subscriber Data Retrieval)

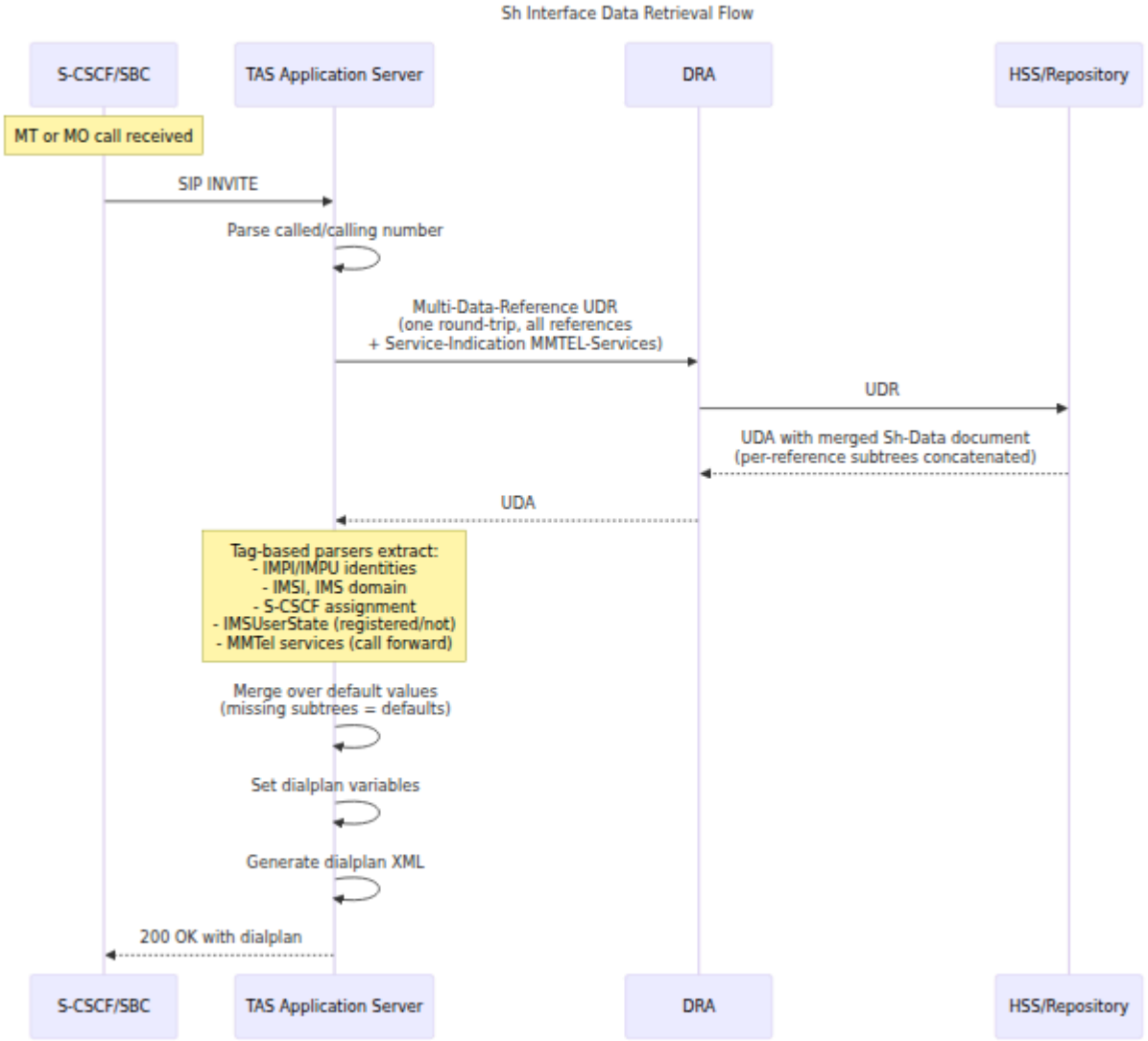
The Sh interface is used to retrieve subscriber profile data from the HSS/Repository before processing calls. This data includes subscriber identities, services, and MMTel configuration.

What is the Sh Interface?

The Sh interface is a 3GPP-standardized Diameter interface between the TAS and HSS/Repository (Repo). It provides real-time access to:

- IMS subscriber identities (IMPI/IMPU)
- Call forwarding settings (MMTel-Config)
- Subscriber service authorization
- S-CSCF assignment

When Sh Lookups Occur



Sh Lookups Happen On:

- **MT Calls:** Lookup called party (destination subscriber)
- **MO Calls:** Lookup calling party (source subscriber)
- **Emergency Calls:** Lookup calling party (for location/identity)

In every case the TAS issues exactly **one** UDR per call leg. Both MO and MT use the same multi-Data-Reference UDR shape — only the metric labels differ.

Multi-Data-Reference UDR (Notif-Eff)

Per 3GPP TS 29.328 §6.1.1.1, when both the AS and the HSS support the **Notif-Eff** feature (negotiated via the `Supported-Features` AVP), a single UDR may carry multiple `Data-Reference` AVPs and the HSS responds with one UDA

whose `User-Data-Sh` is a single `<Sh-Data>` document concatenating the per-reference subtrees as siblings. TAS relies on this — every Sh lookup fetches the full set of references the system knows how to consume in one round-trip.

Data-References requested (TS 29.328 Table 7.6.1)

Ref	Element	Used by TAS to populate
0	RepositoryData (with `Service-Indication = "MMTEL-Services"`)	`call_forward_all_destination`, `call_forward_not_reachable_destination`, `no_reply_timer`
10	IMSPublicIdentity	`ims_public_identity`, `msisdn`
11	IMSUserState	`ims_user_state` (raw enum value from TS 29.328 §7.6.3)
12	SCSCFName	`scscf_address`, `scscf_domain`
13	InitialFilterCriteria	(body returned to TAS but not currently surfaced as a dialplan variable)
14	LocationInformation	`location_rat_type`, `location_mme_name`, `location_vplmn_id`, `location_age_seconds`
15	UserState	`user_state` (raw enum value from TS 29.328 §7.6.7)
17	MSISDN	(cross-checked against IMPU)
32	IMSI	`imsi`
33	IMSPrivateUserIdentity	`ims_private_identity`, `ims_domain` (parsed from suffix)

All of the single-string fields are surfaced to the dialplan as raw string variables — TAS does not interpret their values. See the **Dialplan Variables Set from**

Sh Data table below for the full list.

Example UDA body (sanitised)

A successful merged response in the live trace looks like this — the per-reference subtrees appear under one `<Sh-Data>` wrapper in the order the references were requested:

```
[debug] Fetched caller data for +614xxxxxxxx
        (Data-Ref [0, 10, 11, 12, 13, 14, 15, 17, 32, 33],
         SI="MMTEL-Services"): 4453 bytes

<?xml version="1.0" encoding="UTF-8"?>
<Sh-Data>
  <RepositoryData></RepositoryData>
  <PublicIdentifiers>
    <IMSPublicIdentity>sip:+614xxxxxxxx@ims.mnc001.mcc999.3gppnetwork
    <IMSPublicIdentity>tel:+614xxxxxxxx</IMSPublicIdentity>
  </PublicIdentifiers>
  <ShIMSData>
    <IMSUserState>1</IMSUserState>
  </ShIMSData>
  <ShIMSData>
    <SCSCFName>sip:scscf01.ims.mnc001.mcc999.3gppnetwork.org:5060</SC
  </ShIMSData>
  <IMSSubscription>
    <PrivateID>9999990000xxxxx@ims.mnc001.mcc999.3gppnetwork.org</Pri
    <ServiceProfile>
      ... InitialFilterCriteria entries ...
    </ServiceProfile>
  </IMSSubscription>
  <ShIMSData>
    <LocationInformation>
      <RAT-Type>eutran</RAT-Type>
      <MMEName>mme01.epc.mnc001.mcc999.3gppnetwork.org</MMEName>
      <VPLMNId>999001</VPLMNId>
      <AgeOfLocationInformation>NNNN</AgeOfLocationInformation>
    </LocationInformation>
  </ShIMSData>

<IMSPrivateUserIdentity>9999990000xxxxx@ims.mnc001.mcc999.3gppnetwork
</Sh-Data>
```

How parsers consume the merged body

The TAS does not walk the XML tree. Each per-reference parser is independent and **tag-based**: it searches the merged body for a specific element name (e.g. `<SCSCFName>`, `<IMSPublicIdentity>`, `<CallForwardUnconditional>`, `<CallForwardNoReplyTimer>`, the `not-reachable` `cp:rule` block) and pulls out only the value it cares about. Subtrees that the parser does not recognise are silently ignored.

The result of each parser is a partial subscriber-data map; the partials are merged in order over a defaults map. This makes the lookup robust to heterogeneous HSS implementations and partial responses — see the **graceful degradation** section below.

Selecting which Data-References are requested (`sh_data_references`)

By default TAS requests the **full** set above (`[0, 10, 11, 12, 13, 14, 15, 17, 32, 33]` + `Service-Indication = "MMTEL-Services"`) for every lookup, on both the MO (caller) and MT (called-party) sides. This is the right default for an HSS that supports the full Sh data model.

Some HSS deployments do **not** accept the full set. Because a UDR returns the *worst* single-reference outcome, one rejected reference fails the whole request. Observed on a Nokia HSS, for example:

- `5012 DIAMETER_UNABLE_TO_COMPLY` for `LocationInformation` (14) / `UserState` (15) when the subscriber is detached (these require a real-time interrogation of the serving node).
- `5101 DIAMETER_ERROR_OPERATION_NOT_ALLOWED` for references the AS is not provisioned to read (e.g. `RepositoryData` (0), `IMSUserState` (11), `SCSCFName` (12), `InitialFilterCriteria` (13), `IMSPrivateUserIdentity` (33)).

The optional `:tas` config key **`sh_data_references`** lets operators trim the requested set **per side**. It is **optional** — when unset, the full default set is used for both sides, so existing deployments are unaffected.

```
# config/runtime.exs

# Per-side override (caller = MO source, called = MT destination).
# A side you omit falls back to the full default set.
config :tas,
  sh_data_references: [
    caller: [10, 16, 17, 32],
    called: [10, 16, 17, 32]
  ]

# Or a single bare list applied to BOTH sides:
config :tas, sh_data_references: [10, 16, 17, 32]
```

Accepted shapes:

Value	Effect
(key absent)	Full default set for both sides (`@sh_data_references`).
Keyword list `[caller: [...], called: [...]]`	Per-side override; an omitted side falls back to the default.
Bare list `[10, 17, 32]`	Same list applied to both sides.

`Service-Indication = "MMTEL-Services"` is always sent; it is harmless when `RepositoryData` (0) is not in the list. Trimming references trades data breadth for compatibility: if you drop `SCSCFName` (12) or the MMTel `RepositoryData` (0), the corresponding dialplan variables fall back to their defaults (see graceful degradation below). See `Tas.Config.sh_data_references/1`.

Session-Id call disposition tag

Every Sh UDR's `Session-Id` carries the call **disposition** as the RFC 6733 §8.8 implementation-specific "optional value" — the 4th `;`-separated field — so MO vs MT lookups can be told apart in HSS logs and packet captures:

```
example-dc01-
as01.epc.mnc001.mcc001.3gppnetwork.org;1781225439574;7468896e;mo
                                                    └ timestamp ┘ └
rand ┘ └ disposition
```

`mo` is appended for caller / originating-side lookups, `mt` for called-party / terminating-side lookups. Uniqueness is still provided by the timestamp + random; the tag is purely a correlation hint and is sanitised to `[a-z0-9_]` so it can never break the Session-Id grammar. Lookups with no disposition (e.g. the Web UI Sh Test tool) emit the plain 3-field Session-Id.

Data Retrieved from Sh Interface

The TAS issues a single multi-Data-Reference UDR per call leg (see Multi-Data-Reference UDR (Notif-Eff) above for the request shape and the merged response). The fields the TAS extracts from the merged `<Sh-Data>` body fall into three groups:

1. IMS Identities:

- **IMPI (Private Identity):** parsed from the `<IMSPrivateUserIdentity>` element. Format: `{IMSI}@{IMS-domain}`. The TAS splits on `@` to recover the IMSI and the IMS domain independently.
- **IMPU (Public Identity):** parsed from the `<IMSPublicIdentity>` element. Format: `sip:+{MSISDN}@{IMS-domain}`. The MSISDN is stripped of the leading `+` and surfaced as the `msisdn` dialplan variable.

2. S-CSCF Assignment:

- S-CSCF server name and domain where the subscriber is currently registered, parsed from the `<SCSCFName>` element (Data-Reference 12). Used by the MT dialplan to route the INVITE directly to the registered S-CSCF rather than fanning out into the IMS realm.
- **Note:** the canonical XML element name in TS 29.328 Annex D is `SCSCFName` (no hyphen). The hyphenated form "S-CSCF" only appears in spec prose.

3. MMTel Services (Multimedia Telephony Configuration):

- Returned inside `<RepositoryData>` keyed by `Service-Indication = "MMTEL-Services"`.
- Subscriber-specific call forwarding rules:
  - **Call Forward All (CFA):** Unconditional forwarding to another number
  - **Call Forward Busy (CFB):** Forward when subscriber is busy
  - **Call Forward No Reply (CFNRy):** Forward after timeout (timer value extracted from `<CallForwardNoReplyTimer>`)
  - **Call Forward Not Reachable (CFNRc):** Forward when subscriber is offline/unregistered (extracted from the `not-reachable` `<cp:rule>` inside the MMTel-Services repository document)

What is MMTel-Config?

MMTel-Config is the subscriber's Multimedia Telephony service configuration stored as transparent (repository) data in the HSS, keyed by `Service-Indication = "MMTEL-Services"`. It is fetched as part of the same multi-Data-Reference UDR as the identity lookup (Data-Reference 0 plus the service indication AVP). The document follows the OMA / 3GPP simservs XCAP schema and typically contains a `complete-communication-diversion` block with one or more `cp:rule` entries (`busy`, `noanswer`, `unregistered`, `notreachable`), an optional `<NoReplyTimer>` value, and other MMTel sub-services such as Communication Barring and Identity Presentation.

Common MMTel Services TAS recognises:

- **CDIV (Communication Diversion):** Call forwarding rules — the only block currently parsed end-to-end into dialplan variables. The `notreachable` rule populates `call_forward_not_reachable_destination` and `<NoReplyTimer>` populates `no_reply_timer`.
- **OIP (Originating Identity Presentation):** Caller ID presentation rules (returned in the body but not currently consumed).
- **TIP (Terminating Identity Presentation):** Called party number rules (returned in the body but not currently consumed).

Dialplan Variables Set from Sh Data

After a successful Sh lookup, these variables are populated:

Variable	Source
`ims_private_identity`	IMPI
`ims_public_identity`	IMPU
`msisdn`	IMPU (parsed)
`imsi`	IMPI (parsed)
`ims_domain`	IMPI/IMPU
`scscf_address`	SCSCFName
`scscf_domain`	SCSCFName (parsed)
`call_forward_all_destination`	MMTel CDIV
`call_forward_not_reachable_destination`	MMTel CDIV
`no_reply_timer`	MMTel CDIV
`ims_user_state`	IMSUserState (Data-Ref 11)

Variable	Source
`user_state`	UserState (Data-Ref 15)
`location_rat_type`	LocationInformation/RAT-Type
`location_mme_name`	LocationInformation/MMEName
`location_vplmn_id`	LocationInformation/VPLMNId
`location_age_seconds`	LocationInformation/AgeOfLoca

Priority: Sh Data vs Configuration Defaults

The TAS uses this priority order for call forwarding data:

1. **MMTel-Config from Sh** — highest priority, subscriber-specific settings.
2. **HLR Data from SS7 MAP** — overrides Sh for MT calls if roaming or call forwarding is active in the visited network. See SS7 MAP.
3. **Configuration Defaults** — lowest priority, used when neither Sh nor HLR provides a value (or when the corresponding subtree was missing from the Sh response — see graceful degradation below). The defaults are configured in `runtime.exs` under `config :tas` — `call_forward_not_reachable_destination` and `default_no_reply_timer`.

What Happens When Sh Lookup Fails

Whole-request failure scenarios:

1. **Subscriber Not Provisioned in HSS:**
    - HSS returns `Experimental-Result-Code 5001` (DIAMETER_ERROR_USER_UNKNOWN)
    - TAS treats the call leg as unresolvable
    - `hangup_case` variable set to `"UNALLOCATED_NUMBER"`
    - Call rejected with appropriate SIP response
2. **HSS Unreachable / Timeout:**
    - Sh request times out (default: 5000ms, see Diameter `request_timeout` in `runtime.exs`)
    - Error logged and metric recorded
    - Call leg fails the same way as case (1)
3. **HSS does not support multi-Data-Reference UDRs:**
    - HSS either returns an error or drops the request silently (HSS-dependent)
    - From the TAS side this looks like case (1) or (2) — the lookup fails wholesale and the call leg is rejected
    - The HSS must implement the Notif-Eff feature for TAS to function. See TS 29.328 §6.1.1.1 for the feature definition.

Graceful per-subtree degradation

When the UDR itself succeeds (`Result-Code: 2001`) but individual subtrees of the merged `<Sh-Data>` body are missing, TAS does **not** fail the call. Each per-reference parser is independent and falls back to a defined default when its tag is absent. Operators only have to worry about whole-request failures (above); partial-data degradation is automatic and observable in the debug logs.

Subtree missing	Result
`<SCSCFName>` (Data-Ref 12)	`scscf_address` and `scscf_domain` set to `"nc`
`<IMSPrivateUserIdentity>` (Data-Ref 33)	`ims_private_identity`, `imsi`, `ims_domain` s
`<CallForwardUnconditional>` inside MMTel RepositoryData	`call_forward_all_destination` set to `"none`
`not-reachable`/`<cp:rule>` block inside MMTel RepositoryData	`call_forward_not_reachable_destination` f `Tas.Config.call_forward_not_reachable_de`
`<CallForwardNoReplyTimer>` inside MMTel RepositoryData	`no_reply_timer` falls back to `Tas.Config.default_no_reply_timer()`
Empty `<RepositoryData> </RepositoryData>`	All MMTel-derived fields fall back to config de
Empty/absent `<IMSUserState>`, `<LocationInformation>`, `<InitialFilterCriteria>`	Currently no dialplan side-effect (parsed but plumbed)

The **only** hard requirement is that the response contains an `<IMSPublicIdentity>` element. If that tag is missing the lookup returns `{:error, :sh_parse_failed}` and the call leg is treated as unresolvable (same downstream behaviour as case 1 above). Every other field is "ask freely, take what you can get".

This makes TAS resilient to heterogeneous HSS deployments: an HSS that implements Notif-Eff but only populates IMSPublicIdentity, MSISDN and SCSCFName (for example) will still produce a working call; the dialplan just falls back to config defaults for the MMTel-derived variables.

Monitoring Sh Interface

Key Metrics:

```
# Sh lookup success rate
rate(subscriber_data_lookups_total{result="success"}[5m]) /
rate(subscriber_data_lookups_total[5m]) * 100

# Sh lookup latency (P95)
histogram_quantile(0.95,
  rate(subscriber_data_duration_milliseconds_bucket[5m]))

# Sh error rate
rate(subscriber_data_lookups_total{result="error"}[5m])
```

Alert Thresholds:

- P95 latency > 100ms: Slow HSS responses
- Error rate > 5%: HSS connectivity issues
- Error rate > 20%: Critical HSS failure

Troubleshooting:

1. Check Diameter peer status in Web UI (`/diameter`)
2. Test Sh lookup in Web UI (`/sh_test`) with known subscriber
3. Review logs for "Subscriber Data" errors
4. Verify HSS/Repository is reachable from TAS
5. Check `subscriber_data_lookups_total` metric for patterns

Testing Sh Interface

Use the Web UI Sh Test tool (`/sh_test`):

1. Navigate to `/sh_test` in Control Panel
2. Enter subscriber MSISDN (e.g., `+614xxxxxxxx`)
3. Click "Query Sh"
4. Review returned data:
    - IMPI/IMPU identities

    - S-CSCF assignment
    - MMTel services
    - Call forwarding configuration

Common Test Scenarios:

- Verify newly provisioned subscribers are in HSS
- Check call forwarding settings for specific subscriber
- Validate S-CSCF assignment after IMS registration
- Test HSS connectivity and response times

SS7 MAP / Gateway-MSC Configuration

 Back to Main Documentation

Configuration for HLR queries to retrieve MSRN (roaming numbers) and call forwarding information via SS7 MAP.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - SS7 MAP configuration (`ss7_map` parameters)
- **Operations Guide** - HLR/MAP testing in Control Panel

Call Processing Integration

- **Dialplan Configuration** - Using MSRN and forwarded_to_number in dialplan routing
- ⚙ **Supplementary Services** - HLR-based call forwarding (alternative to Sh/MMTel)
- **Sh Interface** - Sh vs MAP data priority
- **Number Translation** - Number format for HLR queries

Testing & Monitoring

- **HLR & Call Simulator** - Testing HLR/MAP integration
- **Metrics Reference** - HLR/MAP query metrics

Gateway-MSC Configuration

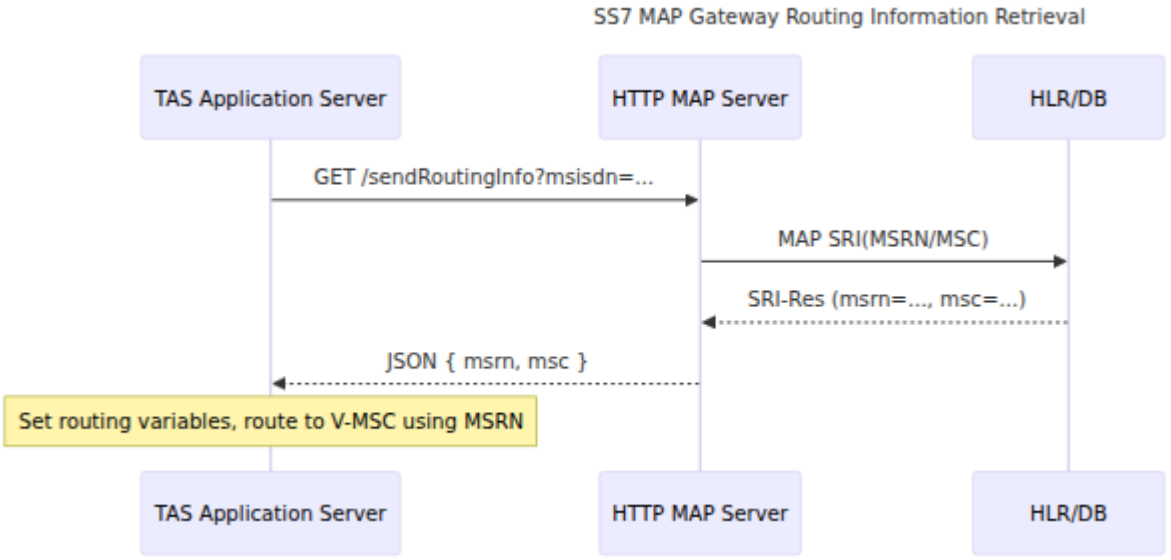
The TAS can query an HLR to retrieve the roaming number (MSRN) or MSC when a subscriber is roaming on 2G/3G networks, and can also retrieve call forwarding information.

This will set the `msrn` or `forwarded_to_number` dialplan variables which can then be used to route the call appropriately.

Configuration Parameters:

- `enabled` - Enable/disable SS7 MAP functionality
- `http_map_server_url_base` - Base URL of the MAP gateway HTTP API
- `gmsc` - Gateway MSC address used for SRI/PRN queries
- `timeout_ms` - HTTP timeout for MAP operations in milliseconds (default: 5000)

```
config :tas,
  ...
  ss7_map: %{
    enabled: true,
    http_map_server_url_base: "http://10.5.1.216:8080",
    gmsc: "55512411506",
    timeout_ms: 5000  # Optional, defaults to 5000ms
  },
```



Functionality: The TAS performs SRI (Send Routing Information) and handles routing based on the following priority:

1. **Call Forwarding Active** - If the SRI response contains a forwarded number, it is treated as an MSRN (no PRN is performed). The forwarded number is set in the `msrn` variable and used for routing.
2. **Roaming (2G/3G)** - If the subscriber is roaming (VLR present) and no call forwarding is active, performs PRN (Provide Roaming Number) to get the MSRN for routing to the V-MSC
3. **Normal** - If neither forwarding nor roaming applies, the call proceeds with standard routing

The `msrn` and `tas_destination_number` dialplan variables are set appropriately for routing (either from PRN or from the forwarded number)

Supplementary Services

Back to Main Documentation

Configuration and implementation of call forwarding, CLI blocking, and emergency calling services.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - Service configuration parameters (emergency codes, CLI blocking, default call forward)
- **Operations Guide** - Testing supplementary services

Call Processing & Data Sources

- **Dialplan Configuration** - Implementing services in dialplan logic
- **Sh Interface** - MMTel-Config for call forwarding settings
- **SS7 MAP** - HLR-based call forwarding (alternative to Sh)
- **Number Translation** - CLI blocking prefix handling

Service Interactions

- **Online Charging** - Emergency calls bypass OCS
- **Voicemail** - Call forward on busy/no-answer routes to voicemail

Monitoring

- **Metrics Reference** - Call forwarding and service metrics

- **Dialplan Metrics** - Service usage metrics

Supplementary Services (Call Forward / Blocked CLI / Emergency Codes)

Config for blocked CLI prefixes, emergency call codes, and default Call Forward data (Call Forward / No Reply data is only used when no MMTel-Config data is returned from the Repository on Sh).

```
config :tas,
  ...
  blocked_cli_prefix: ["*67"],
  call_forward_not_reachable_destination: "2222",
  default_no_reply_timer: 30,
  emergency_call_codes: ["911", "912", "913", "sos"],
  ...
```

Configuration Parameters:

- **`blocked_cli_prefix`** (list of strings): Prefixes that trigger CLI (Calling Line ID) withholding
  - Example: `["*67"]` - dialing *67 before a number hides caller ID
  - Used in dialplan to set `cli_withheld` variable
- **`call_forward_not_reachable_destination`** (string): Default destination for Call Forward Not Reachable (CFNRc)
  - Only used when no MMTel-Config is returned from Sh interface
  - Example: `"2222"` - forwards to voicemail
- **`default_no_reply_timer`** (integer): Default timeout in seconds before CFNRc activates
  - Only used when no MMTel-Config is returned from Sh interface

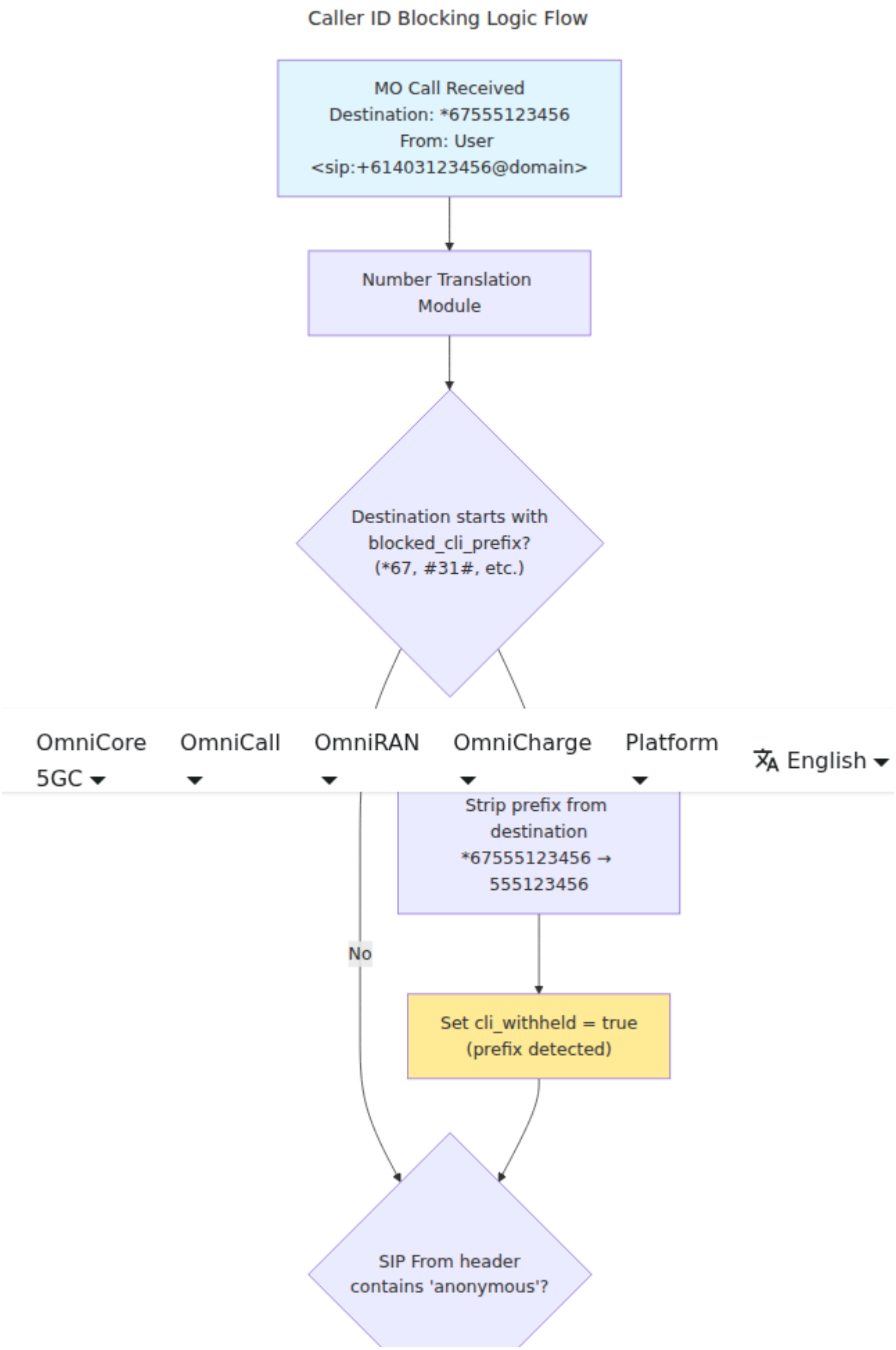
    - Example: `30` - rings for 30 seconds before forwarding

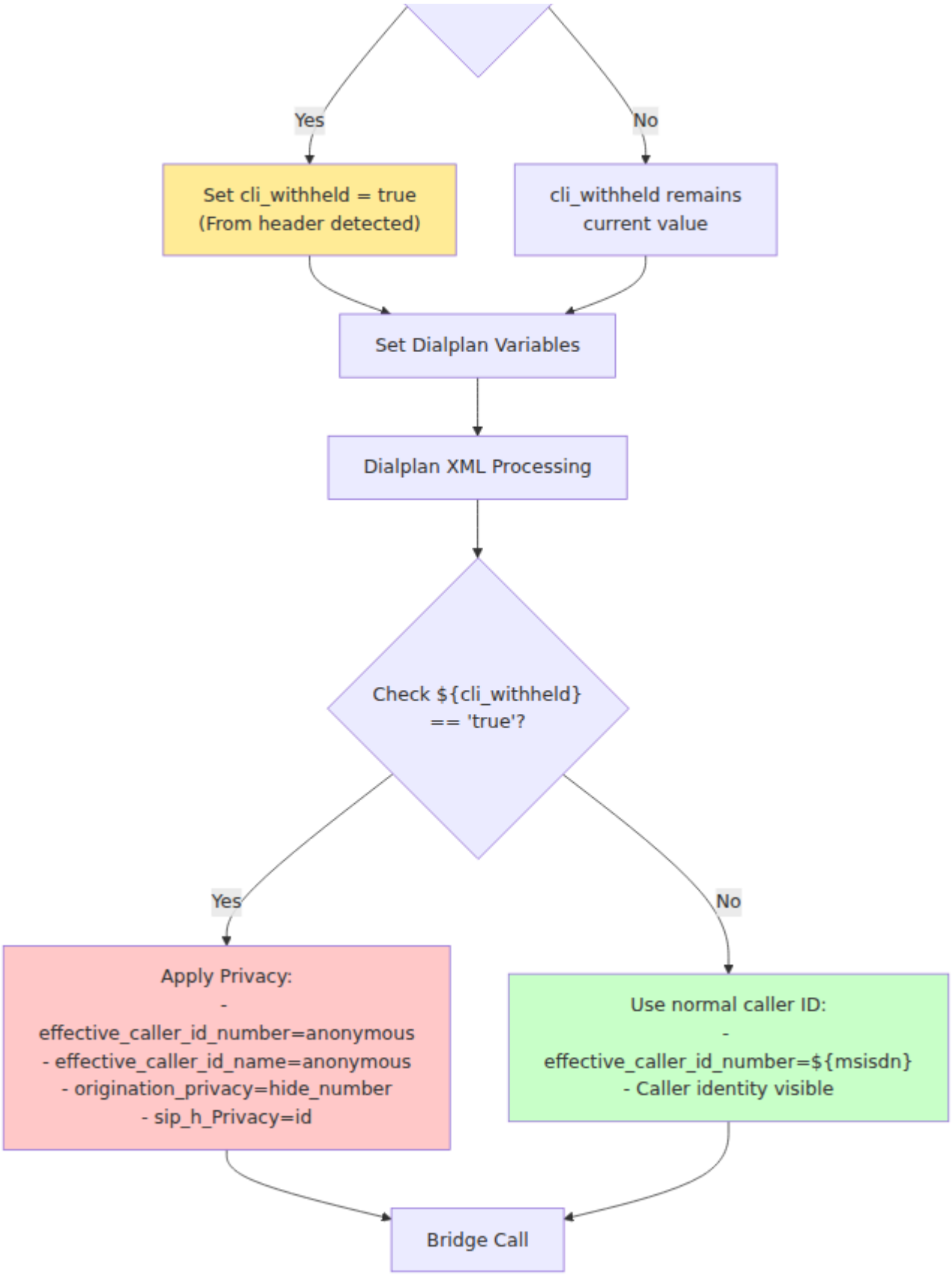
- **`emergency_call_codes`** (list of strings): Emergency service numbers for your jurisdiction

    - Checked during call authorization to detect emergency calls
    - SIP emergency URNs (e.g., `<urn:service:sos>`) are always checked in addition to these codes
    - Common examples: `["911", "112", "000", "999", "sos"]`
    - See Emergency Calling section for detailed usage

How Caller ID Blocking Works

The TAS supports two methods for blocking caller ID (CLI withholding), both of which set the `cli_withheld` dialplan variable to "true":


Caller ID Blocking Logic Flow
MO Call Received
Destination: *67555123456
From: User
<sip:+61403123456@domain>
Number Translation
Module
Destination starts with
blocked_cli_prefix?
(*67, #31#, etc.)
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
English
Strip prefix from
destination
*67555123456 →
555123456
No
Set cli_withheld = true
(prefix detected)
SIP From header
contains 'anonymous'?



Method 1: Prefix-Based Blocking

When a subscriber dials a destination number prefixed with a code from `blocked_cli_prefix`:

1. The number translation module detects the prefix (e.g., caller dials `*67555123456`)
2. The prefix is **stripped** from the destination number (becomes `555123456`)
3. The `cli_withheld` variable is set to `"true"`
4. The dialplan can then use this variable to hide the caller's identity

Example configuration:

```
blocked_cli_prefix: ["*67"]        # US-style blocking
blocked_cli_prefix: ["#31#"]       # European GSM-style blocking
blocked_cli_prefix: ["*67", "#31#"] # Support both
```

Method 2: SIP From Header Detection

When the UE/handset requests privacy via SIP headers:

1. The TAS checks if the SIP From header display name contains "anonymous" (case-insensitive)
2. If found, the `cli_withheld` variable is set to `"true"`
3. This honors the subscriber's privacy request set at the device level

Implementing CLI Blocking in Dialplan

The TAS sets the `cli_withheld` variable, but your **dialplan XML must implement the actual blocking behavior**:

```
<extension name="CLI-Privacy" continue="true">
  <condition field="${cli_withheld}" expression="true">
    <!-- Hide caller identity -->
    <action application="set"
data="effective_caller_id_name=anonymous"/>
    <action application="set"
data="effective_caller_id_number=anonymous"/>
    <action application="set"
data="origination_privacy=hide_number"/>

    <!-- Optionally set P-Asserted-Identity privacy -->
    <action application="set" data="sip_h_Privacy=id"/>
  </condition>
</extension>
```

Variables Set by TAS for CLI Blocking:

The TAS sets these variables before dialplan execution:

Variable	Type	Values	Description
`cli_withheld`	string	`"true"` or `"false"`	Indicates if CLI blocking was requested via prefix OR From header
`tas_destination_number`	string	normalized number	Destination with blocking prefix removed (e.g., `555123456`)
`destination_number`	string	normalized number	Same as `tas_destination_number` (both are set)

Variables Your Dialplan Should Set (when `cli_withheld="true"`):

These variables control how caller identity is presented:

Variable	Recommended Value	Purpose
`effective_caller_id_number`	`"anonymous"`	Hides the caller's phone number
`effective_caller_id_name`	`"anonymous"`	Hides the caller's display name
`origination_privacy`	`"hide_number"`	SIP privacy flag for outbound leg
`sip_h_Privacy`	`"id"`	SIP Privacy header (RFC 3323)
`sip_h_P-Asserted-Identity`	(unset or remove)	Optional: Remove P-Asserted-Identity header

Complete Dialplan Example:

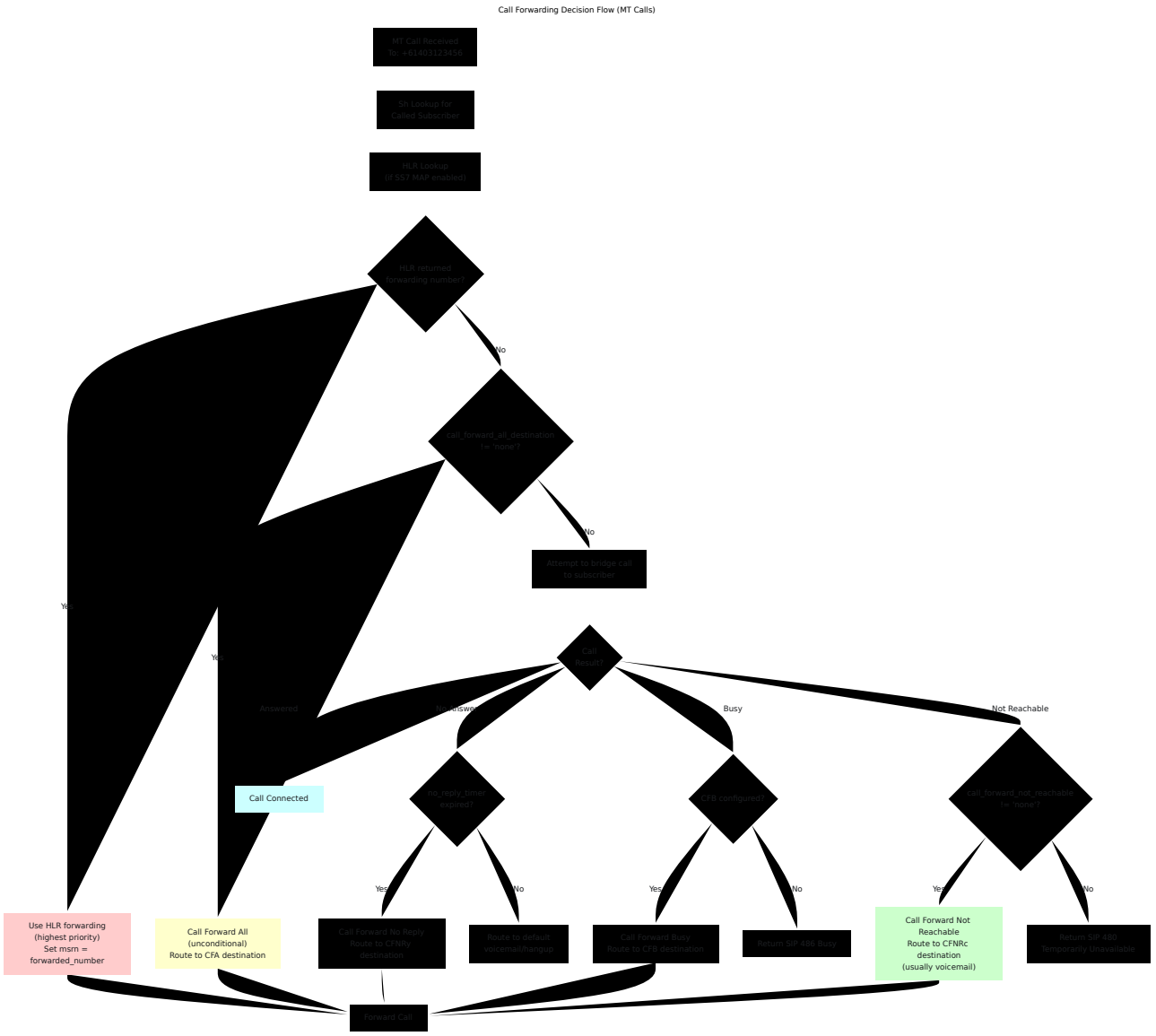
```
<extension name="CLI-Privacy-Handler" continue="true">
  <condition field="${cli_withheld}" expression="true">
    <!-- Log for troubleshooting -->
    <action application="log" data="INFO CLI blocking requested
for call to ${tas_destination_number}"/>

    <!-- Hide caller identity on outbound call -->
    <action application="set"
data="effective_caller_id_name=anonymous"/>
    <action application="set"
data="effective_caller_id_number=anonymous"/>
    <action application="set"
data="origination_privacy=hide_number"/>

    <!-- Set SIP Privacy headers -->
    <action application="set" data="sip_h_Privacy=id"/>

    <!-- Optional: Remove P-Asserted-Identity if present -->
    <action application="unset" data="sip_h_P-Asserted-Identity"/>

    <!-- Anti-action runs if cli_withheld is false -->
    <anti-action application="log" data="DEBUG Using normal caller
ID: ${msisdn}"/>
    <anti-action application="set"
data="effective_caller_id_number=${msisdn}"/>
  </condition>
</extension>

<!-- This extension continues to the actual call routing -->
<extension name="Route-Outbound-Call">
  <condition field="${tas_destination_number}"
expression="^(.+)$">
    <action application="bridge"
data="sofia/gateway/trunk/${tas_destination_number}"/>
  </condition>
</extension>
```

Important Notes:

- Both methods can work simultaneously (prefix OR SIP header triggers blocking)
- The prefix is **always stripped** from the destination number, even if dialplan doesn't implement privacy
- The `cli_withheld` variable is a **string** (`"true"` or `"false"`), not a boolean
- Call Forwarding / Blocked CLI behavior is implemented in your dialplan XML
- The example config includes these features, but if you do not define them in your dialplan, they will not function
- Variables are set during the MO (Mobile Originating) call flow only

How Call Forwarding Works

Call forwarding (also known as Communication Diversion or CDIV) allows subscribers to redirect incoming calls to another destination. The TAS supports multiple types of call forwarding with configurable behavior.



Types of Call Forwarding

1. Call Forward All (CFA) - Unconditional Forwarding

- **Variable:** `call_forward_all_destination`
- **When Active:** All incoming calls are immediately forwarded
- **Priority:** Checked first (after HLR forwarding)
- **Common Use:** Subscriber wants all calls sent to another number
- **Example:** Business calls forwarded to personal phone

2. Call Forward Busy (CFB)

- **When Active:** Call forwarded when subscriber is already on a call
- **SIP Response:** 486 Busy triggers forwarding

- **Common Use:** Forward to voicemail when on another call

3. Call Forward No Reply (CFNRy)

- **Variable:** `no_reply_timer`
- **When Active:** Call forwarded after ringing for specified seconds with no answer
- **Timeout:** Typically 15-30 seconds
- **Common Use:** Forward to voicemail if not answered

4. Call Forward Not Reachable (CFNRc)

- **Variable:** `call_forward_not_reachable_destination`
- **When Active:** Subscriber is offline, unregistered, or unreachable
- **SIP Response:** 480 Temporarily Unavailable
- **Common Use:** Forward to voicemail when phone is off
- **Default:** Configuration parameter used if no MMTel-Config

Data Source Priority

Call forwarding data is retrieved from multiple sources with this priority:

```
1. HLR Data (SS7 MAP)           [Highest Priority - overrides all]
   ↓ (if no HLR forwarding active)
2. MMTel-Config (Sh Interface)  [Subscriber-specific settings from
HSS]
   ↓ (if no MMTel-Config returned)
3. Configuration Defaults        [Lowest Priority - fallback
values]
```

Why This Priority?

- **HLR Data:** Real-time forwarding status for roaming/network scenarios
- **MMTel-Config:** Subscriber-configured preferences in IMS
- **Config Defaults:** Network-wide fallback (typically voicemail)

Dialplan Variables for Call Forwarding

Variable	Type	Source	Exa Va
`call_forward_all_destination`	string	Sh/MMTel or `"none"`	`"61403`
`call_forward_not_reachable_destination`	string	Sh/MMTel or config	`"2222"`
`no_reply_timer`	integer	Sh/MMTel or config	`30`
`msrn`	string	HLR (MT only)	`"61400`
`tas_destination_number`	string	Calculated	`"2222"`

Implementing Call Forwarding in Dialplan

Example MT Dialplan with Call Forwarding:

```
<!-- Check for Call Forward All (highest priority after HLR) -->
<extension name="Check-CFA" continue="true">
  <condition field="${call_forward_all_destination}"
expression="^(?!none$).+$">
    <action application="log" data="INFO Call Forward All active to
${call_forward_all_destination}"/>
    <action application="set"
data="tas_destination_number=${call_forward_all_destination}"/>
  </condition>
</extension>

<!-- Attempt to bridge to subscriber -->
<extension name="Bridge-To-Subscriber">
  <condition field="${msrn}" expression="^none$">
    <!-- No MSRN, route to local subscriber -->
    <action application="set" data="call_timeout=${no_reply_timer}"/>
    <action application="bridge"
data="sofia/internal/${tas_destination_number}@${scscf_address}"/>

    <!-- If bridge fails, check forwarding -->
    <action application="log" data="INFO Bridge failed, checking call
forwarding"/>

    <!-- Call Forward Not Reachable -->
    <action application="set"
data="forward_destination=${call_forward_not_reachable_destination}"/
    <action application="log" data="INFO Forwarding to
${forward_destination}"/>
    <action application="answer"/>
    <!-- Route to TAS ESL voicemail (deposit); see voicemail.md -->
    <action application="set" data="tas_vm_mode=deposit"/>
    <action application="set" data="tas_vm_mailbox=${msisdn}"/>
    <action application="set"
data="tas_vm_caller=${effective_caller_id_number}"/>
    <action application="socket" data="127.0.0.1:8084 async full"/>
  </condition>
</extension>
```

Configuring Default Call Forwarding

Set network-wide defaults in `config/runtime.exs`:

```
config :tas,
  # Default CFNRc destination (used when no MMTel-Config)
  call_forward_not_reachable_destination: "2222",  # Voicemail
access number

  # Default timeout before CFNRy activates (used when no MMTel-
Config)
  default_no_reply_timer: 30  # Ring for 30 seconds
```

When Defaults Are Used:

- Subscriber exists in HSS but has no MMTel-Config provisioned
- Sh lookup succeeds but returns no call forwarding settings
- New subscribers before call forwarding is configured

Troubleshooting Call Forwarding

Problem: Calls not forwarding as expected

1. **Check Sh Data:**
   - Use Web UI `/sh_test` to query subscriber
   - Verify MMTel-Config contains CDIV rules
   - Check `call_forward_all_destination` value
2. **Check Dialplan Variables:**
   - Review call logs for variable values
   - Confirm `call_forward_all_destination` != `"none"`
   - Verify `tas_destination_number` is set to forwarding destination
3. **Check HLR Data (if SS7 MAP enabled):**
   - Use Web UI `/hlr` to query subscriber
   - HLR forwarding overrides Sh data
   - Verify `msrn` variable doesn't contain unexpected forwarding number
4. **Check Configuration Defaults:**

- Verify `call_forward_not_reachable_destination` in config
- Confirm `default_no_reply_timer` is appropriate
- These only apply when no MMTel-Config exists

Problem: Forwarding loops

Symptoms: Call forwards to a number that forwards back, creating a loop

Prevention in Dialplan:

```
<!-- Track forwarding hop count -->
<extension name="Prevent-Forward-Loop" continue="true">
  <condition field="${sip_h_X-Forward-Hop-Count}" expression="^$">
    <action application="set" data="sip_h_X-Forward-Hop-Count=1"/>
    <anti-action application="set" data="sip_h_X-Forward-Hop-
Count=${expr(${sip_h_X-Forward-Hop-Count}+1)}"/>
  </condition>
</extension>

<extension name="Check-Forward-Hop-Limit">
  <condition field="${sip_h_X-Forward-Hop-Count}"
expression="^([3-9]|[1-9][0-9]+)$">
    <action application="log" data="ERROR Forwarding loop
detected, hop count: ${sip_h_X-Forward-Hop-Count}"/>
    <action application="hangup" data="LOOP_DETECTED"/>
  </condition>
</extension>
```

Monitoring Call Forwarding

Key Indicators:

- High rate of calls to voicemail numbers
- Pattern of calls timing out at `no_reply_timer` value
- Calls consistently routed to same forwarding destinations

Useful Logs:

```
INFO Call Forward All active to 61403555123
INFO Forwarding to 2222
INFO Bridge failed, checking call forwarding
```

Business Intelligence:

- Track forwarding activation rates by subscriber
- Monitor voicemail usage patterns
- Identify subscribers with unconditional forwarding

Voicemail & Missed Call Service

Back to Main Documentation

OmniTAS provides its own voicemail service for **deposit** (leaving a message) and **retrieval** (listening to messages). The interactive flow is driven entirely by the TAS over the FreeSWITCH Event Socket Library (ESL), rather than by FreeSWITCH's built-in `mod_voicemail` IVR. Storage stays compatible with `mod_voicemail`, so the voicemail web UI, REST API, and message-waiting (MWI) SMS notifications continue to work unchanged.

Related Documentation

Core Documentation

- **Main README** - Overview and quick start
- **Configuration Guide** - Voicemail configuration (timezone, SMSc, notification templates)
- **Operations Guide** - Voicemail management in Control Panel

Call Processing Integration

- **Dialplan Configuration** - Voicemail deposit/retrieval in dialplan
- ⚙ **Supplementary Services** - Call forward on busy/no-answer to voicemail
- **TTS Prompts** - Voicemail greeting prompts

Related Services

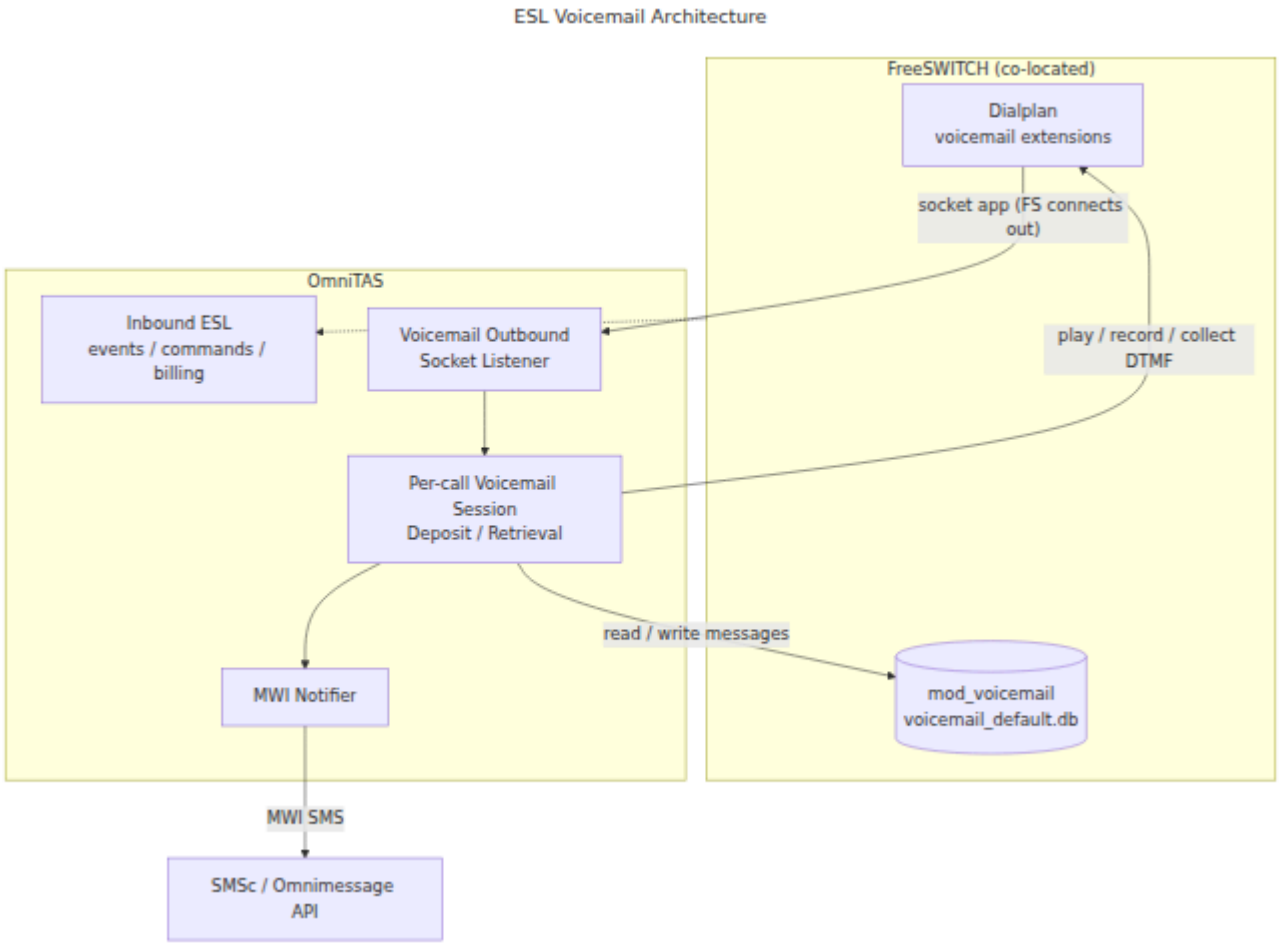
- **Number Translation** - Voicemail access number translation

Monitoring

- [ ] **Metrics Reference** - Voicemail usage metrics

Architecture

The TAS already keeps **inbound** ESL connections to FreeSWITCH for events, commands, and monitoring. Voicemail adds a second, independent path: an **outbound socket** listener that FreeSWITCH connects *into*, per call, when the dialplan hands a call to voicemail. Over that per-call socket the TAS plays prompts, records audio, and collects DTMF directly.



Key points:

- The inbound ESL path (call events, online charging, monitoring) is unchanged.
- The outbound socket is used **only** for the interactive voicemail IVR.

- The TAS and FreeSWITCH are co-located, so the outbound socket binds to loopback.
- Storage uses the `mod_voicemail` database schema, so `vm_boxcount`, the web UI, and the REST API keep working without modification. `mod_voicemail` **must remain loaded** in FreeSWITCH so the database and the `vm_boxcount` command stay available.

Routing a Call to Voicemail

Voicemail is added in the XML dialplan as needed; it is not active unless your dialplan routes a call to it. The dialplan sets three channel variables and then hands the call to the outbound socket, which the TAS reads to decide what to do.

Variable	Set when	Description
`tas_vm_mode`	Always	`deposit` to leave a message, `retrieve` to listen to messages, or `greeting` to record a personal greeting (see Personal Greetings). Selects the flow.
`tas_vm_mailbox`	Always	Mailbox owner's MSISDN. For deposit, the *called* subscriber; for retrieval, the *calling* subscriber (the box owner).
`tas_vm_caller`	Deposit	Calling party's number, stored with the message and used in the notification text.
`default_language`	Optional	Language used for the spoken "received at" date/time during retrieval. Set per subscriber in the dialplan; falls back to the `say_language` config value when unset.

The `socket` application makes FreeSWITCH connect to the TAS voicemail listener. The IP and port **must** match the `outbound_socket` configuration. The

arguments are FreeSWITCH-standard: `async` lets the TAS receive events while applications run, and `full` grants full event access.

Deposit example

Put this after a `bridge` command so it runs when the bridge fails (no answer / unreachable):

```
  <action application="log"
    data="INFO Failed to bridge Call - Routing to Call Forward No-
Answer Destination" />
  <action application="set"
    data="sip_h_History-Info=
<sip:${destination_number}@${ims_domain}>;index=1.1" />
  <action application="set"
data="sip_call_id=${sip_call_id};CALL_FORWARD_NO_ANSWER" />
  <action application="log" data="DEBUG Called Voicemail Deposit
Number for ${msisdn}" />
  <action application="set" data="default_language=fr"/>
  <action application="answer" />
  <action application="sleep" data="500"/>
  <!-- Hand the call to the TAS voicemail IVR (deposit). The TAS
records the message,
       writes the mod_voicemail row, and fires the MWI itself.
       ip:port MUST match :tas voicemail.outbound_socket in
runtime.exs. -->
  <action application="set" data="tas_vm_mode=deposit"/>
  <action application="set" data="tas_vm_mailbox=${msisdn}"/>
  <action application="set"
data="tas_vm_caller=${effective_caller_id_number}"/>
  <action application="socket" data="127.0.0.1:8084 async full"/>
```

> When a call is forwarded to voicemail, deposit to the **original** called party using the History-Info value (e.g. `tas_vm_mailbox=${history_info_value}`), not the voicemail service number. See Dialplan Configuration for History-Info handling.

Retrieval example

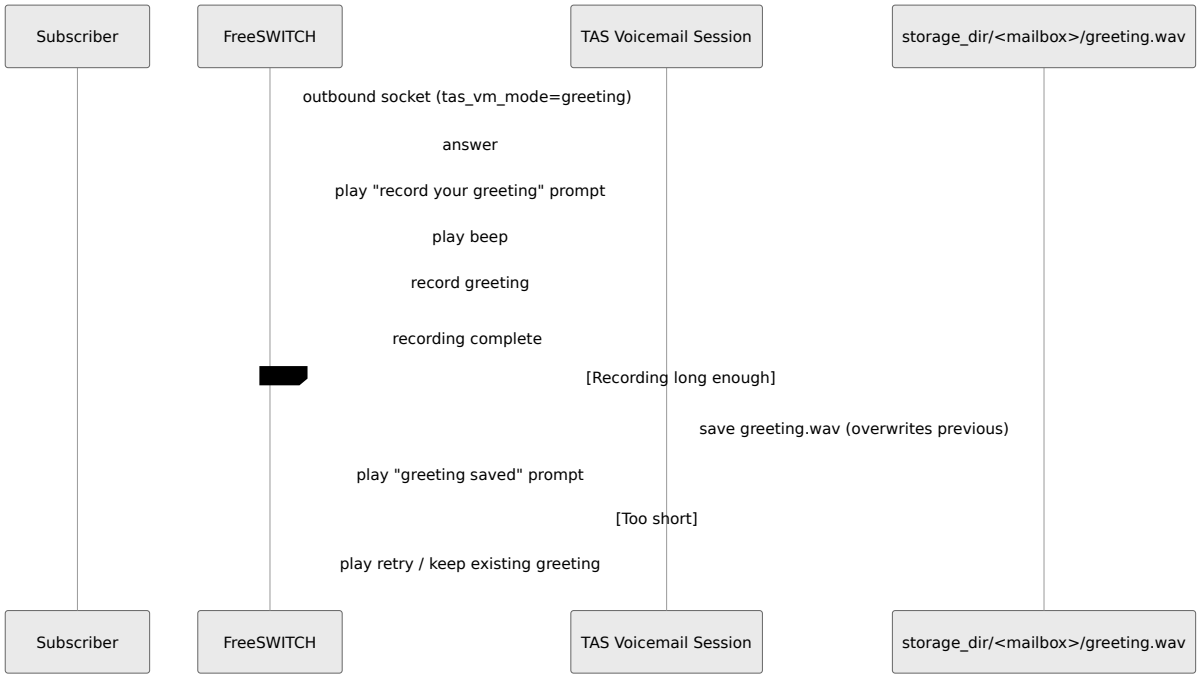
```
  <extension name="Static-Route-Voicemail-Check">
    <condition field="${tas_destination_number}"
expression="^(2222|55512411520)$">
      <action application="log" data="DEBUG Called Voicemail Check
Number" />
      <action application="set" data="default_language=fr"/>
      <action application="answer" />
      <!-- Hand the call to the TAS voicemail IVR (retrieval). The
TAS plays messages,
           handles the hear/delete/save menu, and clears the MWI.
           ip:port MUST match :tas voicemail.outbound_socket in
runtime.exs. -->
      <action application="set" data="tas_vm_mode=retrieve"/>
      <action application="set" data="tas_vm_mailbox=${msisdn}"/>
      <action application="socket" data="127.0.0.1:8084 async
full"/>
    </condition>
  </extension>
```

Greeting-recording example

Route a dedicated access number to the greeting flow (only needed when `greetings.enabled`). The mailbox is the **calling** subscriber, who is recording their own greeting:

```
  <extension name="Static-Route-Voicemail-Greeting">
    <condition field="${tas_destination_number}"
expression="^(2223)$">
      <action application="log" data="DEBUG Called Voicemail
Greeting-Record Number" />
      <action application="set" data="default_language=fr"/>
      <action application="answer" />
      <!-- Hand the call to the TAS voicemail IVR (record personal
greeting). The TAS plays the
           record prompt, records, and stores the greeting for the
mailbox.
           ip:port MUST match :tas voicemail.outbound_socket in
runtime.exs. -->
      <action application="set" data="tas_vm_mode=greeting"/>
      <action application="set" data="tas_vm_mailbox=${msisdn}"/>
      <action application="socket" data="127.0.0.1:8084 async
full"/>
    </condition>
  </extension>
```

Greeting-Recording Flow

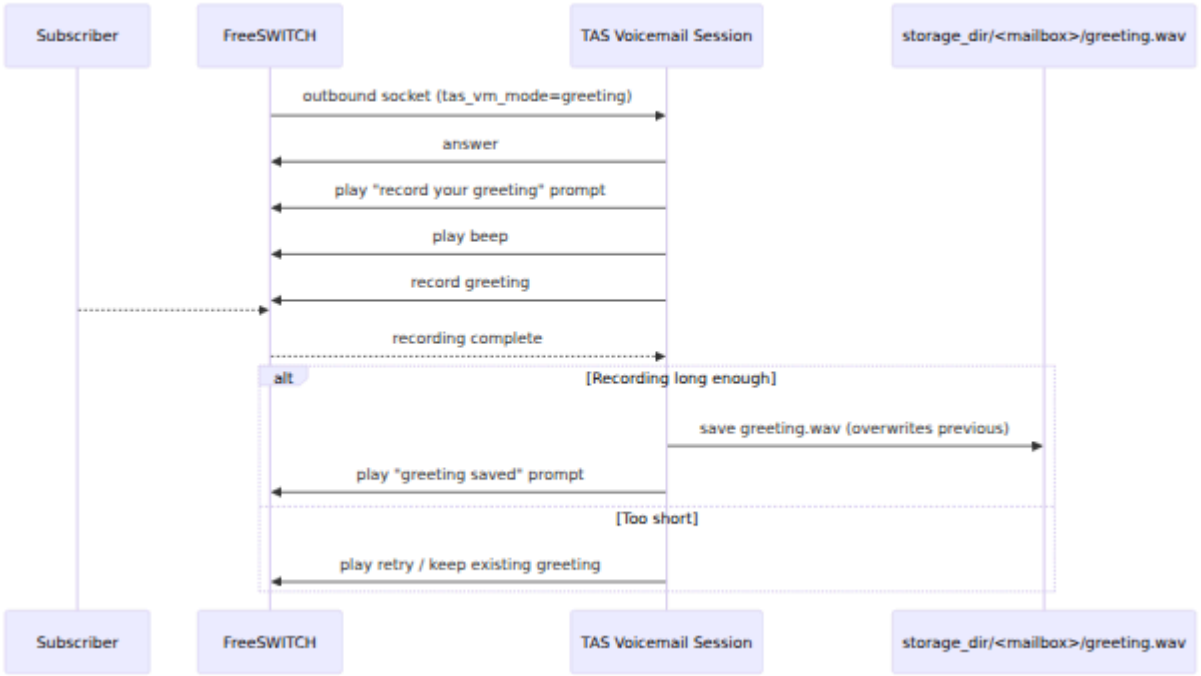


1. **Answer** the call.
2. **Play** the record-greeting prompt (`prompts.greeting_record`).

3. **Play the beep** (`prompts.beep`) and **record**, ending on the terminator key, silence, or the maximum length (the same `record` limits as a deposit).
4. If the recording meets `record.min_seconds`, any **existing greeting is cleared cluster-wide** first (so no stale copy lingers on another TAS — see the multi-TAS note below), the new one is **stored** as `greeting.wav` under the mailbox's directory in `storage_dir`, and `prompts.greeting_saved` is played; otherwise the existing greeting is kept unchanged.

The deposit flow uses this file automatically: when a greeting exists for the called mailbox it is played instead of `prompts.deposit_greeting`. To remove a recorded greeting (revert to the default), see Clearing a greeting.

Deposit Flow

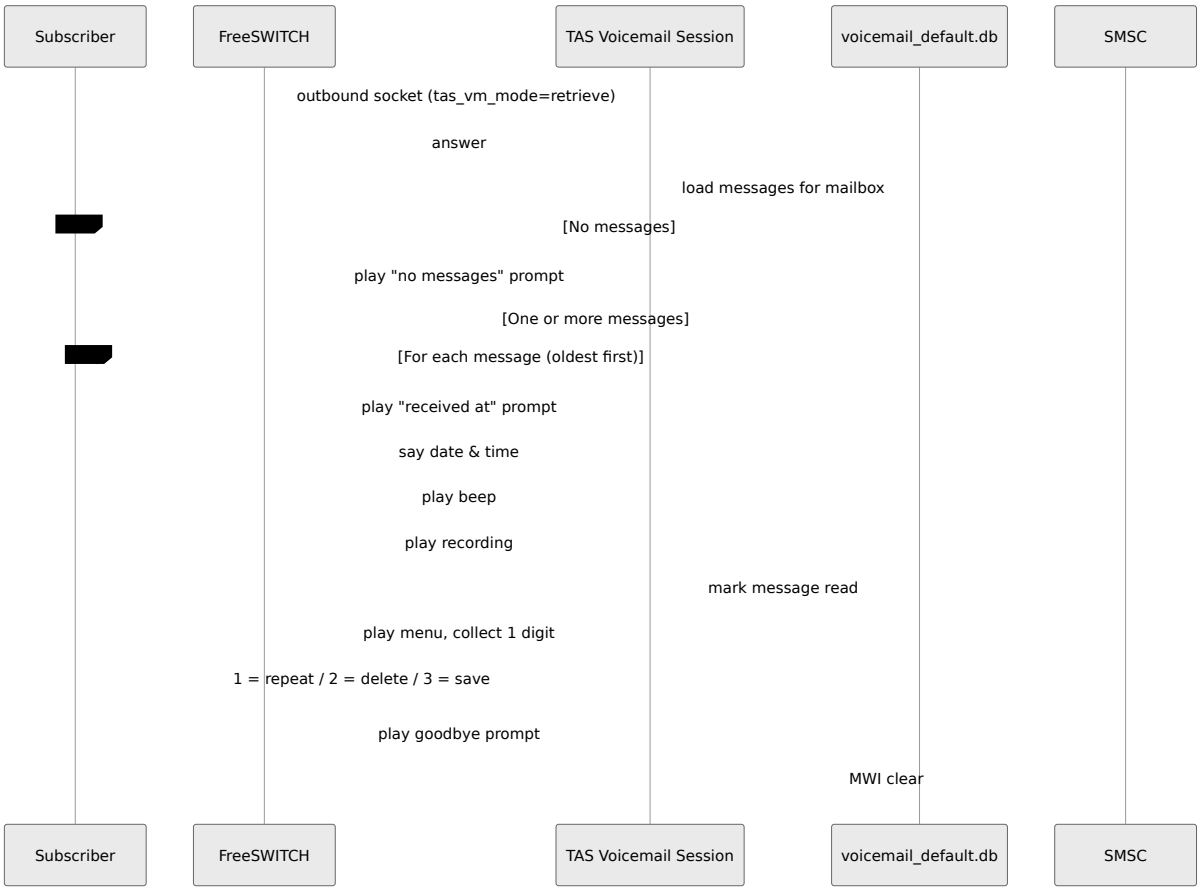


1. **Answer** the call.
2. **Play the greeting**. If the subscriber has recorded a personal greeting it is played; otherwise the default `prompts.deposit_greeting` is used.
3. **Play the beep** (`prompts.beep`).
4. **Record** to `storage_dir`, ending on caller hangup, the terminator key, silence, or the maximum length.
5. If the recording is at least `record.min_seconds`, **store** it as a new (unread) message; otherwise it is discarded and treated as a missed call.

6. **Send the MWI notification**.

The notification is sent immediately after the message is stored. This ordering is deliberate: the caller's own hangup ends the recording, so the message must be written **before** the notification is generated, so the waiting-message count is accurate.

Retrieval Flow



Per-message menu:

Key	Action	Effect
`1`	Hear again	Replays the current message from the top.
`2`	Delete	Removes the message and its recording, plays the "deleted" prompt, advances.
`3`	Save	Keeps the message (already marked read), plays the "saved" prompt, advances.
`4`	Forward	Forwards the recording to another mailbox (see Forwarding a Message). Available when forwarding is enabled.
(none / timeout)	—	Keeps the message and advances.

Playing a message marks it **read**, so it no longer counts as new. Once the subscriber has been through their messages the mailbox has no unread messages and the MWI is cleared.

The "received at" announcement combines a fixed prompt (`prompts.retrieve_received_at`, e.g. an audio file saying "Voicemail received at") with a date/time **spoken on the fly** by FreeSWITCH's `say` module, using the message's stored timestamp. The spoken language is the channel's `default_language` (set per subscriber in the dialplan), falling back to the `say_language` config value when the channel does not set one. This avoids needing a pre-recorded file for every possible date/time.

Message Waiting Indication (MWI)

Notifications are sent through the MWI notifier, which posts to the SMSc's MWI API. It chooses between a "missed call" and a "voicemail waiting" message based on the number of unread messages in the mailbox (queried with FreeSWITCH's `vm_boxcount`).

Event	Indication	Notification text used
Deposit, no message left	Inactive (missed call)	`voicemail_notification_text.not_left`
Deposit, one message waiting	Active	`voicemail_notification_text.single_voicemail`
Deposit, multiple messages waiting	Active	`voicemail_notification_text.multiple_voicemai`
Retrieval finished	Cleared	`voicemail_notification_text.cleared` (or a default)

Variables available in the notification templates:

Variable	Available in	Description
`caller`	All	Calling party number that left the message / missed call.
`day`	All	Day of month, in the configured `timezone`.
`month`	All	Month number, in the configured `timezone`.
`hour`	All	Hour (24h), in the configured `timezone`.
`minute`	All	Minute, zero-padded, in the configured `timezone`.
`message_count`	`multiple_voicemails`	Number of unread messages. Only set when the count is **greater than 1**.

Message Storage

Deposited messages are stored two ways, mirroring `mod_voicemail`:

- **Audio files** are written under `storage_dir`, organised per mailbox.
- **Metadata** (owner, caller, timestamp, file path, length, read/unread state) is written as a row in the FreeSWITCH `voicemail_default.db` SQLite database, in the standard `voicemail_msgs` table.

Because the schema matches `mod_voicemail`, `vm_boxcount`, the voicemail web UI, and the voicemail REST API all read these messages without changes. You can view voicemail box usage and message status from the Control Panel's voicemail tab.

Personal Greetings

By default, callers reaching voicemail hear the system greeting (`prompts.deposit_greeting`). Optionally, subscribers can record their **own** greeting, which is then played instead of the default for calls to their mailbox.

Personal greetings are **opt-in** (`greetings.enabled`). When disabled, or when a subscriber has not recorded one, the deposit flow falls back to the configured default greeting, so behaviour is unchanged.

Recording a greeting

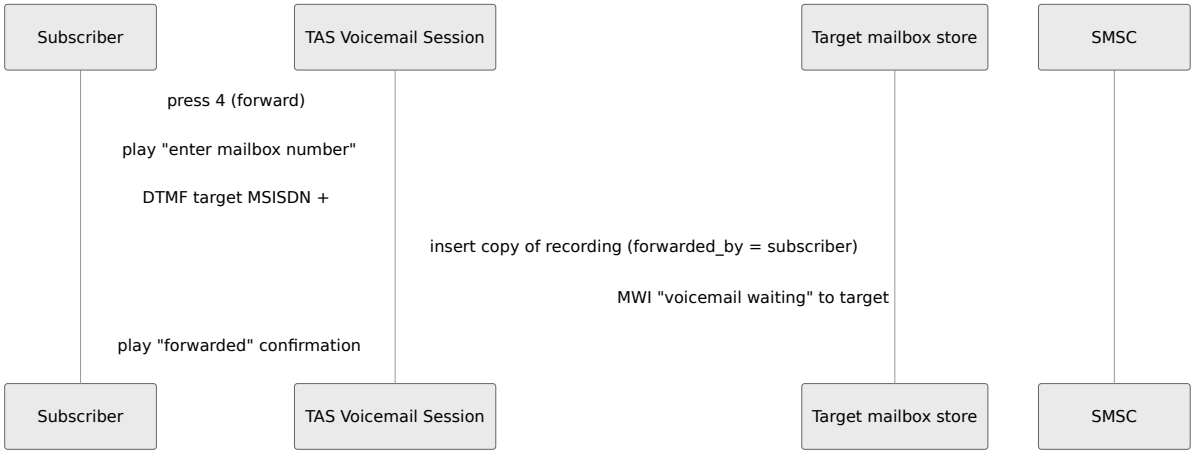
A subscriber records or replaces their greeting through a dedicated IVR flow selected with `tas_vm_mode=greeting`, routed from its own access number in the dialplan the same way deposit and retrieval are. See the Greeting-recording example dialplan and the Greeting-Recording Flow for the step-by-step behaviour.

The greeting is stored per mailbox as `greeting.wav` under that mailbox's directory in `storage_dir` (alongside its message recordings). Re-recording overwrites the previous greeting; to remove it and revert to the default, see Clearing a greeting.

> **Multi-TAS:** a greeting may be recorded on one TAS but a deposit may land on another (see Multi-TAS Voicemail), so the deposit flow locates the greeting cluster-wide — it checks locally first, then fans out to peers — before falling back to the default. To keep this unambiguous, **recording a greeting first clears any existing greeting on every TAS**, then writes the new one, so exactly one copy exists cluster-wide (a subscriber re-recording on a different node never leaves a stale greeting behind). The greeting is fetched and played the same way a remote message recording is.

Forwarding a Message to Another Mailbox

During retrieval, a subscriber can forward the message they are listening to into another subscriber's mailbox. Forwarding is **opt-in** (`forwarding.enabled`); when disabled, key `4` is ignored.



1. The subscriber presses `4` and is prompted (`prompts.forward_enter_mailbox`) to key in the target mailbox MSISDN, terminated with `#`.
2. The current recording is **copied** into the target mailbox as a new, unread message. The original caller details are preserved and the `forwarded_by` column records the forwarding subscriber, so the target sees who forwarded it.
3. An MWI notification fires to the target mailbox exactly as a normal deposit would.
4. The subscriber hears a confirmation (`prompts.forward_done`); an unknown/invalid target plays `prompts.forward_invalid` and returns to the message menu.

The original message is unaffected — forwarding leaves it in the subscriber's own mailbox.

> **Multi-TAS:** the forwarded copy becomes a normal message owned by the node that performed the forward (it copies the WAV locally and inserts the row), and the target's waiting count is computed cluster-wide, so the target sees the forwarded message from whichever TAS they retrieve on.

Message Retention and Expiry

Voicemails do not accumulate forever. A periodic sweeper deletes messages (row **and** recording) once they exceed a configured age, keeping mailboxes and disk usage bounded. Expiry is **opt-in** (`expiry.enabled`); when disabled, messages are kept until manually deleted.

- The sweeper runs every `expiry.sweep_interval_minutes`.
- A message is expired when its age (from `created_epoch`) exceeds `expiry.max_age_days`.
- Optionally, already-**read** messages can expire sooner via `expiry.read_max_age_days` (e.g. keep unread messages 30 days but purge listened-to messages after 7).
- Expiring a message removes both its `voicemail_msgs` row and its recording file, so the web UI, REST API, and waiting count all reflect the removal.

If expiring messages clears a mailbox's last unread message, the MWI is cleared for that mailbox so the handset badge stays accurate.

> **Multi-TAS:** each node sweeps **its own** store on its own schedule. Because every message has a single owning node (see below), no cross-node coordination is needed — each owner expires the messages it holds.

Multi-TAS Voicemail (Clustered Mailboxes)

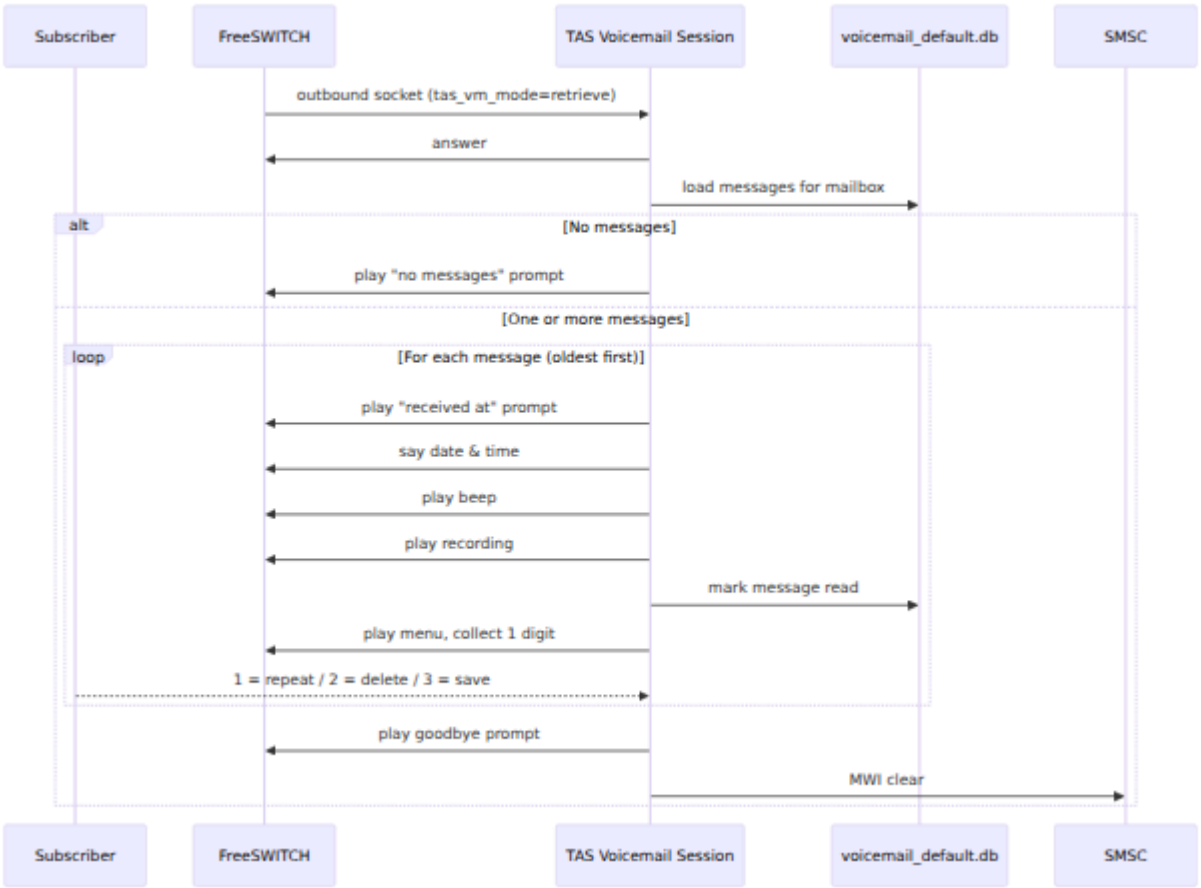
When more than one OmniTAS serves the same subscribers, a subscriber's calls can land on **any** TAS at **any** time — there is no stickiness. Since each TAS records into its own local store, a message left on TAS-A is, by default, invisible to TAS-B. Clustering makes a subscriber's mailbox behave as **one logical mailbox across every TAS**, so the waiting count reflects messages left on *any* node and retrieval from *any* node can list, play, and delete *every* message.

This is achieved **without** Erlang clustering, distributed Mnesia, or a shared/central database. Each TAS keeps its own local store; nodes simply **ask each other over HTTP** when they need the full picture. Clustering is **opt-in** —

when the `cluster` config key is absent the TAS behaves exactly as a single node.

Partitioned ownership, scatter-gather reads

Each voicemail stays where it was recorded. The node that took the deposit is the message's **owning node** and keeps both the metadata row and the recording WAV. **Nothing is replicated and there is no schema change** — ownership is simply whichever node returns the row. When a node needs the full view of a mailbox it queries its own store and **fans out an HTTP request to every other TAS**, then merges the results. Because each message has exactly one owner, marking-read and delete always route back to that single owner, so there are **no distributed write conflicts**.



Node discovery

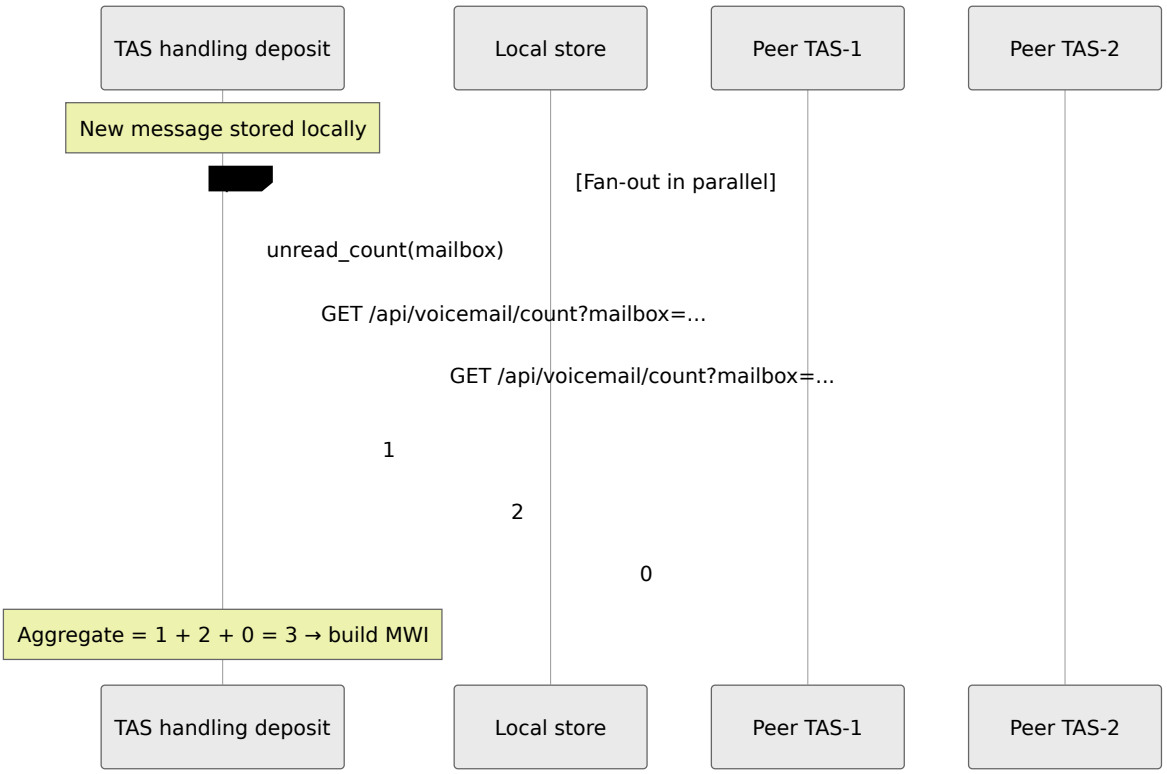
A TAS finds its peers in one of two ways (set by `cluster.discovery`); in both cases it **excludes itself** so it never queries itself over HTTP.

- **Static (`:static`)** — the full member list is configured in `runtime.exs` and deployed identically to every node. Each node identifies itself with `self_id` and drops that entry. Predictable and dependency-free; adding/removing a node means editing config everywhere.
- **DNS (`:dns`)** — the node resolves a DNS name that returns every TAS address (A/AAAA records, one per TAS) and builds a peer URL per address, dropping its own. A short DNS TTL keeps the peer set current without redeploying config; an SRV record can carry the port per target.

Either way the peer set is just a list of base URLs; everything downstream is identical.

Getting the count (fan-out)

The waiting-message count is computed by **fanning out to every other TAS and summing** — it runs on every deposit because the MWI text depends on it.



Fan-out runs concurrently with a short per-request timeout, so a slow or unreachable peer never blocks the deposit flow.

Retrieval from any node (media streaming)

On retrieval the handling node builds the **merged mailbox** — its own messages plus every peer's — sorted oldest-first, each message tagged with its owning node. To play a **remote** message it streams the WAV from the owner via `GET /api/voicemail/media`, writes it to a temporary file under its local `storage_dir` (so the co-located FreeSWITCH can read it), plays it, then removes the temp file. Recordings are fetched **lazily**, only when a message is about to play, so a subscriber who hangs up early triggers no further transfers. Marking-read and delete are routed to the owning node (`POST /api/voicemail/mark_read` / `.../delete`).

Inter-TAS HTTP API

Fan-out runs over the existing TAS HTTP listener. The read/single-write endpoints operate on the node's **own local store only** — the cluster-wide view is assembled by the *calling* node.

Method & Path	Purpose
`GET /api/voicemail/count?mailbox=<msisdn>`	Unread count in this node's store.
`GET /api/voicemail/messages?mailbox=<msisdn>`	This node's message rows for the mailbox.
`GET /api/voicemail/media?mailbox=<msisdn>&uuid=<uuid>`	Streams the recording WAV held by this node.
`GET /api/voicemail/greeting?mailbox=<msisdn>`	Streams this node's personal greeting for the mailbox, if any.
`POST /api/voicemail/mark_read {mailbox, uuid}`	Marks one message read in this node's store.
`POST /api/voicemail/delete {mailbox, uuid}`	Deletes one message (row + recording) from this node.
`DELETE /api/voicemail/greeting?mailbox=<msisdn>`	Clears the personal greeting (reverts to default). Cluster-aware — see management.
`DELETE /api/voicemail/mailbox?mailbox=<msisdn>`	Deletes **all** messages (rows + recordings) for the mailbox. Cluster-aware — see management.

The `GET`/`POST` endpoints above act on the **local store only** — the cluster-wide view is assembled by the *calling* node. The two `DELETE` management actions instead fan out to peers; the peer leg carries a `?scope=local` flag so a fanned-out request is applied locally and **not** re-fanned, preventing recursion.

Failure handling (best-effort, never silent)

A single unreachable peer must not break voicemail for everyone:

- **Counting** — a peer that times out contributes `0`. This can briefly *under*-count, but every such failure is **logged and recorded as a metric**; it is never silently swallowed.
- **Retrieval** — messages from an unreachable peer are omitted (logged); a media fetch that fails mid-session is logged and skipped rather than dropping the call.
- A recovering peer simply rejoins the next fan-out — there is no re-sync step, because nothing was ever replicated.

Security

Inter-TAS requests carry voicemail content, so the cluster endpoints require a **shared secret** header (verified by the receiving node) and optional **source-IP allow-listing** (the same allow-list pattern as other TAS interfaces). Self-signed certificates between nodes are accepted as for the SMSc integration.

Management API: Clearing Greetings and Mailboxes

Besides the per-node primitives above, two **management** actions let an operator (or self-care front-end) reset a subscriber's voicemail over HTTP. Both are **cluster-aware**: because a greeting or a message can live on any node, the request **fans out to every TAS** and applies the action on each node that holds the relevant data. On a single node (no `cluster` configured) the action simply applies locally.

Clearing a greeting

Removes a subscriber's personal greeting so callers revert to the default `deposit_greeting`.

```
DELETE /api/voicemail/greeting?mailbox=<msisdn>
```

The receiving node deletes its own `greeting.wav` for the mailbox (a no-op if it holds none) and, in a cluster, fans the same `DELETE` out to peers so a greeting

recorded on any node is removed. Returns success once every reachable node has applied it; an unreachable node is reported so the caller knows the clear was partial.

Clearing a mailbox

Deletes **all** messages (rows and recordings) for a mailbox — e.g. when deprovisioning a subscriber or on operator request.

```
DELETE /api/voicemail/mailbox?mailbox=<msisdn>
```

The receiving node deletes every message it owns for the mailbox and fans the `DELETE` out to peers, so messages held on any node are removed. Because this clears the last unread message everywhere, the MWI is cleared for the mailbox. As with greeting clears, a partial result is reported if any node is unreachable.

> These management endpoints are protected by the same **shared secret** (and optional source-IP allow-list) as the rest of the inter-TAS API.

Configuration

All voicemail settings live under the `:tas` application's `voicemail` key in `runtime.exs`.

`outbound_socket`, `record`, and `menu` are optional — they default in code and are shown here commented out. The speech prompts are TTS-generated by the shared `:tas, :prompts` recordings block (see Prompt files); the `prompts` map here just points at the resulting paths.

```
config :tas,
  voicemail: %{
    timezone: "Pacific/Tahiti",                          #
Timezone used in timestamps

    say_language: "fr",                                  #
Fallback language for the spoken date/time
                                                         #
(channel default_language wins per call)
    storage_dir: "/usr/local/freeswitch/storage/voicemail", #
Where recordings are written

    # Optional — defaults shown; omit to use them:
    # outbound_socket: %{listen_ip: "127.0.0.1", listen_port:
8084},
    # record: %{max_seconds: 120, silence_threshold: 200,
silence_seconds: 5, min_seconds: 2},
    # menu: %{tries: 3, timeout_ms: 5000, terminators: "#"},

    # Prompt files FreeSWITCH plays (ABSOLUTE paths; $${base_dir}
is NOT expanded over ESL).
    # The speech prompts are generated by the :tas, :prompts
recordings block; `beep` is a tone.
    prompts: %{
      deposit_greeting:
"/usr/local/freeswitch/sounds/tas/vm/deposit_greeting.wav",
      beep: "tone_stream://%(500,0,800)",
      retrieve_received_at:
"/usr/local/freeswitch/sounds/tas/vm/received_at.wav",
      retrieve_menu:
"/usr/local/freeswitch/sounds/tas/vm/menu.wav",
      retrieve_no_messages:
"/usr/local/freeswitch/sounds/tas/vm/no_messages.wav",
      retrieve_deleted:
"/usr/local/freeswitch/sounds/tas/vm/deleted.wav",
      retrieve_saved:
"/usr/local/freeswitch/sounds/tas/vm/saved.wav",
      retrieve_goodbye:
"/usr/local/freeswitch/sounds/tas/vm/goodbye.wav",

      # Personal greetings (only needed when greetings.enabled)
      greeting_record:
"/usr/local/freeswitch/sounds/tas/vm/greeting_record.wav",
```

```
      greeting_saved:
"/usr/local/freeswitch/sounds/tas/vm/greeting_saved.wav",

      # Forwarding (only needed when forwarding.enabled)
      forward_enter_mailbox:
"/usr/local/freeswitch/sounds/tas/vm/forward_enter_mailbox.wav",
      forward_done:
"/usr/local/freeswitch/sounds/tas/vm/forward_done.wav",
      forward_invalid:
"/usr/local/freeswitch/sounds/tas/vm/forward_invalid.wav"
    },

    # Optional — per-subscriber greetings. Omit (or enabled:
false) to always use deposit_greeting.
    # greetings: %{enabled: true},

    # Optional — forward a message to another mailbox from the
retrieval menu (key 4).
    # forwarding: %{enabled: true, menu_key: "4"},

    # Optional — auto-expire old messages. Omit to keep messages
until manually deleted.
    # expiry: %{
    #   enabled: true,
    #   max_age_days: 30,          # delete messages older than
this
    #   read_max_age_days: 7,      # optional: purge already-read
messages sooner
    #   sweep_interval_minutes: 60
    # },

    # Optional — multi-TAS clustering. Omit entirely to run
single-node (no fan-out).
    # cluster: %{
    #   discovery: :static,        # :static (nodes list) or :dns
(dns_name)
    #   self_id: "tas-a",
    #   nodes: [
    #     %{id: "tas-a", base_url: "https://10.8.82.60:8080"},
    #     %{id: "tas-b", base_url: "https://10.8.82.61:8080"}
    #   ],
    #   # discovery: :dns, dns_name: "omnitas-
vm.internal.example.com", scheme: "https", port: 8080,
    #   shared_secret: System.get_env("VM_CLUSTER_SECRET"),
```

```
    #   request_timeout_ms: 1500
    # },

    smsc: %{
      smsc_url: "https://10.80.14.219:8443",             # SMSc /
Omnimessage API base URL
      source_msisdn: "2222"                              # Sender
for notification messages
    },

    # For usage of variables in this section see the MWI table
above.
    voicemail_notification_text: %{
      not_left:
        "Vous avez 1 appel manqué du <%= caller %> le <%= day
%>/<%= month %> à <%= hour %>:<%= minute %>",
      single_voicemail:
        "Vous avez un nouveau message vocal du <%= caller %> le
<%= day %>/<%= month %> à <%= hour %>:<%= minute %>. Pour le
consulter, composez le 2222.",
      multiple_voicemails:
        "Vous avez <%= message_count %> nouveaux messages vocaux.
Pour les consulter, composez le 2222."
    }
  }
```

Parameters

Parameter	Type	Required
`timezone`	String	No
`outbound_socket.listen_ip`	String	No
`outbound_socket.listen_port`	Integer	No
`say_language`	String	No
`storage_dir`	String	Yes

Parameter	Type	Required
`record.max_seconds`	Integer	No
`record.silence_threshold`	Integer	No
`record.silence_seconds`	Integer	No
`record.min_seconds`	Integer	No
`menu.tries`	Integer	No
`menu.timeout_ms`	Integer	No

Parameter	Type	Required
`menu.terminators`	String	No
`prompts.*`	String	Yes
`smsc.smsc_url`	String	Yes
`smsc.source_msisdn`	String	Yes
`voicemail_notification_text.not_left`	String	Yes
`voicemail_notification_text.single_voicemail`	String	Yes
`voicemail_notification_text.multiple_voicemails`	String	Yes

Parameter	Type	Required
`voicemail_notification_text.cleared`	String	No
`greetings.enabled`	Boolean	No
`forwarding.enabled`	Boolean	No
`forwarding.menu_key`	String	No
`expiry.enabled`	Boolean	No
`expiry.max_age_days`	Integer	No

Parameter	Type	Required
`expiry.read_max_age_days`	Integer	No
`expiry.sweep_interval_minutes`	Integer	No
`cluster`	Map	No
`cluster.discovery`	Atom	Yes (if `cluster` set)
`cluster.self_id`	String	Yes (`:static`)
`cluster.nodes`	List of maps	Yes (`:static`)
`cluster.dns_name`	String	Yes (`:dns`)

Parameter	Type	Required
`cluster.scheme`	String	No (`:dns`)
`cluster.port`	Integer	No (`:dns`)
`cluster.shared_secret`	String	Yes (if `cluster` set)
`cluster.request_timeout_ms`	Integer	No

Prompt files

Prompt values are **absolute** paths to audio files readable by FreeSWITCH (the `${base_dir}` shortcut is **not** expanded for files played over ESL), or a `tone_stream://` URI for tones.

The speech prompts are **generated at boot by the shared `:tas, :prompts` recordings block** (the same OpenAI TTS step used for the credit/announcement prompts). Add a recording there with the text and a base-relative path, then point the matching `voicemail.prompts` entry at the absolute path (the generator writes under `/usr/local/freeswitch`). Only missing files are generated.

Parameter	Description
`deposit_greeting`	Played to the caller before the beep during deposit.
`beep`	Played before recording, and before each message during retrieval. A `tone_stream://` URI (e.g. `tone_stream://%(500,0,800)`) generates the tone with no file.
`retrieve_received_at`	Fixed "voicemail received at" lead-in, followed by the spoken date/time.
`retrieve_menu`	The options menu, e.g. "to hear this again press 1, to delete press 2, to save press 3".
`retrieve_no_messages`	Played when the mailbox is empty.
`retrieve_deleted`	Confirmation played after a message is deleted.
`retrieve_saved`	Confirmation played after a message is saved.
`retrieve_goodbye`	Played at the end of the retrieval session.
`greeting_record`	Prompt to record a personal greeting (only when `greetings.enabled`).
`greeting_saved`	Confirmation after a personal greeting is saved.
`forward_enter_mailbox`	Prompt to key in the target mailbox MSISDN (only when `forwarding.enabled`).
`forward_done`	Confirmation after a message is forwarded.
`forward_invalid`	Played when the forward target mailbox is unknown/invalid.

Troubleshooting

Calls to voicemail drop immediately

Symptoms: A call routed to a voicemail extension hangs up before the greeting or menu.

Possible causes:

- The dialplan `socket` address/port does not match `outbound_socket`.
- The voicemail listener is not running.
- A firewall is blocking the loopback port (unusual when co-located).

Resolution:

1. Confirm the dialplan `socket` action uses the same IP and port as `outbound_socket`.
2. Verify the TAS is running and the voicemail listener started.
3. Confirm FreeSWITCH can reach `listen_ip`:`listen_port`.

Messages not stored or not visible in the UI

Symptoms: A message is left but does not appear in the UI/REST API, or `vm_boxcount` returns 0.

Possible causes:

- `mod_voicemail` is not loaded, so the `voicemail_msgs` table / `vm_boxcount` are unavailable.
- The recording was shorter than `record.min_seconds` and was treated as a missed call.
- `storage_dir` is not writable by FreeSWITCH.

Resolution:

1. Ensure `mod_voicemail` remains loaded in FreeSWITCH.
2. Check the recording length against `record.min_seconds`.

3. Verify `storage_dir` permissions.

No prompts are heard

Symptoms: Silence where a greeting, beep, or menu should play.

Possible causes:

- A `prompts` path is wrong or not readable by FreeSWITCH.
- A `prompts` path used `$${base_dir}`, which is not expanded over ESL.

Resolution:

1. Confirm each `prompts` value is an absolute path to a file FreeSWITCH can read.
2. Replace any `$${base_dir}`-style paths with absolute paths.

The message-waiting indicator does not clear

Symptoms: After listening to all messages, the handset still shows messages waiting.

Possible causes:

- The SMSc rejected the clear because the message body was empty.
- Unread messages remain (the session ended before all messages were heard).

Resolution:

1. Ensure a `cleared` (or default) notification body is configured.
2. Confirm the subscriber listened to all messages; only played messages are marked read.

Count is too low / messages missing across a cluster

Symptoms: A subscriber's badge shows fewer messages than were left, or messages left on another TAS are not heard during retrieval.

Possible causes:

- A peer TAS is down or unreachable, so it contributed nothing to the fan-out (logged + metric).
- The `cluster.shared_secret` does not match between nodes, so peers reject the requests.
- Discovery is misconfigured (wrong `nodes` list, or `dns_name` not returning all addresses).
- A firewall blocks the HTTP listener port between TAS nodes.

Resolution:

1. Check the cluster fan-out error logs/metrics to see which peer failed.
2. Confirm `shared_secret` is identical on every node and discovery returns every node.
3. Verify each node can reach every other node's HTTP listener port.

A personal greeting is not played

Symptoms: Callers hear the default greeting even though the subscriber recorded their own.

Possible causes:

- `greetings.enabled` is not set.
- In a cluster, the greeting was recorded on a different node and that node was unreachable during the deposit's greeting lookup.

Resolution:

1. Confirm `greetings.enabled` is true and the greeting was saved (confirmation prompt heard).

2. Confirm the node that recorded the greeting is reachable from the depositing node.

Old messages are not being removed

Symptoms: Mailboxes keep messages indefinitely / disk usage grows.

Possible causes:

- `expiry.enabled` is not set, or `max_age_days` is unset.
- The sweep interval has not elapsed yet.

Resolution:

1. Confirm `expiry.enabled` is true and `max_age_days` is configured.
2. Allow up to `sweep_interval_minutes` for the next sweep, then check the logs.