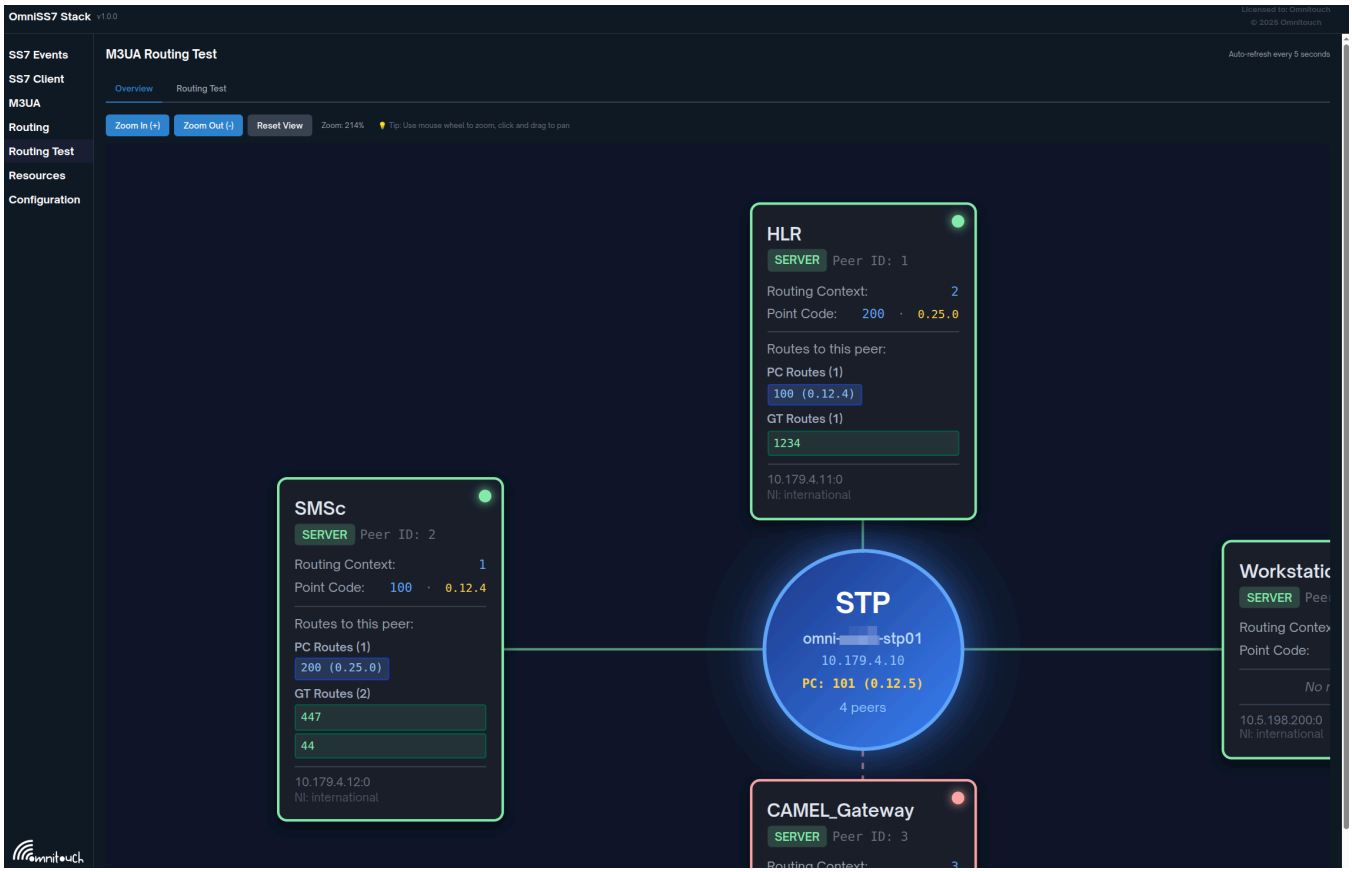
OmniSS7 - User Guide

OmniSS7 by Omnitouch Network Services is a comprehensive, general-purpose SS7 signaling stack that provides flexible network element functionality.



Documentation Overview

This documentation is organized by network element role. Choose the guide that matches your deployment:

◇ Configuration Guides

- **STP Guide** - Signal Transfer Point Configuration
  - Route SS7 traffic between network peers
  - Point Code and Global Title routing
  - Load balancing and topology hiding
  - **Use this if you're routing SS7 traffic between networks**
- **MAP Client Guide** - MAP Client Configuration
  - Connect as M3UA client to send MAP requests
  - HLR queries, authentication, routing info
  - Generic MAP protocol support
  - **Use this if you're sending MAP requests to network elements**
- **SMS Center Guide** - SMS Center (SMSc) Configuration
  - SMS message routing and delivery
  - Database-backed message queuing
  - Auto-flush and delivery reports
  - **Use this if you're operating an SMS Center**
- **HLR Guide** - Home Location Register Configuration
  - Subscriber database management
  - Authentication vector generation
  - Location updates and routing information
  - **Use this if you're operating an HLR/HSS**
- **CAMEL Gateway Guide** - CAMEL Gateway Configuration
  - Intelligent network services (CAP/CAMEL)
  - Real-time call control and charging
  - OCS integration for billing
  - Interactive request builder and session monitoring
  - **Use this if you're providing IN services or real-time charging**

◇ Common Features

- **Common Features Guide** - Shared Components
  - Web UI overview and configuration
  - API documentation
  - Monitoring and metrics (Prometheus)
  - Best practices and troubleshooting

◇ Reference Documentation

- **Appendix** - Technical Reference
  - SS7 protocol specifications
  - MAP operation codes
  - TCAP transaction flows
  - Character encodings and formats

Quick Start

1. System Overview

OmniSS7 can operate in different modes depending on your network requirements:

3. Configuration

OmniSS7 can run in 5 different operational modes. The configuration file `config/runtime.exs` contains complete, ready-to-use examples.

To switch modes:

1. Open `config/runtime.exs`
2. Uncomment your desired configuration section (STP, HLR, SMSc, or CAMEL GW)
3. Comment out the other sections
4. Update IP addresses and API URLs as needed
5. Restart the application

→ See the mode-specific guides below for complete configuration instructions

Example configurations in `runtime.exs`:

STP Mode:

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,
  camelgw_mode_enabled: false,
  map_client_m3ua: %{...}
```

HLR Mode:

```
config :omniss7,
  map_client_enabled: false,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,
  camelgw_mode_enabled: false,
  hlr_api_base_url: "...",
  map_client_m3ua: %{...}
```

SMSc Mode:

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,
  camelgw_mode_enabled: false,
  smsc_api_base_url: "...",
  auto_flush_enabled: true,
  map_client_m3ua: %{...}
```

CAMEL Gateway Mode:

```
config :omniss7,
  cap_client_enabled: true,
  camelgw_mode_enabled: true,
  ocs_enabled: true,
  ocs_url: "http://your-ocs-server/api/charging",
  cap_version: :v2,  # CAP version: :v1, :v2, :v3, or :v4
  cap_client_m3ua: %{...}
```

4. Access Web UI

Navigate to `http://localhost` (or your configured hostname)

System Architecture

Feature Matrix

Feature	STP Mode	MAP Client	SMSc Mode	HLR Mode	CAMEL GW
Point Code Routing	✅	✅	✅	✅	✅
Global Title Routing	✅	✅	✅	✅	✅
SSN Rewriting	✅	✅	✅	✅	✅
Multi-Peer Support	✅	✅	✅	✅	✅
MAP Requests (Send)	✅	✅	✅	✅	✅
MAP Responses (Receive)	✅	✅	✅	✅	✅
SMS Queue Management	✅	✅	✅	✅	✅
Auto-Flush SMS	✅	✅	✅	✅	✅
Subscriber Database	✅	✅	✅	✅	✅
Authentication Vectors	✅	✅	✅	✅	✅
Location Updates	✅	✅	✅	✅	✅
CAP/CAMEL Support	✅	✅	✅	✅	✅
Real-time Charging	✅	✅	✅	✅	✅
Call Control (IN Services)	✅	✅	✅	✅	✅
Web UI	✅	✅	✅	✅	✅
REST API	✅	✅	✅	✅	✅
Prometheus Metrics	✅	✅	✅	✅	✅

Common Operations

Web UI Access

- URL: `http://localhost` (or configured hostname)
- Swagger API: `http://localhost/swagger`
- Metrics: `http://localhost/metrics`

Monitoring

```
# Check M3UA peer status
curl http://localhost/api/m3ua-status

# View Prometheus metrics
curl http://localhost/metrics

# Check application health
curl http://localhost/api/health
```

Logs

```
# Configure log level in config/runtime.exs
config :logger,
  level: :debug  # Options: :debug, :info, :warning, :error
```

Key Capabilities

- **Full MAP Protocol Support** - MAP Phase 2/3 operations
- **CAP/CAMEL Protocol Support** - CAP v1/v2/v3/v4 for intelligent network services
- **M3UA/SCTP Signaling** - IP-based SS7 transport
- **Real-time Charging** - OCS integration for prepaid/postpaid billing
- **Real-time Message Queue** - Database-backed SMS delivery
- **Interactive Request Builder** - Web UI for CAMEL/CAP testing
- **Session Monitoring** - Real-time CAMEL call session tracking
- **Interactive API Docs** - Swagger UI for testing
- **Prometheus Metrics** - Complete observability
- **Multi-role Configuration** - STP, MAP Client, SMSc, HLR, CAMEL Gateway

---

Protocol Stack Overview

Use Case Examples

Network Gateway (STP)

Route SS7 traffic between different mobile networks

- Connect operator networks
- International SS7 gateway
- Load balancing across HLRs
- Global Title Translation
- SCCP NAT (Smart Global Title reuse)
- → **STP Guide**

SMS Center (SMSc)

Deliver SMS messages to mobile subscribers

- MT-SMS delivery
- MO-SMS origination
- SMS Home Routing
- IMSI Hiding
- SMS Firewall
- Message queue management
- Delivery reports
- Integrated into OmniMessage to handle all MAP SMS
- → **SMS Center Guide**

MAP Client

Interact with any network elements over MAP using a simple RESTful API

- PRN / SRI / ATI / etc
- Build your own SS7/MAP applications using RESTful APIs
- USSD Gateways
- Authentication vector requests
- IMSI/MSISDN lookups
- Routing information queries
- → **MAP Client Guide**

Subscriber Database (HLR)

Manage subscriber data and authentication

- Location updates
- Authentication generation
- Routing information provisioning
- Integrates fully into OmniHSS
- → **HLR Guide**

Intelligent Network Platform (CAMEL Gateway)

Real-time call control and charging for telecom operators

- Prepaid/postpaid call charging
- Call control (connect, release, routing)
- Session management and CDR generation
- Interactive request builder for testing
- → **CAMEL Gateway Guide**

Support and Resources

Documentation

Core Configuration Guides:

- STP Configuration Guide - Signal Transfer Point routing
- MAP Client Configuration Guide - MAP protocol client
- SMS Center Configuration Guide - SMS routing and delivery
- HLR Configuration Guide - Subscriber database
- CAMEL Gateway Configuration Guide - Intelligent network & charging

Integration & Reference:

- CAMEL Request Builder Guide - Interactive testing tool
- Common Features Guide - Shared components & Web UI
- Technical Reference - Protocol specifications

Contact Information

Product: OmniSS7 **Manufacturer:** Omnitouch Network Services **Documentation Version:** 2.0 **Last Updated:** 2025

For technical support, implementation assistance, or sales inquiries, please contact Omnitouch Network Services.

This documentation covers OmniSS7 runtime operation and end-user functionality. For installation, development, or advanced configuration, please refer to the technical documentation.

REST API Guide

← Back to Main Documentation

This guide provides comprehensive documentation for the OmniSS7 **REST API** and **Swagger UI**.

Table of Contents

Overview

OmniSS7 provides a REST API for programmatic access to MAP (Mobile Application Part) operations. The API allows you to:

- Send MAP requests (SRI, SRI-for-SM, UpdateLocation, etc.)
- Retrieve MAP responses
- Monitor system metrics via Prometheus

API Architecture

HTTP Server Configuration

Server Details

Parameter	Value	Configurable
Protocol	HTTP	No
IP Address	0.0.0.0 (all interfaces)	Via code only
Port	8080	Via code only
Transport	Plug.Cowboy	No

Access URL: `http://[server-ip]:8080`

Enabling/Disabling the HTTP Server

Control whether the HTTP server starts:

```
config :omniss7,
  start_http_server: true  # Set to false to disable
```

Default: `true` (enabled)

When Disabled: The HTTP server will not start, and REST API/Swagger UI will be unavailable.

Swagger UI

The API includes a **Swagger UI** for interactive API documentation and testing.

Accessing Swagger UI

URL: `http://[server-ip]:8080/swagger`

Features:

- Interactive API documentation
- Try-it-out functionality for testing endpoints
- Request/response schemas
- Example payloads

Swagger JSON

The OpenAPI specification is available at:

URL: `http://[server-ip]:8080/swagger.json`

Use Cases:

- Import into Postman or other API clients
- Generate client libraries
- API documentation automation

API Endpoints

All MAP operation endpoints follow the pattern: `POST /api/{operation}`

Endpoint Summary

Endpoint	Method	Purpose	Timeout
`/api/sri`	POST	Send Routing Info	10s
`/api/sri-for-sm`	POST	Send Routing Info for SM	10s
`/api/send-auth-info`	POST	Send Authentication Info	10s
`/api/MT-forwardSM`	POST	Mobile Terminated Forward SM	10s
`/api/forwardSM`	POST	Forward SM	10s
`/api/updateLocation`	POST	Update Location	10s
`/api/prn`	POST	Provide Roaming Number	10s
`/metrics`	GET	Prometheus metrics	N/A
`/swagger`	GET	Swagger UI	N/A
`/swagger.json`	GET	OpenAPI spec	N/A

Note: All MAP requests have a **hardcoded 10-second timeout**.

SendRoutingInfo (SRI)

Retrieve routing information for establishing a call to a mobile subscriber.

Endpoint: `POST /api/sri`

Request Body:

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567"
}
```

Parameters:

Field	Type	Required	Description
`msisdn`	String	Yes	Called party MSISDN
`gmsc`	String	Yes	Gateway MSC Global Title

Response (200 OK):

```
{
  "result": {
    "imsi": "001001234567890",
    "msrn": "5551234999",
    "vlr_number": "5551234800",
    ...
  }
}
```

Error (504 Gateway Timeout):

```
{
  "error": "timeout"
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/sri \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "gmsc": "5551234567"
  }'
```

SendRoutingInfoForSM (SRI-for-SM)

Retrieve routing information for delivering an SMS to a mobile subscriber.

Endpoint: `POST /api/sri-for-sm`

Request Body:

```
{
  "msisdn": "1234567890",
  "service_center": "5551234567"
}
```

Parameters:

Field	Type	Required	Description
`msisdn`	String	Yes	Destination MSISDN
`service_center`	String	Yes	Service Center Global Title

Response (200 OK):

```
{
  "result": {
    "imsi": "001001234567890",
    "msc_number": "5551234800",
    "location_info": {...},
    ...
  }
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "service_center": "5551234567"
  }'
```

SendAuthenticationInfo

Request authentication vectors for a subscriber.

Endpoint: `POST /api/send-auth-info`

Request Body:

```
{
  "imsi": "001001234567890",
  "vectors": 3
}
```

Parameters:

Field	Type	Required	Description
`imsi`	String	Yes	Subscriber IMSI
`vectors`	Integer	Yes	Number of authentication vectors to generate

Response (200 OK):

```
{
  "result": {
    "authentication_sets": [
      {
        "rand": "0123456789ABCDEF...",
        "xres": "...",
        "ck": "...",
        "ik": "...",
        "autn": "..."
      }
    ],
    ...
  }
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/send-auth-info \
  -H "Content-Type: application/json" \
```

```
  -d '{
    "imsi": "001001234567890",
    "vectors": 3
  }'
```

MT-ForwardSM

Deliver a Mobile Terminated SMS to a subscriber.

Endpoint: `POST /api/MT-forwardSM`

Request Body:

```
{
  "imsi": "001001234567890",
  "destination_service_centre": "5551234567",
  "originating_service_center": "5551234568",
  "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
}
```

Parameters:

Field	Type	Required	Description
`imsi`	String	Yes	Destination subscriber IMSI
`destination_service_centre`	String	Yes	Destination service center GT
`originating_service_center`	String	Yes	Originating service center GT
`smsPDU`	String	Yes	SMS TPDU in hexadecimal format

Note: `smsPDU` must be a hex-encoded string (uppercase or lowercase).

Response (200 OK):

```
{
  "result": {
    "delivery_status": "success",
    ...
  }
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/MT-forwardSM \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "destination_service_centre": "5551234567",
    "originating_service_center": "5551234568",
```

```
    "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
  }'
```

ForwardSM

Forward an SMS message (MO-SMS from subscriber).

Endpoint: `POST /api/forwardSM`

Request Body: Same as MT-ForwardSM

cURL Example:

```
curl -X POST http://localhost:8080/api/forwardSM \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "destination_service_centre": "5551234567",
    "originating_service_center": "5551234568",
    "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
  }'
```

UpdateLocation

Notify HLR of subscriber location change (VLR registration).

Endpoint: `POST /api/updateLocation`

Request Body:

```
{
  "imsi": "001001234567890",
  "vlr": "5551234800"
}
```

Parameters:

Field	Type	Required	Description
`imsi`	String	Yes	Subscriber IMSI
`vlr`	String	Yes	VLR Global Title address

Response (200 OK):

```
{
  "result": {
    "hlr_number": "5551234567",
```

```
    "subscriber_data": {...},
    ...
  }
}
```

Note: In HLR mode, this triggers InsertSubscriberData (ISD) sequence with 10-second timeout per ISD.

cURL Example:

```
curl -X POST http://localhost:8080/api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }'
```

ProvideRoamingNumber (PRN)

Request MSRN (Mobile Station Roaming Number) for call routing to roaming subscriber.

Endpoint: `POST /api/prn`

Request Body:

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567",
  "msc_number": "5551234800",
  "imsi": "001001234567890"
}
```

Parameters:

Field	Type	Required	Description
`msisdn`	String	Yes	Subscriber MSISDN
`gmsc`	String	Yes	Gateway MSC GT
`msc_number`	String	Yes	MSC number for subscriber
`imsi`	String	Yes	Subscriber IMSI

Response (200 OK):

```
{
  "result": {
    "msrn": "5551234999",
    ...
```

```
  }
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/prn \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "gmsc": "5551234567",
    "msc_number": "5551234800",
    "imsi": "001001234567890"
  }'
```

Authentication

Current Status: The API does **not require authentication**.

Security Considerations:

- API is intended for internal/trusted network use
- Consider using firewall rules to restrict access
- For production deployments, consider implementing authentication middleware

---

Response Formats

All responses use **JSON** format.

Success Response

HTTP Status: 200 OK

Structure:

```
{
  "result": {
    // Operation-specific response data
  }
}
```

Error Response

HTTP Status:

- 400 Bad Request - Invalid request body
- 504 Gateway Timeout - MAP request timeout (10 seconds)
- 404 Not Found - Invalid endpoint

Structure:

```
{
  "error": "timeout"
}
```

or

```
{
  "error": "invalid request"
}
```

Error Handling

Common Errors

Error	HTTP Code	Description	Solution
Invalid JSON	400	Request body is not valid JSON	Check JSON syntax
Missing fields	400	Required fields missing	Include all required parameters
Timeout	504	MAP request exceeded 10s timeout	Check M3UA connectivity, HLR/VLR availability
Not Found	404	Invalid endpoint	Check endpoint URL

Timeout Behavior

All MAP requests have a **hardcoded 10-second timeout**:

1. Request sent to MapClient GenServer
2. Waits for response up to 10 seconds
3. If no response → returns 504 Gateway Timeout
4. If response received → returns 200 OK with result

Troubleshooting Timeouts:

- Check M3UA connection status (Web UI → M3UA page)
- Verify network element (HLR/VLR/MSC) is reachable
- Check routing configuration
- Review SS7 event logs for errors

Metrics (Prometheus)

The API exposes Prometheus metrics for monitoring.

Metrics Endpoint

URL: `http://[server-ip]:8080/metrics`

Format: Prometheus text format

Example Output:

```
# HELP map_requests_total Total MAP requests
# TYPE map_requests_total counter
map_requests_total{operation="sri"} 42
map_requests_total{operation="sri_for_sm"} 158
map_requests_total{operation="updateLocation"} 23

# HELP cap_requests_total Total CAP requests
# TYPE cap_requests_total counter
cap_requests_total{operation="initialDP"} 87
cap_requests_total{operation="requestReportBCSMEvent"} 91

# HELP map_request_duration_milliseconds Duration of MAP request/
responses in ms
# TYPE map_request_duration_milliseconds histogram
map_request_duration_milliseconds_bucket{operation="sri",le="10"} 5
map_request_duration_milliseconds_bucket{operation="sri",le="50"} 12
map_request_duration_milliseconds_bucket{operation="sri",le="100"} 35
...

# HELP map_pending_requests Number of pending MAP TID waiters
# TYPE map_pending_requests gauge
map_pending_requests 3
```

Available Metrics

Metric	Type	Labels	Description
`map_requests_total`	Counter	`operation`	Total number of MAP requests by operation type
`cap_requests_total`	Counter	`operation`	Total number of CAP requests by operation type
`map_request_duration_milliseconds`	Histogram	`operation`	Request duration in milliseconds

Metric	Type	Labels	Description
`map_pending_requests`	Gauge	-	Number of pending MAP transactions

Prometheus Configuration

Add to your `prometheus.yml`:

```yaml
scrape_configs:
  - job_name: 'omniss7'
    static_configs:
      - targets: ['server-ip:8080']
    metrics_path: '/metrics'
    scrape_interval: 15s
```

Example Requests

Python Example

```python
import requests
import json

# SRI-for-SM Request
url = "http://localhost:8080/api/sri-for-sm"
payload = {
    "msisdn": "1234567890",
    "service_center": "5551234567"
}

response = requests.post(url, json=payload, timeout=15)

if response.status_code == 200:
    result = response.json()
    print(f"Success: {result}")
elif response.status_code == 504:
    print("Timeout - no response from network")
else:
    print(f"Error: {response.status_code} - {response.text}")
```

JavaScript Example

```javascript
const axios = require('axios');

async function sendSRI() {
  try {
    const response = await axios.post('http://localhost:8080/api/
```

```
sri', {
      msisdn: '1234567890',
      gmsc: '5551234567'
    }, {
      timeout: 15000
    });

    console.log('Success:', response.data);
  } catch (error) {
    if (error.code === 'ECONNABORTED') {
      console.error('Timeout - no response from network');
    } else {
      console.error('Error:', error.response?.data || error.message);
    }
  }
}

sendSRI();
```

Bash/cURL Example

```
#!/bin/bash

# UpdateLocation Request
response=$(curl -s -w "\n%{http_code}" -X POST http://localhost:8080/
api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }')

http_code=$(echo "$response" | tail -n 1)
body=$(echo "$response" | sed '$d')

if [ "$http_code" -eq 200 ]; then
  echo "Success: $body"
elif [ "$http_code" -eq 504 ]; then
  echo "Timeout - no response from network"
else
  echo "Error $http_code: $body"
fi
```

Flow Diagrams

API Request Flow

Summary

The OmniSS7 REST API provides:

◈ **MAP Operations** - Full support for SRI, SRI-for-SM, UpdateLocation, SMS delivery, authentication ◈ **Swagger UI** - Interactive API documentation and testing ◈ **Prometheus Metrics** - Monitoring and observability ◈ **Hardcoded Timeouts** - 10-second timeout for all MAP requests ◈ **HTTP Server** - Runs on port 8080 (configurable via `start_http_server`)

For Web UI access, see the Web UI Guide.

For configuration details, see the Configuration Reference.

Technical Reference (Appendix)

← Back to Main Documentation

Technical reference for SS7 protocols and OmniSS7 implementation.

SS7 Protocol Stack

MAP Operation Codes

Operation	Opcode	Purpose
updateLocation	2	Register subscriber location
cancelLocation	3	Deregister from VLR
provideRoamingNumber	4	Request MSRN
sendRoutingInfo	22	Query call routing
mt-forwardSM	44	Deliver SMS to subscriber
sendRoutingInfoForSM	45	Query SMS routing
mo-forwardSM	46	Forward SMS from subscriber
sendAuthenticationInfo	56	Request auth vectors

TCAP Message Types

- **BEGIN** - Start transaction
- **CONTINUE** - Mid-transaction
- **END** - Final response
- **ABORT** - Cancel transaction

SCCP Addressing

Global Title Formats

- **E.164** - International phone number (e.g., 447712345678)
- **E.212** - IMSI format (e.g., 234509876543210)
- **E.214** - Point code format

Subsystem Numbers (SSN)

- **SSN 6**: HLR

- **SSN 7**: VLR
- **SSN 8**: MSC/SMSC
- **SSN 9**: GMLC
- **SSN 10**: SGSN

SMS TPDU

Message Types

- **SMS-DELIVER** (MT) - Network to mobile
- **SMS-SUBMIT** (MO) - Mobile to network
- **SMS-STATUS-REPORT** - Delivery status
- **SMS-COMMAND** - Remote command

Character Encodings

- **GSM7** - 7-bit GSM alphabet (160 chars per SMS)
- **UCS2** - 16-bit Unicode (70 chars per SMS)
- **8-bit** - Binary data (140 bytes per SMS)

M3UA States

- **DOWN** - No SCTP connection
- **CONNECTING** - SCTP connecting
- **ASPUP_SENT** - Waiting for ASPUP ACK
- **INACTIVE** - ASP up but not active
- **ASPAC_SENT** - Waiting for ASPAC ACK
- **ACTIVE** - Ready for traffic

Common SS7 Point Codes

Point codes are typically 14-bit (ITU) or 24-bit (ANSI) values.

Example Format (ITU):

- Network: 3 bits
- Cluster: 8 bits
- Member: 3 bits

SCCP Error Codes

- **0** - No translation for address
- **1** - No translation for specific address
- **2** - Subsystem congestion
- **3** - Subsystem failure
- **4** - Unequipped user
- **5** - MTP failure
- **6** - Network congestion
- **7** - Unqualified
- **8** - Error in message transport

MAP Error Codes

Code	Error	Description
1	unknownSubscriber	Subscriber not in HLR
27	absentSubscriber	Subscriber not reachable
34	systemFailure	Network failure
35	dataMissing	Required data not available
36	unexpectedDataValue	Invalid parameter value

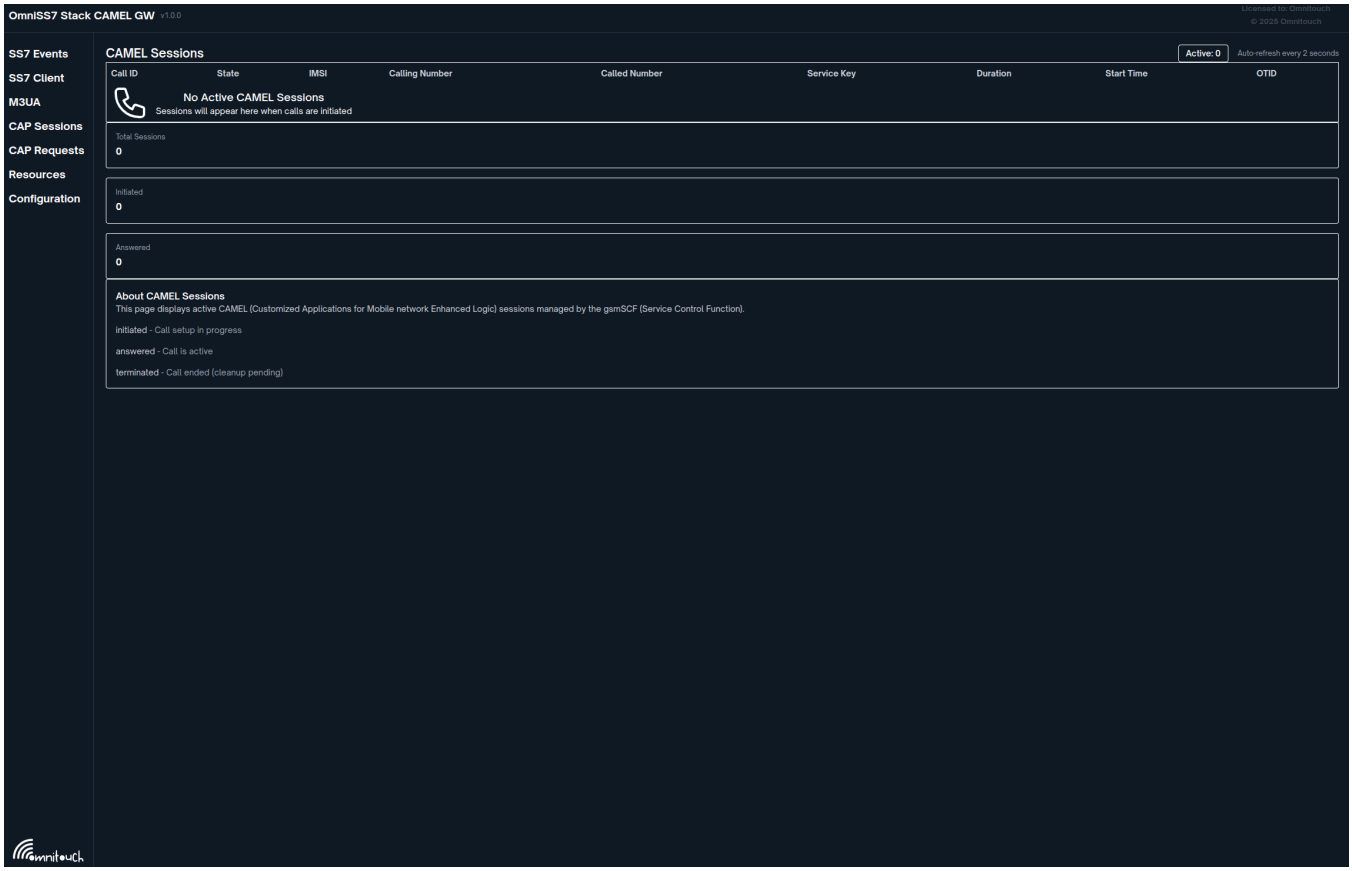
Related Documentation

- ← Back to Main Documentation
- STP Guide
- MAP Client Guide
- SMS Center Guide
- HLR Guide
- Common Features

CAMEL Gateway Configuration Guide

Overview

The **CAMEL Gateway (CAMELGW)** mode transforms OmniSS7 into an Intelligent Network (IN) platform that provides real-time call control and charging services using the CAMEL Application Part (CAP) protocol.



What is CAMEL?

CAMEL (Customized Applications for Mobile network Enhanced Logic) is a set of standards designed to work on either a GSM core network or UMTS network. It allows operators to provide services that require real-time control of calls, such as:

- **Prepaid calling** - Real-time balance checking and charging
- **Premium rate services** - Special billing for value-added services
- **Call routing control** - Dynamic destination routing based on time/location
- **Virtual private networks** - Corporate numbering plans
- **Call screening** - Allow/block calls based on criteria

CAP Protocol Versions

OmniSS7 CAMELGW supports multiple CAP versions:

Version	Phase	Features
CAP v1	CAMEL Phase 1	Basic call control, limited operations
CAP v2	CAMEL Phase 2	Enhanced operations, SMS support
CAP v3	CAMEL Phase 3	GPRS support, additional operations
CAP v4	CAMEL Phase 4	Advanced features, multimedia support

Default: CAP v2 (most widely deployed)

Architecture

Call Flow Example

Configuration

Prerequisites

- OmniSS7 installed and running
- M3UA connectivity to MSC/GMSC (gsmSSF)
- Online Charging System (OCS) with API endpoint (optional, for real-time charging)

Enable CAMEL Gateway Mode

Edit `config/runtime.exs` and configure the CAMEL Gateway section:

```
config :omniss7,
  # Mode flags - Enable CAP/CAMEL features
  cap_client_enabled: true,
  camelgw_mode_enabled: true,

  # Disable other modes
  map_client_enabled: false,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  # CAP/CAMEL Version Configuration
  # Determines which CAP version to use for outgoing requests and dialogue
  # Options: :v1, :v2, :v3, :v4
  cap_version: :v2,

  # OCS Integration (for real-time charging)
  ocs_enabled: true,
  ocs_url: "http://your-ocs-server/api/charging",
  ocs_timeout: 5000,  # milliseconds
  ocs_auth_token: "your-api-token"  # Optional, if OCS requires authentication

  # M3UA Connection Configuration for CAMEL
  # Connect as ASP (Application Server Process) for CAP operations
  cap_client_m3ua: %{
    mode: "ASP",
    callback: {CapClient, :handle_payload, []},
    process_name: :camelgw_client_asp,

    # Local endpoint (CAMELGW system)
    local_ip: {10, 179, 4, 13},
    local_port: 2905,

    # Remote endpoint (MSC/GMSC - gsmSSF)
```

```
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,

    # M3UA Parameters
    routing_context: 1,
    network_appearance: 0,
    asp_identifier: 13
  }
```

Configure Web UI Pages

The Web UI includes specialized pages for CAMEL operations:

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.TestClientLive, "/client", "SS7 Client"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAP Sessions"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAP Requests"}
  ],
  page_order: ["/events", "/client", "/m3ua", "/camel_sessions",
               "/camel_request", "/application", "/configuration"]
```

CAP Operations Supported

Incoming Operations (from gsmSSF → gsmSCF)

Operation	Opcode	Description	Handler
InitialDP	0	Initial Detection Point - call setup notification	`handle_initial_dp/1`
EventReportBCSM	6	Basic Call State Model event (answer, disconnect, etc.)	`handle_event_report_bcsm/1`
ApplyChargingReport	71	Charging report from gsmSSF	`handle_apply_charging_report/1`
AssistRequestInstructions	16	Request for assistance from gsmSRF	`handle_assist_request_instructions/1`

Outgoing Operations (from gsmSCF → gsmSSF)

Operation	Opcode	Description	Generator
Connect	20	Connect call to destination number	`CapRequestGenerator.connect_request/2`
Continue	31	Continue call processing without modification	`CapRequestGenerator.continue_request/1`
ReleaseCall	22	Release/terminate the call	`CapRequestGenerator.release_call_request/2`
RequestReportBCSMEvent	23	Request notification of call events	`CapRequestGenerator.request_report_bcsm_event_request/2`
ApplyCharging	35	Apply charging to the call	`CapRequestGenerator.apply_charging_request/3`

Web UI Features

CAMEL Sessions Page

URL: `http://localhost/camel_sessions`

Real-time monitoring of active CAMEL call sessions:

Features:

- **Live session list** - Auto-refreshes every 2 seconds
- **Session details** - OTID, Call ID, State, Duration
- **CAP Version** - Displays protocol version (CAP v1/v2/v3/v4) detected from InitialDP
- **Call information** - IMSI, A-number, B-number, Service Key
- **State tracking** - Initiated, Answered, Terminated
- **Duration timer** - Real-time call duration display

Table Columns:

- Call ID, State, Version, IMSI, Calling Number, Called Number, Service Key, Duration, Start Time, OTID

Session States:

- ◇ **Initiated** - InitialDP received, waiting for answer
- ◇ **Answered** - Call answered, charging in progress
- ◇ **Terminated** - Call ended, CDR generated

CAP Version Detection: The system automatically detects the CAP protocol version from the InitialDP dialogue portion and displays it in the Version column. This helps identify which CAP version each MSC is using.

CAMEL Request Builder

URL: `http://localhost/camel_request`

Interactive tool for building and sending CAP requests:

Features:

- **Request type selector** - InitialDP, Connect, ReleaseCall, etc.
- **Dynamic form fields** - Adapts to selected request type
- **SCCP/M3UA options** - Advanced addressing configuration
- **Request history** - Last 20 requests with status
- **Session tracking** - Maintains OTID for follow-up requests
- **Real-time feedback** - Success/error messages

Request Types:

1. **InitialDP** - Start new call session
   - Service Key (integer)
   - Calling Number (A-party)
   - Called Number (B-party)
2. **Connect** - Route call to destination
   - Destination Number
3. **ReleaseCall** - Terminate call
   - Cause Code (16=Normal, 17=Busy, 31=Unspecified)
4. **RequestReportBCSMEvent** - Request event notifications
   - Events: oAnswer, oDisconnect, tAnswer, tDisconnect
5. **Continue** - Continue call without modification
   - No parameters required
6. **ApplyCharging** - Apply call duration limits
   - Duration (seconds, 1-864000)
   - Release on Timeout (boolean)
   - See CAMEL Request Builder Guide for detailed usage

Advanced SCCP Options:

- Called Party Global Title
- Calling Party Global Title
- Called SSN (default: 146 = gsmSSF)
- Calling SSN (default: 146)

M3UA Options:

- OPC (Originating Point Code, default: 5013)
- DPC (Destination Point Code, default: 5011)

Integration with OCS

Call Lifecycle with Charging

1. Call Initiation (InitialDP)

When MSC sends InitialDP, CAMELGW:

1. **Detects CAP version** - Examines dialogue portion to identify CAP v1/v2/v3/v4
2. **Decodes CAP message** - Extracts IMSI, calling/called numbers
3. **Calls OCS** - `InitiateSession` API
4. **Receives authorization** - MaxUsage (e.g., 30 seconds)
5. **Stores session** - In SessionStore (ETS table) with CAP version
6. **Responds to MSC** - RequestReportBCSMEvent + Continue (using same CAP version)

Example:

```
# Decoded InitialDP data
%{
  imsi: "310150123456789",
  calling_party_number: "14155551234",
  called_party_number: "14155556789",
  service_key: 1,
  msc_address: "19216800123",
  cap_version: :v2  # Detected from dialogue
}

# OCS response
{:ok, %{max_usage: 30}}  # 30 seconds authorized

# SessionStore entry
```

```
%{
  call_id: "CAMEL-4B000173",
  initial_dp_data: %{...},
  cap_version: :v2,  # Stored for response generation
  start_time: 1730246400,
  state: :initiated
}
```

2. Call Answer (EventReportBCSM - oAnswer)

When call is answered:

1. **Receives oAnswer event** - From MSC
2. **Updates OCS** - `UpdateSession` with usage=0
3. **Starts debit loop** - OCS begins charging
4. **Updates session state** - `:answered` in SessionStore
5. **Continues call** - Sends Continue to MSC

3. Periodic Updates (Optional)

For long calls, request additional credit:

```
# Every 30 seconds
OCS.Client.update_session(call_id, %{}, current_usage)
```

If MaxUsage returns 0, subscriber has no credit → Send ReleaseCall

4. Call Termination (EventReportBCSM - oDisconnect)

When call ends:

1. **Receives oDisconnect event** - From MSC
2. **Calculates total duration** - From session start time
3. **Terminates OCS session** - `TerminateSession` API
4. **CDR generated** - By OCS with final cost
5. **Cleans up session** - Removes from SessionStore
6. **Sends ReleaseCall** - Confirms termination to MSC

CDR Analysis

CDRs are generated by your OCS and typically include:

CDR Fields from CAMEL:

- `Account` - IMSI or calling number
- `Destination` - Called party number
- `OriginID` - Unique call identifier (CAMEL-OTID)
- `Usage` - Total call duration (seconds)
- `Cost` - Calculated cost
- `IMSI` - Subscriber IMSI
- `CallingPartyNumber` - A-party
- `CalledPartyNumber` - B-party
- `MSCAddress` - Serving MSC point code
- `ServiceKey` - CAMEL service key

Testing

Manual Testing with Request Builder

1. **Navigate to Request Builder:**

   ```
   http://localhost/camel_request
   ```

2. **Send InitialDP:**
   - Select "InitialDP" from dropdown
   - Service Key: `100`
   - Calling Number: `14155551234`
   - Called Number: `14155556789`
   - Click "Send InitialDP Request"
   - Note the OTID generated

3. **Monitor Session:**
   - Open new tab: `http://localhost/camel_sessions`
   - See active session with state "Initiated"

4. **Simulate Call Answer:**
   - Return to Request Builder
   - Select "EventReportBCSM"
   - Event Type: `oAnswer`
   - Click "Send EventReportBCSM Request"
   - Session state changes to "Answered"

5. **End Call:**
   - Select "ReleaseCall"
   - Cause Code: 16 (Normal)
   - Click "Send ReleaseCall Request"
   - Session state changes to "Terminated"

Testing with Real MSC

Configure MSC CAMEL Service

On your MSC/VLR, configure CAMEL service:

```
# Example Huawei MSC configuration
ADD CAMELSERVICE:
  SERVICEID=1,
  SERVICEKEY=100,
  GSMSCFADDR="55512341234",  # CAMELGW Global Title
  DEFAULTCALLHANDLING=CONTINUE;

ADD CAMELSUBSCRIBER:
  IMSI="310150123456789",
  SERVICEID=1,
  TRIGGERTYPE=TERMCALL;
```

Monitor Logs

Watch CAMELGW logs for incoming CAP messages:

```
# View logs in real-time
tail -f /var/log/omniss7/omniss7.log

# Filter for CAP events
grep "CAP:" /var/log/omniss7/omniss7.log

# View event log (JSON formatted)
curl http://localhost/api/events | jq '.[] | select(.map_event | startswith("CAP:"))'
```

Load Testing

Use the Request Builder in a loop for load testing:

```
# Send 100 InitialDP requests
for i in {1..100}; do
  curl -X POST http://localhost/api/camel/initial_dp \
    -H "Content-Type: application/json" \
    -d '{
      "service_key": 100,
      "calling_number": "1415555'$i'",
      "called_number": "14155556789"
    }'
  sleep 0.1
done
```

Monitoring & Operations

Prometheus Metrics

CAMELGW exposes metrics at `http://localhost:8080/metrics`:

CAP-specific metrics:

- `cap_requests_total{operation}` - Total CAP requests by operation type (e.g., initialDP, requestReportBCSMEvent)

Additional MAP/API metrics:

- `map_requests_total{operation}` - Total MAP requests by operation type
- `map_request_duration_milliseconds{operation}` - Request duration histogram
- `map_pending_requests` - Number of pending MAP transactions

M3UA STP metrics (if STP mode enabled):

- `m3ua_stp_messages_received_total{peer_name,point_code}` - Messages received from peers
- `m3ua_stp_messages_sent_total{peer_name,point_code}` - Messages sent to peers
- `m3ua_stp_routing_failures_total{reason}` - Routing failures by reason

Example queries:

```
# CAP requests
curl http://localhost:8080/metrics | grep cap_requests_total

# Total InitialDP received
curl http://localhost:8080/metrics | grep 'cap_requests_total{operation="initialDP"}'

# MAP pending requests
curl http://localhost:8080/metrics | grep map_pending_requests
```

Health Checks

```
# Check M3UA connectivity
curl http://localhost/api/m3ua-status

# Check OCS connectivity
curl http://localhost/api/ocs-status

# Check active sessions
curl http://localhost/api/camel/sessions/count
```

Logging Configuration

Adjust log level in `config/runtime.exs`:

```
config :logger,
  level: :info  # Options: :debug, :info, :warning, :error

# Enable CAP debug logging
config :logger, :console,
  metadata: [:cap_operation, :otid, :call_id]
```

Troubleshooting

Issue: No CAP messages received

Symptoms: Request Builder works, but MSC doesn't send InitialDP

Check:

1. M3UA link status: `curl http://localhost/api/m3ua-status`
2. MSC CAMEL service configuration (Service Key, gsmSCF address)
3. SCCP routing (Global Title must route to CAMELGW)
4. Firewall rules (allow SCTP port 2905)

Solution:

```
# Verify M3UA connectivity
tcpdump -i eth0 sctp

# Check if MSC can reach CAMELGW
ss -tuln | grep 2905
```

Issue: OCS errors

Symptoms: `INSUFFICIENT_CREDIT` or timeout errors

Check:

1. OCS is reachable: `curl http://your-ocs-server/api/health`
2. Account has balance in OCS
3. Rating plan configured in OCS
4. Network connectivity to OCS
5. Authentication token is valid (if required)

Solution:

- Verify OCS URL configuration in `runtime.exs`
- Check OCS logs for errors
- Test OCS API manually with curl
- Verify firewall rules allow connectivity

Issue: Session not found

Symptoms: EventReportBCSM fails with "Session not found"

Cause: OTID mismatch or session expired

Solution:

1. Verify OTID in logs
2. Check session timeout (default: no expiration)
3. Ensure DTID matches OTID in Continue/End messages

```
# Check active sessions
iex> CAMELGW.SessionStore.list_sessions()
```

Issue: Decode errors

Symptoms: `Failed to decode InitialDP` in logs

Cause: CAP version mismatch or malformed message

Solution:

1. Check CAP version configuration matches MSC
2. Verify ASN.1 encoding is correct
3. Capture PCAP and analyze with Wireshark

```
# Capture CAP messages
tcpdump -i eth0 -w cap_trace.pcap sctp port 2905

# Analyze with Wireshark (filter: m3ua)
wireshark cap_trace.pcap
```

Advanced Configuration

Multiple CAP Versions

Support different CAP versions per service key:

```
config :omniss7,
  cap_version_map: %{
    100 => :v2,  # Service Key 100 uses CAP v2
    200 => :v3,  # Service Key 200 uses CAP v3
    300 => :v4   # Service Key 300 uses CAP v4
  },
  cap_version: :v2  # Default
```

Summary

The CAMEL Gateway mode enables OmniSS7 to function as a complete Intelligent Network platform with:

◈ **Full CAP protocol support** (v1/v2/v3/v4) ◈ **Real-time charging** via OCS integration ◈ **Call control operations** (Connect, Release, Continue) ◈ **Session management** with ETS storage ◈ **Interactive testing** via Web UI Request Builder ◈ **Live monitoring** of active call sessions ◈ **CDR generation** for billing and analytics ◈ **Production-ready** performance and reliability

For additional information:

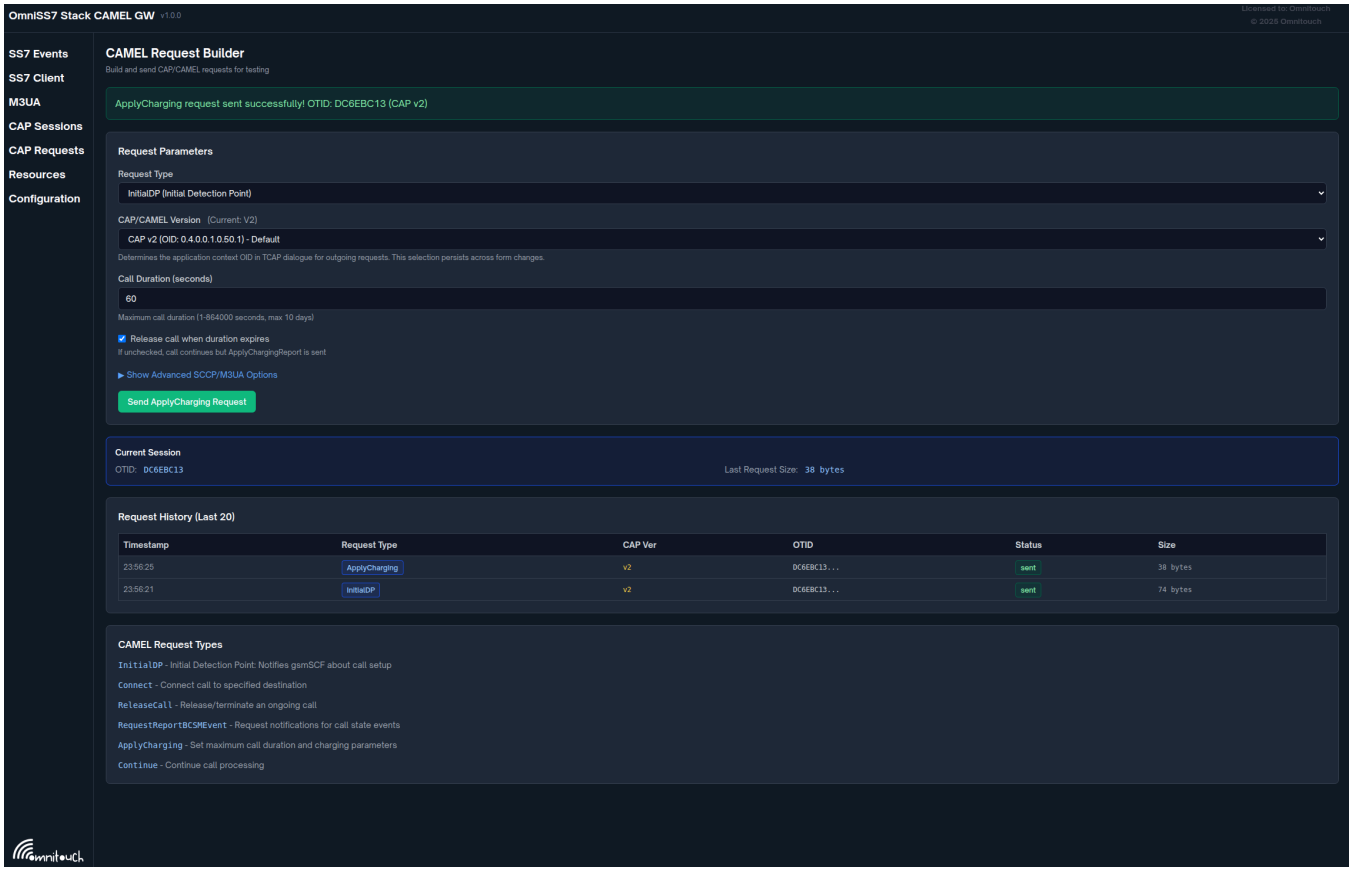
- CAMEL Request Builder Documentation
- Technical Reference - CAP Operations

Product: OmniSS7 CAMEL Gateway **Documentation Version:** 1.0 **Last Updated:** 2025-10-26

CAMEL Request Builder - Implementation Summary

Overview

A new LiveView component has been created to build and send CAMEL/CAP requests for testing purposes. This provides an interactive UI for creating InitialDP and other CAMEL operations.



New Components

1. CAMEL Request Builder LiveView

Features:

- Interactive form-based UI for building CAMEL requests
- Support for multiple request types:
  - **InitialDP** - Initial Detection Point (call setup notification)
  - **Connect** - Connect call to destination
  - **ReleaseCall** - Release/terminate call
  - **RequestReportBCSMEvent** - Request event notifications
  - **Continue** - Continue call processing
  - **ApplyCharging** - Apply charging/duration limits to calls

Key Capabilities:

- Request type selection dropdown
- Dynamic form fields based on selected request type
- Advanced SCCP/M3UA options (collapsible section)
  - Called/Calling Party Global Titles
  - SSN (Subsystem Number) configuration
  - OPC/DPC (Point Code) settings
- Real-time request history (last 20 requests)
- Session tracking via OTID
- Success/error feedback
- Request size tracking

Route: `/camel_request`

2. Enhanced EventLog with CAMEL Support

New Functions:

- `paklog_camel/2` - Dedicated CAMEL/CAP message logging
- `lookup_cap_opcode_name/1` - CAP operation code lookup
- `find_cap_opcode/1` - Extract CAP opcode from JSON
- `extract_cap_tids/1` - Extract OTID/DTID from CAP messages
- `format_cap_to_json/1` - Convert CAP PDUs to JSON format

CAP Operation Codes Supported:

```
0  => "initialDP"
5  => "connect"
6  => "releaseCall"
7  => "requestReportBCSMEvent"
8  => "eventReportBCSM"
10 => "continue"
13 => "furnishChargingInformation"
35 => "applyCharging"
... (47 total operations)
```

Features:

- JSON logging of all CAMEL requests/responses
- Automatic TCAP action detection (Begin/Continue/End/Abort)
- SCCP addressing extraction
- Error handling for malformed messages
- Background task processing (non-blocking)
- Event prefixed with "CAP:" for easy filtering

3. Updated CapClient

Changes:

- Added `paklog_camel/2` calls for incoming and outgoing messages
- Dual logging: Both MAP (`paklog`) and CAP (`paklog_camel`) for compatibility
- Outgoing messages logged in `sccp_m3ua_maker/2`
- Incoming messages logged in `handle_payload/1`

Configuration

The new LiveView pages have been added to the runtime configuration:

```
# File: config/runtime.exs

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.TestClientLive, "/client", "SS7 Client"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.HlrLinksLive, "/hlr_links", "HLR Links"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAMEL Sessions"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAMEL Request Builder"}
  ],
  page_order: ["/events", "/client", "/m3ua", "/hlr_links",
               "/camel_sessions", "/camel_request",
               "/application", "/configuration"]
```

Usage

Accessing the Request Builder

1. Navigate to: `https://your-server:8087/camel_request`
2. Select request type from dropdown
3. Fill in required parameters
4. Optionally expand "Advanced SCCP/M3UA Options" for fine-tuning
5. Click "Send [RequestType] Request"

Request Flow

InitialDP (New Call)

1. Set Service Key (e.g., 100)
2. Set Calling Number (A-Party)
3. Set Called Number (B-Party)
4. Send request → Generates new OTID
5. OTID stored in session for follow-up requests

Follow-up Requests (Connect, ReleaseCall, etc.)

1. Must have active OTID from InitialDP
2. Request automatically uses stored OTID
3. Warning shown if no active OTID

Request Parameters

InitialDP:

- Service Key (integer)
- Calling Number (ISDN format)
- Called Number (ISDN format)

Connect:

- Destination Number (where to route call)

ReleaseCall:

- Cause Code (16 = Normal, 17 = Busy, 31 = Unspecified)

RequestReportBCSMEvent:

- BCSM Events (comma-separated: oAnswer, oDisconnect, etc.)

Continue:

- No parameters (uses active OTID)

ApplyCharging:

- Duration (seconds, 1-864000) - Maximum call duration before action
- Release on Timeout (boolean) - Whether to release call when duration expires

Advanced Options

SCCP Addressing:

- Called Party GT (Global Title)
- Calling Party GT
- Called SSN (default 146 = gsmSSF)
- Calling SSN (default 146)

M3UA Point Codes:

- OPC (Originating Point Code, default 5013)
- DPC (Destination Point Code, default 5011)

JSON Logging

All CAMEL messages are now logged in JSON format in the event log with:

- **Direction:** incoming/outgoing
- **TCAP Action:** Begin/Continue/End/Abort
- **CAP Operation:** e.g., "CAP:initialDP", "CAP:connect"
- **SCCP Addressing:** Called/Calling Party info
- **TIDs:** OTID/DTID for correlation
- **Full Message:** JSON-encoded CAP PDU

Example Log Entry

```
{
  "map_event": "CAP:initialDP",
  "direction": "outgoing",
  "tcap_action": "Begin",
  "otid": "A1B2C3D4",
  "sccp_called": {
    "SSN": 146,
    "GlobalTitle": {
      "Digits": "55512341234",
      "NumberingPlan": "isdn_tele",
      "NatureOfAddress_Indicator": "international"
    }
  },
  "event_message": "{ ... full CAP PDU ... }"
}
```

Request History

The UI displays the last 20 requests with:

- Timestamp
- Request type (with color-coded badge)
- OTID (first 8 hex chars)
- Status (sent/error)
- Message size in bytes

Session Tracking

Current Session Info Panel:

- Displays active OTID
- Shows last request byte size
- Visible only when session is active

Testing Workflow

1. **Start New Call:**
   - Send InitialDP → Get OTID
   - System creates session
2. **Control Call:**
   - Send RequestReportBCSMEvent → Request notifications
   - Send ApplyCharging → Set call duration limit (e.g., 290 seconds)
   - Send Connect → Route to destination
   - OR Send ReleaseCall → Terminate
3. **View Results:**
   - Check request history
   - Monitor CAMEL Sessions page
   - Review event logs with "CAP:" prefix

ApplyCharging - Call Duration Control

Overview

The ApplyCharging operation allows you to set a maximum call duration and optionally release the call when that duration expires. This is typically used for prepaid charging scenarios or enforcing time limits on calls.

Use Cases

- **Prepaid Charging**: Limit call duration based on subscriber balance
- **Time-Based Billing**: Enforce periodic charging intervals

- **Resource Management**: Prevent calls from running indefinitely
- **OCS Integration**: Coordinate with Online Charging Systems for real-time credit control

Parameters

Duration (maxCallPeriodDuration)

- **Type**: Integer (1-864000 seconds)
- **Description**: Maximum number of seconds the call can run before the timer expires
- **Examples**:
  - `60` = 1 minute
  - `290` = 4 minutes 50 seconds (common test value)
  - `3600` = 1 hour
  - `86400` = 24 hours

Release on Timeout (releaseIfDurationExceeded)

- **Type**: Boolean (true/false)
- **Default**: true
- **Description**: What happens when the duration expires:
  - `true`: Automatically release/disconnect the call
  - `false`: Send notification but keep call active (allows gsmSCF to take action)

Message Structure

The ApplyCharging message is encoded as a TCAP Continue with:

- **TCAP**: Continue message (uses existing transaction)
- **Opcode**: 35 (applyCharging)
- **Parameters**: ApplyChargingArg containing:
  - `aChBillingChargingCharacteristics`: Time-based charging info
    - `timeDurationCharging`: Maximum duration and release flag
  - `partyToCharge`: Which party is charged (default: sendingSideID)

Example Usage

Scenario: Prepaid call with 5-minute limit

1. Send **InitialDP** to start call monitoring

   ```
   Service Key: 100
   Calling: 447700900123
   Called: 447700900456
   → OTID: A1B2C3D4
   ```

2. Send **ApplyCharging** to set 5-minute limit

   ```
   Duration: 300 (seconds)
   Release on Timeout: true
   → Uses OTID: A1B2C3D4
   ```

3. Send **Connect** to complete the call

   ```
   Destination: 447700900456
   → Uses OTID: A1B2C3D4
   ```

4. After 5 minutes (300 seconds):
   - Call automatically released by network
   - gsmSCF receives disconnect notification

Best Practices

1. **Always send ApplyCharging BEFORE Connect**
   - Ensures charging is active when call connects
   - Prevents uncharged call segments
2. **Use with RequestReportBCSMEvent**
   - Request `oAnswer` and `oDisconnect` events
   - Allows tracking of actual call duration
   - Enables re-application of charging if needed
3. **Set reasonable durations**
   - Too short: Frequent charging operations, poor user experience
   - Too long: Risk of revenue loss on prepaid calls
   - Typical: 60-300 seconds for prepaid, longer for postpaid
4. **Handle timeout gracefully**
   - If `release=false`, be prepared to handle timer expiry notifications
   - Implement logic to extend duration or release call

Error Handling

Common issues:

- **No active OTID**: Must send InitialDP first
- **Invalid duration**: Must be 1-864000 seconds
- **Network support**: Some SSF implementations may not support ApplyCharging
- **Timer accuracy**: Network timer resolution typically 1 second, but may vary

Monitoring

Track ApplyCharging operations via:

- **Request History**: Shows sent ApplyCharging requests
- **Event Log**: Search for "CAP:applyCharging"
- **CAMEL Sessions**: Monitor active sessions with charging applied
- **TCAP Trace**: Debug encoding/decoding issues

Implementation Details

State Management

- LiveView assigns track form state
- OTID stored in socket assigns
- Request history limited to 20 entries
- Auto-refresh disabled (manual send only)

Request Generation

- Uses existing `CapRequestGenerator` module
- Builds proper TCAP/CAP structures
- Encodes with `:TCAPMessages` codec
- Wraps in SCCP via `CapClient.sccp_m3ua_maker/2`

Sending Mechanism

- Sends via M3UA to `:camelgw_client_asp`
- Uses routing context 1
- Automatic SCCP/M3UA encapsulation

Error Handling

- Form validation with user feedback
- Graceful handling of missing OTID
- Parse errors shown in UI
- Encoding failures logged

Future Enhancements

Potential additions:

1. Request templates/presets
2. Response correlation and display
3. Call flow visualization
4. Session detail drill-down
5. Export request history
6. Load testing (bulk requests)
7. PCAP export of generated messages
8. CAP parameter validation

Integration Notes

- Compatible with existing MAP logging (`paklog`)
- Shares event log database with MAP events
- Uses same SCCP/M3UA infrastructure
- Works with CAMELSessionsLive for monitoring
- Integrates with existing M3UA routing

Files Modified

- `config/runtime.exs` - UPDATED

Dependencies

- Existing CapRequestGenerator
- CapClient for M3UA sending
- M3UA.Server for packet transmission

- EventLog for message logging
- Phoenix LiveView framework
- Control Panel for UI infrastructure

Common Features Guide

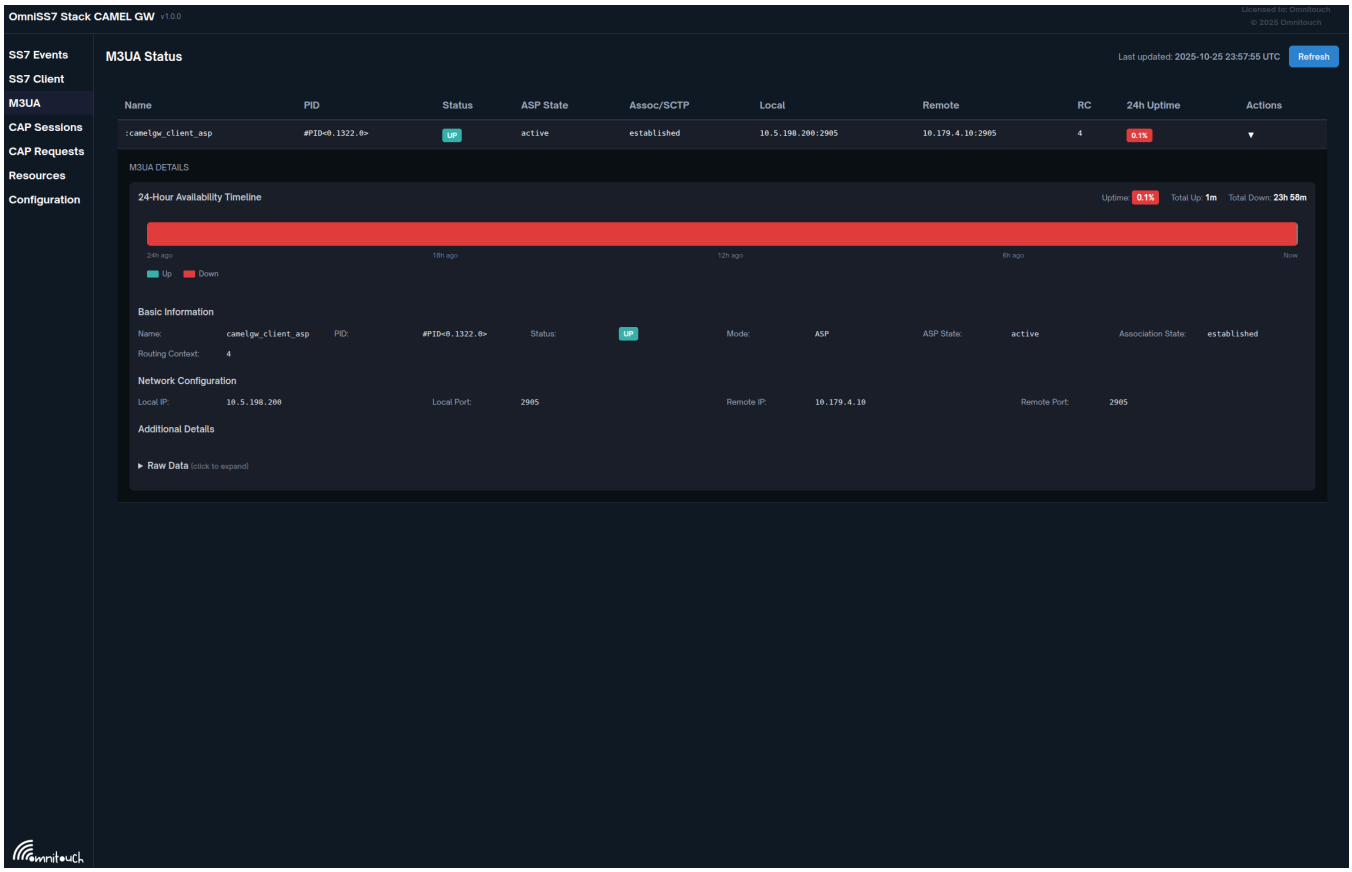
← Back to Main Documentation

This guide covers features common to all OmniSS7 operating modes.

Table of Contents

Web UI Overview

The Web UI is accessible via your configured web server address.



Main Navigation

- **Events** - Real-time SS7 signaling events and message logs
- **Application** - Application status and runtime information
- **Configuration** - System configuration viewer
- **M3UA Status** - M3UA peer connections (STP mode)
- **SMS Queue** - Outgoing SMS messages (SMSc mode)

Accessing the Web UI

1. Open your web browser
2. Navigate to configured hostname (e.g., `http://localhost`)
3. View system status dashboard

Swagger API Documentation

Interactive API documentation:

```
http://your-server/swagger
```

Web UI Configuration

Configure in `config/runtime.exs`:

```
config :control_panel,
  # Page order in navigation menu
  page_order: ["/events", "/application", "/configuration"],

  # Web server settings
  web: %{
    listen_ip: "0.0.0.0",    # IP to bind (0.0.0.0 for all interfaces)
    port: 80,                 # HTTP port (443 for HTTPS)
    hostname: "localhost",    # Server hostname for URL generation
    enable_tls: false,        # Set true to enable HTTPS
    tls_cert: "cert.pem",    # Path to TLS certificate file
    tls_key: "key.pem"       # Path to TLS private key file
  }
```

Configuration Parameters:

Parameter	Type	Default	Description
`page_order`	List	`["/events", "/application", "/configuration"]`	Order of pages in navigation menu
`listen_ip`	String	`"0.0.0.0"`	IP address to bind web server
`port`	Integer	`80`	HTTP port (use 443 for HTTPS)
`hostname`	String	`"localhost"`	Server hostname for URL generation
`enable_tls`	Boolean	`false`	Enable HTTPS with TLS
`tls_cert`	String	`"cert.pem"`	Path to TLS certificate (when TLS enabled)
`tls_key`	String	`"key.pem"`	Path to TLS private key (when TLS enabled)

Logger Configuration

Configure logging level in `config/runtime.exs`:

```
config :logger,
  level: :debug  # Options: :debug, :info, :warning, :error
```

Log Levels:

- `:debug` - Detailed debugging information
- `:info` - General informational messages
- `:warning` - Warning messages for potential issues

- `:error` - Error messages only

API Documentation

API Base URL

```
http://your-server/api
```

Response Codes

- **200** - Success
- **400** - Bad Request
- **504** - Gateway Timeout

OpenAPI Specification

```
http://your-server/swagger.json
```

Monitoring and Metrics

Prometheus Metrics Endpoint

```
http://your-server/metrics
```

Key Metrics Categories

M3UA/SCTP Metrics:

- SCTP association state changes
- M3UA ASP state transitions
- Protocol data units sent/received

M2PA Metrics:

- Link state transitions (DOWN → ALIGNMENT → PROVING → READY)
- Messages and bytes sent/received per link
- Link-specific errors (decode, encode, SCTP)

STP Metrics:

- Messages received/sent per peer
- Routing failures by reason
- Traffic distribution across peers

MAP Client Metrics:

- MAP requests by operation type
- Request duration histograms
- Pending transactions gauge

CAP Metrics:

- CAP requests by operation type
- CAMEL gateway operations

SMSc Metrics:

- Queue depth
- Delivery rates
- Failed messages

Grafana Integration

OmniSS7 metrics are compatible with Prometheus and Grafana.

Best Practices

Security Recommendations

1. **Network Isolation**
   - Deploy in dedicated VLAN
   - Firewall rules to restrict access
   - Allow SCTP only from known addresses
2. **Web UI Security**
   - Enable TLS for production
   - Use reverse proxy with authentication
   - Restrict to management IPs
3. **API Security**
   - Implement rate limiting
   - Use API keys or OAuth
   - Log all requests for audit

Performance Tuning

1. **TPS Limits**
   - Configure appropriate TPS
   - Monitor system load
   - Adjust SCTP buffers
2. **Database Optimization**
   - Add indexes
   - Archive old messages
   - Monitor connection pool
3. **M3UA Tuning**
   - Adjust SCTP heartbeat intervals
   - Configure timeout values
   - Use multiple links for redundancy

Monitoring and Alerting

Key Metrics:

- M3UA connection state
- MAP request success rate
- API response times
- Message queue depth

Alert Thresholds:

- M3UA down > 1 minute
- MAP timeout rate > 10%
- Queue depth > 1000
- API error rate > 5%

Complete Configuration Reference

All Configuration Parameters

This section provides a complete reference of all available configuration parameters across all operating modes.

Logger Configuration (`:logger`)

```
config :logger,
  level: :debug  # :debug | :info | :warning | :error
```

Web UI Configuration (`:control_panel`)

```
config :control_panel,
  page_order: ["/events", "/application", "/configuration"],
  web: %{
    listen_ip: "0.0.0.0",
    port: 80,
    hostname: "localhost",
    enable_tls: false,
    tls_cert: "cert.pem",
    tls_key: "key.pem"
  }
```

Parameter	Type	Required	Default	Description
`page_order`	List of Strings	No	`["/events", "/application", "/configuration"]`	Navigation menu page order
`web.listen_ip`	String	Yes	`"0.0.0.0"`	IP address to bind web server
`web.port`	Integer	Yes	`80`	HTTP/HTTPS port number
`web.hostname`	String	Yes	`"localhost"`	Server hostname
`web.enable_tls`	Boolean	No	`false`	Enable HTTPS
`web.tls_cert`	String	If TLS enabled	`"cert.pem"`	TLS certificate path
`web.tls_key`	String	If TLS enabled	`"key.pem"`	TLS private key path

M3UA STP Configuration (`:omniss7`)

```
config :omniss7,
  m3ua_stp: %{
    enabled: false,
    local_ip: {127, 0, 0, 1},
    local_port: 2905
  },
  enable_gt_routing: true,
  m3ua_peers: [...],
  m3ua_routes: [...],
  m3ua_gt_routes: [...]
```

Parameter	Type	Required	Default	Description
`m3ua_stp.enabled`	Boolean	Yes	`false`	Enable STP mode at boot
`m3ua_stp.local_ip`	Tuple	Yes	`{127, 0, 0, 1}`	IP to bind for incoming M3UA
`m3ua_stp.local_port`	Integer	Yes	`2905`	SCTP port for M3UA
`enable_gt_routing`	Boolean	No	`false`	Enable Global Title routing

M3UA Peer Parameters:

Parameter	Type	Required	Description
`peer_id`	Integer	Yes	Unique peer identifier
`name`	String	Yes	Descriptive peer name
`role`	Atom	Yes	`:client` or `:server`
`local_ip`	Tuple	If `:client`	Local IP to bind
`local_port`	Integer	If `:client`	Local port (0 for dynamic)
`remote_ip`	Tuple	Yes	Remote peer IP
`remote_port`	Integer	If `:client`	Remote peer port
`routing_context`	Integer	Yes	M3UA routing context
`point_code`	Integer	Yes	SS7 point code
`network_indicator`	Atom	No	`:international` or `:national`

M3UA Route Parameters:

Parameter	Type	Required	Description
`dest_pc`	Integer	Yes	Destination point code
`peer_id`	Integer	Yes	Peer to route through
`priority`	Integer	Yes	Route priority (lower = higher priority)
`network_indicator`	Atom	No	`:international` or `:national`

M3UA GT Route Parameters:

Parameter	Type	Required	Description
`gt_prefix`	String	Yes	Global Title prefix to match
`peer_id`	Integer	Yes	Destination peer
`priority`	Integer	Yes	Route priority
`description`	String	No	Route description for logging
`source_ssn`	Integer	No	Match only if source SSN matches
`dest_ssn`	Integer	No	Rewrite destination SSN to this value

MAP Client Configuration (`:omniss7`)

```
config :omniss7,
  map_client_enabled: false,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

Parameter	Type	Required	Default	Description
`map_client_enabled`	Boolean	Yes	`false`	Enable MAP client mode
`map_client_m3ua.mode`	String	Yes	`"ASP"`	M3UA connection mode (`"ASP"` or `"SGP"`)
`map_client_m3ua.callback`	Tuple	Yes	`{MapClient, :handle_payload, []}`	Message callback handler
`map_client_m3ua.process_name`	Atom	Yes	`:map_client_asp`	Registered process name
`map_client_m3ua.local_ip`	Tuple	Yes	-	Local IP address
`map_client_m3ua.local_port`	Integer	Yes	`2905`	Local SCTP port
`map_client_m3ua.remote_ip`	Tuple	Yes	-	Remote STP/SGP IP
`map_client_m3ua.remote_port`	Integer	Yes	`2905`	Remote SCTP port
`map_client_m3ua.routing_context`	Integer	Yes	-	M3UA routing context

SMS Center Configuration (`:omniss7`)

```
config :omniss7,
  auto_flush_enabled: false,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: nil,
  auto_flush_tps: 10
```

Parameter	Type	Required	Default	Description
`auto_flush_enabled`	Boolean	No	`false`	Enable auto-flush of SMS queue
`auto_flush_interval`	Integer	No	`10000`	Queue poll interval (milliseconds)
`auto_flush_dest_smsc`	String/nil	No	`nil`	Filter by dest SMSC (nil = all)
`auto_flush_tps`	Integer	No	`10`	Max transactions per second

HTTP API Configuration (`:omniss7`)

The SMS backend now uses HTTP API instead of direct database connections.

```
config :omniss7,
  smsc_api_base_url: "https://10.5.198.200:8443",
  frontend_name: "omni-smsc01"  # Optional: defaults to hostname_SMSc
```

API Parameters:

Parameter	Type	Required	Default	Description
`smsc_api_base_url`	String	Yes	`"https://10.5.198.200:8443"`	Base URL for SMS backend API
`frontend_name`	String	No	`"{hostname}_SMSc"`	Frontend identifier for registration

API Endpoints Used:

- `POST /api/frontends` - Register this frontend instance with backend
- `POST /api/messages_raw` - Insert new SMS messages
- `GET /api/messages` - Retrieve message queue (with `smsc` header)
- `PATCH /api/messages/{id}` - Mark message as delivered
- `PUT /api/messages/{id}` - Update message status
- `POST /api/events` - Add event tracking
- `GET /api/status` - Health check endpoint

Frontend Registration:

The system automatically registers itself with the backend API on startup and re-registers every 5 minutes. Registration includes:

- Frontend name and type (SMSc)
- Hostname
- Uptime in seconds
- Configuration details (JSON format)

Configuration Notes:

- SSL verification is disabled by default for self-signed certificates
- HTTP requests timeout after 5 seconds
- All timestamps are in ISO 8601 format
- The API uses JSON for request/response bodies

Related Documentation

- ← Back to Main Documentation
- STP Guide
- MAP Client Guide
- SMS Center Guide
- HLR Guide

Configuration Reference

← Back to Main Documentation

This document provides a comprehensive reference for all OmniSS7 configuration parameters.

Table of Contents

Overview

OmniSS7 configuration is managed via `config/runtime.exs`. The system supports three operational modes:

- **STP Mode** - Signal Transfer Point for routing
- **HLR Mode** - Home Location Register for subscriber management
- **SMSc Mode** - SMS Center for message delivery

Configuration File: `config/runtime.exs`

Operational Mode Flags

Control which features are enabled.

Parameter	Type	Default	Description	Modes
`map_client_enabled`	Boolean	`false`	Enable MAP client and M3UA connectivity	All
`hlr_mode_enabled`	Boolean	`false`	Enable HLR-specific features	HLR
`smsc_mode_enabled`	Boolean	`false`	Enable SMSc-specific features	SMSc

Example:

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false
```

HLR Mode Parameters

Configuration for HLR (Home Location Register) mode.

HLR API Configuration

Parameter	Type	Default	Required	Description
`hlr_api_base_url`	String	-	**Yes**	Backend HLR API endpoint URL (SSL verify hardcoded to disabled)
`hlr_service_center_gt_address`	String	-	**Yes**	HLR Global Title address returned in UpdateLocation responses
`smsc_service_center_gt_address`	String	-	**Yes**	SMSC GT address returned in SRI-for-SM responses

Example:

```
config :omniss7,
  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112"
```

MSISDN ↔ IMSI Mapping

Configuration for synthetic IMSI generation from MSISDNs. For detailed technical explanation of the mapping algorithm, see MSISDN ↔ IMSI Mapping in HLR Guide.

Parameter	Type	Default	Required	Description
`hlr_imsi_plmn_prefix`	String	`"50557"`	No	PLMN prefix (MCC+MNC) for synthetic IMSI generation
`hlr_msisdn_country_code`	String	`"61"`	No	Country code prefix for IMSI→MSISDN reverse mapping
`hlr_msisdn_nsn_offset`	Integer	`0`	No	Offset into MSISDN where NSN starts (typically length of country code)
`hlr_msisdn_nsn_length`	Integer	`9`	No	Length of National Subscriber Number to extract from MSISDN

Example (2-digit country code):

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "99",      # Example 2-digit country code
  hlr_msisdn_nsn_offset: 2,           # Skip 2-digit country code
  hlr_msisdn_nsn_length: 9            # Extract 9-digit NSN
```

Example (3-digit country code):

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "999",     # Example 3-digit country code
  hlr_msisdn_nsn_offset: 3,           # Skip 3-digit country code
  hlr_msisdn_nsn_length: 8            # Extract 8-digit NSN
```

Important: Set `nsn_offset` to the length of your country code to properly extract the NSN. For example:

- Country code "9" (1 digit) → `nsn_offset: 1`
- Country code "99" (2 digits) → `nsn_offset: 2`
- Country code "999" (3 digits) → `nsn_offset: 3`

InsertSubscriberData (ISD) Configuration

Configuration for subscriber provisioning data sent to VLRs during UpdateLocation. For detailed explanation of the ISD sequence and message flow, see InsertSubscriberData Configuration in HLR Guide.

Parameter	Type	Default	Required	Description
`isd_network_access_mode`	Atom	`:packetAndCircuit`	No	Network access type: `:packetAndCircuit`, `:packetOnly`, or `:circuitOnly`
`isd_send_ss_data`	Boolean	`true`	No	Send ISD #2 with Supplementary Services data
`isd_send_call_barring`	Boolean	`true`	No	Send ISD #3 with Call Barring data

Example:

```
config :omniss7,
  isd_network_access_mode: :packetAndCircuit,
  isd_send_ss_data: true,
  isd_send_call_barring: true
```

CAMEL Configuration

Configuration for CAMEL-based intelligent call routing. For detailed explanation of CAMEL integration and service keys, see CAMEL Integration in HLR Guide.

Parameter	Type	Default	Required	Description
`camel_service_key`	Integer	`11_110`	No	CAMEL service key for SRI responses
`camel_trigger_detection_point`	Atom	`:termAttemptAuthorized`	No	CAMEL trigger point: `:termAttemptAuthorized`, `:tBusy`, `:tNoAnswer`, `:tAnswer`
`camel_gsmscf_gt_address`	String	(uses called GT)	No	Default gsmSCF Global Title for CAMEL responses (can be overridden by GT NAT)

Example:

```
config :omniss7,
  camel_service_key: 11_110,
  camel_trigger_detection_point: :termAttemptAuthorized
```

Home VLR Prefixes

Configuration for distinguishing home vs roaming subscribers. For detailed explanation of home/roaming detection and PRN operations, see Roaming Subscriber Handling in HLR Guide.

Parameter	Type	Default	Required	Description
`home_vlr_prefixes`	List	`["5551231"]`	No	VLR GT prefixes considered "home" network

Example:

```
config :omniss7,
  home_vlr_prefixes: ["5551231", "5551234"]
```

SMSc Mode Parameters

Configuration for SMS Center mode.

SMSc API Configuration

Parameter	Type	Default	Required	Description
`smsc_api_base_url`	String	-	**Yes**	Backend SMSc API endpoint

Parameter	Type	Default	Required	Description
				URL (SSL verify hardcoded to disabled)
`smsc_name`	String	`"{hostname}_SMSc"`	No	SMSc identifier for backend registration
`smsc_service_center_gt_address`	String	-	**Yes**	Service Center Global Title address

Example:

```
config :omniss7,
  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112"
```

Note: Frontend registration occurs every **5 minutes** (hardcoded) via `SMS.FrontendRegistry` module.

Auto-Flush Configuration

Parameter	Type	Default	Required	Description
`auto_flush_enabled`	Boolean	`true`	No	Enable automatic SMS queue processing
`auto_flush_interval`	Integer	`10_000`	No	Queue processing interval in milliseconds
`auto_flush_dest_smsc`	String	-	**Yes**	Destination SMSC name for auto-flush
`auto_flush_tps`	Integer	`10`	No	Message processing rate (transactions/second)

Example:

```
config :omniss7,
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10
```

STP Mode Parameters

Configuration for M3UA Signal Transfer Point mode. For detailed routing configuration and examples, see the STP Configuration Guide.

Standalone STP Server

Parameter	Type	Default	Required	Description
`m3ua_stp.enabled`	Boolean	`false`	No	Enable standalone M3UA STP server
`m3ua_stp.local_ip`	Tuple	`{127, 0, 0, 1}`	No	IP address to listen for connections
`m3ua_stp.local_port`	Integer	`2905`	No	Port to listen on
`m3ua_stp.point_code`	Integer	-	**Yes** (if enabled)	This STP's own SS7 point code

Example:

```
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  }
```

Global Title Routing

Parameter	Type	Default	Required	Description
`enable_gt_routing`	Boolean	`false`	No	Enable GT routing in addition to PC routing

Example:

```
config :omniss7,
  enable_gt_routing: true
```

Global Title NAT Parameters

Global Title Network Address Translation allows different response GTs based on calling party prefix. For detailed explanation and examples, see the Global Title NAT Guide.

Parameter	Type	Default	Required	Description
`gt_nat_enabled`	Boolean	`false`	No	Enable/disable GT NAT feature
`gt_nat_rules`	List of Maps	`[]`	**Yes** (if enabled)	List of prefix-to-GT mappings

Rule Format: Each rule in `gt_nat_rules` must be a map with:

- `calling_prefix`: String prefix to match against calling GT
- `response_gt`: Global Title to use in responses

Example:

```
config :omniss7,
  gt_nat_enabled: true,
  gt_nat_rules: [
    # When called from GT starting with "8772", respond with "55512341112"
    %{calling_prefix: "8772", response_gt: "55512341112"},
    # When called from GT starting with "8773", respond with "55512341111"
    %{calling_prefix: "8773", response_gt: "55512341111"},
    # Default fallback (empty prefix matches all)
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

See Also: GT NAT Guide for detailed usage and examples.

M3UA Connection Parameters

M3UA connection configuration for MAP client mode. For detailed usage and examples, see the MAP Client Guide.

Parameter	Type	Default	Required	Description
`map_client_m3ua.mode`	String	-	**Yes**	Connection mode: `"ASP"` or `"SGP"`
`map_client_m3ua.callback`	Tuple	-	**Yes**	Callback module/function: `{MapClient, :handle_payload, []}`
`map_client_m3ua.process_name`	Atom	-	**Yes**	Process name for registration
`map_client_m3ua.local_ip`	Tuple	-	**Yes**	Local IP address to bind
`map_client_m3ua.local_port`	Integer	`2905`	**Yes**	Local SCTP port
`map_client_m3ua.remote_ip`	Tuple	-	**Yes**	Remote STP/SGW IP address
`map_client_m3ua.remote_port`	Integer	`2905`	**Yes**	Remote SCTP port
`map_client_m3ua.routing_context`	Integer	-	**Yes**	M3UA routing context ID

Example:

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

HTTP Server Parameters

Configuration for the REST API HTTP server.

Parameter	Type	Default	Required	Description
`start_http_server`	Boolean	`true`	No	Enable/disable HTTP server (port 8080)

Hardcoded Values (not configurable):

- **IP**: 0.0.0.0 (all interfaces)
- **Port**: 8080
- **Transport**: Plug.Cowboy

Example:

```
config :omniss7,
  start_http_server: true  # Set to false to disable
```

API Endpoints:

- REST API: `http://[server-ip]:8080/api/*`
- Swagger UI: `http://[server-ip]:8080/swagger`
- Prometheus metrics: `http://[server-ip]:8080/metrics`

See API Guide for details.

Database Parameters

Configuration for Mnesia database persistence.

Parameter	Type	Default	Required	Description
`mnesia_storage_type`	Atom	`:disc_copies`	No	Mnesia storage type: `:disc_copies` or `:ram_copies`

Example:

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # Production
  # mnesia_storage_type: :ram_copies  # Testing only
```

Storage Types:

- `:disc_copies` - Persistent disk storage (survives restarts) - **Recommended for production**
- `:ram_copies` - In-memory only (lost on restart) - For testing only

Mnesia Tables:

- `m3ua_peer` - M3UA peer connections

- `m3ua_route` - Point Code routes
- `m3ua_gt_route` - Global Title routes

Location: `Mnesia.{node_name}/` directory

Hardcoded Values

The following values are **hardcoded in the source code** and cannot be changed via configuration.

Timeouts

Value	Impact	Workaround
MAP request timeout: **10 seconds**	All MAP operations timeout after 10s	Modify source code
ISD timeout: **10 seconds**	Each ISD message times out after 10s	Modify source code

HTTP Server

Value	Impact	Workaround
HTTP IP: **0.0.0.0**	Server listens on all interfaces	Modify source code
HTTP Port: **8080**	REST API runs on port 8080	Modify source code

SSL Verification

Value	Impact	Workaround
HLR API SSL: **disabled**	SSL verification always disabled	Modify source code
SMSc API SSL: **disabled**	SSL verification always disabled	Modify source code

Registration Intervals

Value	Impact	Workaround
Frontend registration: **5 minutes**	SMSc registers with backend every 5 min	Modify source code

Web UI Auto-Refresh

Page	Interval
Routing Management	5 seconds
Active Subscribers	2 seconds

Configuration Examples

Minimal HLR Configuration

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112",

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
```

```
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

Minimal SMSc Configuration

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112",

  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

STP with Standalone Server

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  enable_gt_routing: true,
  mnesia_storage_type: :disc_copies,

  m3ua_stp: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  },

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 10},
    local_port: 2906,
    remote_ip: {10, 179, 4, 11},
    remote_port: 2905,
    routing_context: 1
  }
```

Summary

Total Configuration Parameters: 32

By Category:

- Operational Mode: 3 parameters
- HLR Mode: 13 parameters
- SMSc Mode: 7 parameters
- STP Mode: 5 parameters
- M3UA Connection: 8 parameters
- HTTP Server: 1 parameter
- Database: 1 parameter

Required Parameters (must be set):

- `hlr_api_base_url` (HLR mode)
- `hlr_service_center_gt_address` (HLR mode)
- `smsc_api_base_url` (SMSc mode)
- `smsc_service_center_gt_address` (SMSc/HLR mode)
- All `map_client_m3ua.*` parameters
- `m3ua_stp.point_code` (if STP enabled)

Related Documentation

- **HLR Guide** - HLR-specific configuration
- **SMSc Guide** - SMSc-specific configuration
- **STP Guide** - STP routing configuration
- **API Guide** - REST API reference
- **Web UI Guide** - Web interface documentation

Global Title NAT Guide

Overview

Global Title Network Address Translation (GT NAT) is a feature that allows OmniSS7 to respond with different Global Title addresses based on the calling party's GT prefix, the called party's GT prefix, or a combination of both. This is essential when operating with multiple Global Titles and needing to ensure responses use the correct GT based on which network or peer is calling and/or which GT they called.

What's New (Enhanced GT NAT)

The GT NAT feature has been enhanced with powerful new capabilities:

New Features

1. **Called Party Prefix Matching**: Rules can now match on `called_prefix` in addition to `calling_prefix`
2. **Combined Matching**: Rules can match on both calling AND called prefixes simultaneously
3. **Weight-Based Prioritization**: Rules now use a `weight` field (lower = higher priority) instead of just prefix length
4. **Flexible Matching**: You can now create rules with:
    - Only calling prefix
    - Only called prefix
    - Both calling and called prefixes
    - Neither (wildcard/fallback rule)

New Rule Format

Required fields:

- `weight`: Integer priority (lower = higher priority)
- `response_gt`: The GT to respond with

Optional fields (at least one recommended for specific matching):

- `calling_prefix`: Match on calling party GT prefix
- `called_prefix`: Match on called party GT prefix

Example:

```
gt_nat_rules: [
  # Specific rule with both prefixes - highest priority
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # Specific rules - medium priority
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},
  %{called_prefix: "555", weight: 10, response_gt: "333333"},

  # Wildcard fallback - lowest priority
  %{weight: 100, response_gt: "999999"}
]
```

Use Cases

Multi-Network Operation

When you have multiple peer networks and each expects responses from a specific GT:

- **Network A** calls your GT `111111` and expects responses from `111111`
- **Network B** calls your GT `222222` and expects responses from `222222`

Without GT NAT, you would need separate instances or complex routing. With GT NAT, a single OmniSS7 instance can handle this intelligently.

Roaming Scenarios

When operating as an HLR or SMSc with roaming agreements:

- **Home network** subscribers use GT `555000`
- **Roaming partner 1** uses GT `555001`
- **Roaming partner 2** uses GT `555002`

GT NAT ensures each partner receives responses from the correct GT they're configured to route to.

Testing and Migration

During network migrations or testing:

- Gradually migrate traffic from old GT to new GT
- Maintain both GTs during transition period
- Route responses based on which GT the caller used

How It Works

Address Translation Flow

1. **Incoming Request**: OmniSS7 receives an SCCP message with:
    - Called Party GT: `55512341112` (your GT)
    - Calling Party GT: `877234567` (their GT)
2. **GT NAT Lookup**: System checks calling GT `877234567` against configured prefix rules
3. **Prefix Matching**: Finds longest matching prefix (e.g., `8772` matches `877234567`)
4. **Response GT Selection**: Uses `response_gt` from matched rule (e.g., `55512341112`)
5. **Response Sent**: SCCP response uses:
    - Called Party GT: `877234567` (reversed - their GT)
    - Calling Party GT: `55512341112` (NAT'd GT)

Affected Response Types

GT NAT applies to multiple layers of the SS7 stack:

SCCP Layer (All Responses)

- SCCP Called/Calling GT addresses in all response messages
- ISD (InsertSubscriberData) acknowledgments
- UpdateLocation responses
- Error responses

MAP Layer (Operation-Specific)

- **SRI-for-SM Responses**: `networkNode-Number` (SMSc GT address)
- **UpdateLocation**: `hlr-Number` in responses
- **InsertSubscriberData**: HLR GT in ISD messages

Configuration

Basic Configuration

Add to `config/runtime.exs`:

```
config :omniss7,
  # Enable GT NAT
  gt_nat_enabled: true,

  # Define GT NAT rules
  gt_nat_rules: [
    # Rule 1: Calls from prefix "8772" get response from
"55512341112"
    %{calling_prefix: "8772", response_gt: "55512341112"},

    # Rule 2: Calls from prefix "8773" get response from
"55512341111"
    %{calling_prefix: "8773", response_gt: "55512341111"},

    # Default rule (empty prefix matches everything)
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

Configuration Parameters

For complete configuration reference, see Global Title NAT Parameters in Configuration Reference.

Parameter	Type	Required	Description
`gt_nat_enabled`	Boolean	Yes	Enable/disable GT NAT feature
`gt_nat_rules`	List of Maps	Yes (if enabled)	List of prefix matching rules

Rule Format

Each rule is a map with the following keys:

```
%{
  calling_prefix: "8772",      # (Optional) Prefix to match against
calling GT
  called_prefix: "555",        # (Optional) Prefix to match against
called GT
  weight: 10,                  # (Required) Priority value (lower =
higher priority)
  response_gt: "55512341112"   # (Required) GT to use in responses
}
```

Rule Fields:

- **`calling_prefix`** (Optional): String prefix to match against incoming calling GT
    - Matching is done by `String.starts_with?/2`
    - Empty string `""` or `nil` acts as wildcard (matches any calling GT)

    - Can be omitted to match any calling GT

- **`called_prefix`** (Optional): String prefix to match against incoming called GT

    - Matching is done by `String.starts_with?/2`
    - Empty string `""` or `nil` acts as wildcard (matches any called GT)
    - Can be omitted to match any called GT

- **`weight`** (Required): Integer priority value

    - Lower weight = higher priority (processed first)
    - Must be >= 0
    - Used as primary sorting criterion for matching rules

- **`response_gt`** (Required): The Global Title address to use in responses

    - Must be a valid E.164 number string
    - Should match one of your configured GTs

At least one of `calling_prefix` or `called_prefix` should be specified for specific routing. Both can be omitted for a wildcard/fallback rule.

Rule Matching Logic

Rules are evaluated by **weight first (ascending), then by combined prefix specificity**:

Matching Algorithm:

1. Filter rules where all specified prefixes match
    - If `calling_prefix` is set, it must match the calling GT
    - If `called_prefix` is set, it must match the called GT
    - If both are set, both must match
    - If neither is set, rule acts as a wildcard
2. Sort matching rules by:
    - **Primary**: Weight (ascending - lower values first)
    - **Secondary**: Combined prefix length (descending - longer = more specific)
3. Return the first matching rule

Examples:

```
# Example rules
gt_nat_rules: [
  # Weight 1: Highest priority - matches both prefixes
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},
```

```
  # Weight 10: Medium priority - specific rules
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  #
Calling only
  %{called_prefix: "555", weight: 10, response_gt: "333333"},    #
Called only

  # Weight 100: Lowest priority - wildcard fallback
  %{weight: 100, response_gt: "444444"}  # Matches everything
]

# Matching examples:
# Calling: "877234567", Called: "555123" -> "111111" (weight 1, both
match)
# Calling: "877234567", Called: "999999" -> "222222" (weight 10,
calling only)
# Calling: "999999999", Called: "555123" -> "333333" (weight 10,
called only)
# Calling: "999999999", Called: "888888" -> "444444" (weight 100,
wildcard)
```

Examples

Example 1: Two Network Partners

Scenario: You operate an SMSc with two network partners. Each expects responses from a different GT.

```
config :omniss7,
  gt_nat_enabled: true,

  # Default SMSc GT (used when GT NAT is disabled or no rule matches)
  smsc_service_center_gt_address: "5551000",

  # GT NAT rules for partners
  gt_nat_rules: [
    # Partner A (prefix 4412) expects responses from GT 5551001
    %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

    # Partner B (prefix 4413) expects responses from GT 5551002
    %{calling_prefix: "4413", weight: 10, response_gt: "5551002"},

    # Default: use standard SMSc GT (wildcard fallback)
    %{weight: 100, response_gt: "5551000"}
  ]
```

Traffic Flow:

```
Incoming SRI-for-SM from 44121234567:
```

```
  Called GT: 5551001 (your GT that Partner A uses)
  Calling GT: 44121234567 (Partner A's GT)

GT NAT Lookup:
  "44121234567" matches prefix "4412"
  Selected response_gt: "5551001"

Response SRI-for-SM to 44121234567:
  Called GT: 44121234567 (reversed)
  Calling GT: 5551001 (NAT'd)
  networkNode-Number: 5551001 (in MAP response)
```

Example 2: HLR with Regional GTs

Scenario: National HLR with different GTs per region.

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "555000",  # Default HLR GT

  gt_nat_rules: [
    # Northern region VLRs (prefix 5551)
    %{calling_prefix: "5551", weight: 10, response_gt: "555100"},

    # Southern region VLRs (prefix 5552)
    %{calling_prefix: "5552", weight: 10, response_gt: "555200"},

    # Western region VLRs (prefix 5553)
    %{calling_prefix: "5553", weight: 10, response_gt: "555300"},

    # Default for other regions (wildcard)
    %{weight: 100, response_gt: "555000"}
  ]
```

Example 3: Migration Scenario

Scenario: Migrating from old GT to new GT gradually.

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "123456789",  # Old GT (default)

  gt_nat_rules: [
    # Migrated networks (already updated their configs)
    %{calling_prefix: "555", weight: 10, response_gt: "987654321"},
# New GT
    %{calling_prefix: "666", weight: 10, response_gt: "987654321"},
# New GT
```

```
    # Everyone else still uses old GT (wildcard)
    %{weight: 100, response_gt: "123456789"}  # Old GT
  ]
```

Example 4: Called Party Prefix Matching (NEW)

Scenario: You have multiple GTs for different services, and want to respond with the correct GT based on which GT was called.

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # When they call your SMS GT (5551xxx), respond with that GT
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # When they call your Voice GT (5552xxx), respond with that GT
    %{called_prefix: "5552", weight: 10, response_gt: "555200"},

    # When they call your Data GT (5553xxx), respond with that GT
    %{called_prefix: "5553", weight: 10, response_gt: "555300"},

    # Default fallback
    %{weight: 100, response_gt: "555000"}
  ]
```

Traffic Flow:

```
Incoming request to Called GT: 555100 (your SMS GT)
Calling GT: 441234567 (any caller)

GT NAT Lookup:
  Called GT "555100" matches prefix "5551"
  Selected response_gt: "555100"

Response uses Calling GT: 555100 (matches what they called)
```

Example 5: Combined Calling + Called Prefix Matching (ADVANCED)

Scenario: Different partners call different GTs, and you want fine-grained control.

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # Partner A calling your SMS GT - highest priority (weight 1)
```

```
    %{calling_prefix: "4412", called_prefix: "5551", weight: 1,
response_gt: "555101"},

    # Partner B calling your SMS GT - highest priority (weight 1)
    %{calling_prefix: "4413", called_prefix: "5551", weight: 1,
response_gt: "555102"},

    # Anyone calling your SMS GT - medium priority (weight 10)
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # Partner A calling any GT - medium priority (weight 10)
    %{calling_prefix: "4412", weight: 10, response_gt: "555200"},

    # Default fallback - low priority (weight 100)
    %{weight: 100, response_gt: "555000"}
  ]
```

Matching Examples:

```
# Partner A calls SMS GT
Calling: "441234567", Called: "555100"
→ Matches weight 1 rule (both prefixes) → "555101"

# Partner A calls Voice GT
Calling: "441234567", Called: "555200"
→ Matches weight 10 rule (calling only) → "555200"

# Unknown caller calls SMS GT
Calling: "999999999", Called: "555100"
→ Matches weight 10 rule (called only) → "555100"

# Unknown caller calls Voice GT
Calling: "999999999", Called: "555200"
→ Matches weight 100 wildcard → "555000"
```

Operational Modes

GT NAT works across all OmniSS7 operational modes:

HLR Mode

GT NAT affects:

- UpdateLocation responses (HLR GT in response)
- InsertSubscriberData messages (HLR GT as calling party)
- SendAuthenticationInfo responses
- Cancel Location responses

For more information on HLR operations, see the HLR Configuration Guide.

Configuration:

```
config :omniss7,
  hlr_mode_enabled: true,
  hlr_service_center_gt_address: "5551234567",  # Default HLR GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "331", weight: 10, response_gt: "5551234568"},
# France
    %{calling_prefix: "44", weight: 10, response_gt: "5551234569"},
# UK
    %{weight: 100, response_gt: "5551234567"}  # Default wildcard
  ]
```

SMSc Mode

GT NAT affects:

- SRI-for-SM responses (`networkNode-Number` field) - see SRI-for-SM Details
- MT-ForwardSM acknowledgments

For more information on SMSc operations, see the SMSc Configuration Guide.

Configuration:

```
config :omniss7,
  smsc_mode_enabled: true,
  smsc_service_center_gt_address: "5559999",  # Default SMSc GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "1", weight: 10, response_gt: "5559991"},   #
North America
    %{calling_prefix: "44", weight: 10, response_gt: "5559992"},  #
UK
    %{calling_prefix: "86", weight: 10, response_gt: "5559993"},  #
China
    %{weight: 100, response_gt: "5559999"}  # Default wildcard
  ]
```

CAMEL Gateway Mode

GT NAT affects:

- All SCCP-level responses (gsmSCF GT as Calling Party)

- CAMEL/CAP operation responses (InitialDP, EventReportBCSM, etc.)
- RequestReportBCSMEvent acknowledgments
- ApplyCharging responses
- Continue responses

Configuration:

```
config :omniss7,
  camelgw_mode_enabled: true,
  camel_gsmscf_gt_address: "55512341112",  # Default gsmSCF GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "555", weight: 10, response_gt: "55512341111"},
# Network A
    %{calling_prefix: "666", weight: 10, response_gt: "55512311555"},
# Network B
    %{weight: 100, response_gt: "55512341112"}  # Default wildcard
  ]
```

Use Case: When operating as a gsmSCF (Service Control Function) for multiple networks, each network's gsmSSF may expect responses from a specific gsmSCF GT. GT NAT ensures the correct GT is used based on which gsmSSF is calling.

Logging and Debugging

Enable GT NAT Logging

GT NAT includes automatic logging of all translations:

```
# In logs, you'll see:
[info] GT NAT [SRI-for-SM response]: Calling GT 877234567 -> Response
GT 55512341112
[info] GT NAT [UpdateLocation ISD]: Calling GT 331234567 -> Response
GT 55512341111
[info] GT NAT [MAP BEGIN response]: Calling GT 441234567 -> Response
GT 55512311555
```

The context field shows where the NAT was applied:

- `"SRI-for-SM response"` - In SRI-for-SM handler
- `"UpdateLocation ISD"` - In InsertSubscriberData messages
- `"UpdateLocation END"` - In UpdateLocation END response
- `"MAP BEGIN response"` - Generic MAP BEGIN responses
- `"ISD ACK"` - ISD acknowledgment
- `"HLR error response"` - Error response from HLR
- `"CAMEL response"` - CAMEL/CAP operation responses (gsmSCF)

Validation

The system validates GT NAT configuration at startup:

```
# Check GT NAT config
iex> GtNat.validate_config()
{:ok, [
  %{calling_prefix: "8772", weight: 10, response_gt: "55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt: "55512341111"}
]}

# Check if enabled
iex> GtNat.enabled?()
true

# Get all rules
iex> GtNat.get_rules()
[
  %{calling_prefix: "8772", weight: 10, response_gt: "55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt: "55512341111"}
]
```

Testing GT NAT

Test GT NAT logic programmatically:

```
# Test translation with calling GT only (called_gt is nil)
iex> GtNat.translate_response_gt("877234567", nil, "default_gt")
"55512341112"

# Test translation with both calling and called GT
iex> GtNat.translate_response_gt("877234567", "555123", "default_gt")
"55512341112"

# Test with logging (nil called GT)
iex> GtNat.translate_response_gt_with_logging("877234567", nil,
"default_gt", "test")
# Logs: GT NAT [test]: Calling GT 877234567 -> Response GT
55512341112
"55512341112"

# Test with logging (both GTs)
iex> GtNat.translate_response_gt_with_logging("877234567", "555123",
"default_gt", "test")
# Logs: GT NAT [test]: Calling GT 877234567, Called GT 555123 ->
Response GT 55512341112
"55512341112"
```

```
# Test no match (returns default)
iex> GtNat.translate_response_gt("999999999", "888888", "default_gt")
"default_gt"
```

Troubleshooting

Issue: GT NAT Not Working

Check 1: Is it enabled?

```
iex> Application.get_env(:omniss7, :gt_nat_enabled)
true  # Should be true
```

Check 2: Are rules configured?

```
iex> Application.get_env(:omniss7, :gt_nat_rules)
[%{calling_prefix: "8772", response_gt: "55512341112"}, ...]  #
Should return list
```

Check 3: Check logs Search for "GT NAT" in logs to see if translations are happening.

Issue: Wrong GT in Responses

Symptom: Responses use unexpected GT address

Cause: Rule prefix matching might be too broad or default rule is catching traffic

Solution: Review rule weights and prefixes:

```
# BAD: Wildcard with low weight (catches everything first)
gt_nat_rules: [
  %{weight: 1, response_gt: "111111"},               # This matches
everything first!
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"}  #
Never reached
]

# GOOD: Specific rules with lower weight, wildcard with higher weight
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  #
Specific, low weight
  %{weight: 100, response_gt: "111111"}  # Wildcard, high weight
(fallback)
]
```

Issue: GT NAT Not Applied to Specific Message Type

Symptom: Some responses use NAT'd GT, others don't

Current Coverage:

- ✅ SCCP Calling GT (all responses)
- ✅ SRI-for-SM responses (networkNode-Number)
- ✅ UpdateLocation ISD messages (HLR GT)
- ✅ UpdateLocation END responses
- ✅ ISD acknowledgments
- ✅ MAP BEGIN responses

If a specific message type isn't using GT NAT, it may not be implemented yet. Check the source code or contact support.

Performance Considerations

Lookup Performance

GT NAT uses simple prefix matching with O(n) complexity where n is the number of rules.

Performance tips:

- Keep rule count under 100 for best performance
- Use specific prefixes to reduce rule count
- Default rule (empty prefix) should be last

Benchmark (typical system):

- 10 rules: < 1µs per lookup
- 50 rules: < 5µs per lookup
- 100 rules: < 10µs per lookup

Memory Usage

Each rule requires ~100 bytes of memory:

- 10 rules ≈ 1 KB
- 100 rules ≈ 10 KB

Best Practices

1. Always Include a Wildcard Fallback Rule

```
gt_nat_rules: [
```

```
  %{calling_prefix: "8772", weight: 10, response_gt: "111111"},
  %{calling_prefix: "8773", weight: 10, response_gt: "222222"},
  %{weight: 100, response_gt: "default_gt"}  # Always have a wildcard
with high weight
]
```

2. Use Meaningful Prefixes and Weights

```
# GOOD: Clear, specific prefixes with appropriate weights
%{calling_prefix: "331", weight: 10, response_gt: "..."}  # France
%{calling_prefix: "44", weight: 10, response_gt: "..."}   # UK

# BAD: Overly broad prefixes or confusing weights
%{calling_prefix: "3", weight: 5, response_gt: "..."}     # Too many
countries
%{calling_prefix: "331", weight: 100, response_gt: "..."}  # Weight
should be lower for specific rules
```

3. Document Your Rules

```
gt_nat_rules: [
  # Partner XYZ - UK network (GT range: 4412xxxxxxx)
  # Weight 10: Standard partner priority
  %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

  # Partner ABC - France network (GT range: 33123xxxxxx)
  # Weight 10: Standard partner priority
  %{calling_prefix: "33123", weight: 10, response_gt: "5551002"}
]
```

4. Test Before Deployment

```
# Test in iex before deploying
iex> GtNat.translate_response_gt("44121234567", nil, "default")
"5551001"  # Expected result

# Test with called GT
iex> GtNat.translate_response_gt("44121234567", "555123", "default")
"5551001"  # Expected result
```

5. Monitor Logs

Enable INFO level logging to see all GT NAT translations in production.

Integration with Other Features

STP Mode

GT NAT works independently of STP routing. STP routes based on point codes and destination GTs, while GT NAT handles response addressing.

For more information on STP routing, see the STP Configuration Guide.

CAMEL Integration

GT NAT is **fully integrated** with CAMEL/CAP operations:

SCCP Layer:

- Calling Party GT in all CAMEL responses
- Automatically applied based on incoming gsmSSF GT

Configuration:

- `camel_gsmscf_gt_address` - Default gsmSCF GT (optional)
- If not configured, uses the Called Party GT from incoming request
- GT NAT rules override the default based on calling party prefix

Example:

```
# When gsmSSF 555123456 calls your gsmSCF
# Incoming: Called=55512341112, Calling=555123456
# GT NAT lookup: "555" -> response_gt="55512341111"
# Response: Called=555123456, Calling=55512341111
```

Load Balancing

GT NAT can be combined with M3UA load balancing for advanced traffic management.

Migration Guide

Enabling GT NAT on Existing System

1. **Prepare Configuration**

   ```
   # Add to runtime.exs (keep disabled initially)
   config :omniss7,
     gt_nat_enabled: false,  # Start disabled
     gt_nat_rules: [
       # Your rules here with weights
   ```

```
    %{calling_prefix: "877", weight: 10, response_gt: "111111"},
    %{weight: 100, response_gt: "999999"}  # Wildcard fallback
  ]
```

2. **Test Configuration**

```
# Validate config compiles
mix compile

# Test in iex
iex -S mix
iex> GtNat.validate_config()
```

3. **Enable in Staging**

```
gt_nat_enabled: true  # Change to true
```

4. **Monitor Logs**

```
tail -f log/omniss7.log | grep "GT NAT"
```

5. **Deploy to Production**

    - Deploy during maintenance window
    - Monitor first 24 hours closely
    - Have rollback plan ready (set `gt_nat_enabled: false`)

Support

For issues or questions:

- Check logs for "GT NAT" messages
- Validate config with `GtNat.validate_config()`
- Review this guide's troubleshooting section
- Contact OmniSS7 support with log excerpts

See Also

- HLR Guide - HLR mode configuration
- SMSC Guide - SMSc mode configuration
- STP Guide - STP routing configuration
- Configuration Reference - Complete config reference

HLR Configuration Guide

← Back to Main Documentation

This guide provides configuration for using OmniSS7 as a **Home Location Register (HLR/HSS)** with **OmniHSS** as the backend subscriber database.

OmniHSS Integration

OmniSS7 HLR mode functions as an SS7 signaling frontend that interfaces with **OmniHSS**, a full-featured Home Subscriber Server (HSS) backend. This architecture separates concerns:

- **OmniSS7 (HLR Frontend)**: Handles all SS7/MAP protocol signaling, SCCP routing, and network communication
- **OmniHSS (HSS Backend)**: Manages subscriber data, authentication, provisioning, and advanced features

Why OmniHSS?

OmniHSS provides carrier-grade subscriber management with features including:

- **Multi-IMSI Support**: Each subscriber can have multiple IMSIs associated with a single MSISDN for international roaming, network switching, and eSIM provisioning
- **Flexible Authentication**: Support for both Milenage (3G/4G/5G) and COMP128 (2G) authentication algorithms
- **Circuit & Packet Session Tracking**: Independent tracking of CS (circuit-switched) and PS (packet-switched) network registrations
- **Advanced Provisioning**: Customizable service profiles, supplementary services, and CAMEL subscription data
- **API-First Design**: RESTful HTTP API for integration with billing, CRM, and provisioning systems
- **Real-time Updates**: Location tracking, session management, and authentication vector generation

All subscriber data, authentication credentials, and service configurations are stored and managed in OmniHSS. OmniSS7 queries OmniHSS via HTTPS API calls to respond to MAP operations like UpdateLocation, SendAuthenticationInfo, and SendRoutingInfo.

Important: OmniSS7 HLR mode is a **signaling frontend only**. All subscriber management logic, authentication algorithms, provisioning rules, and database operations are handled by OmniHSS. This guide covers the SS7/MAP protocol configuration in OmniSS7. For information about subscriber provisioning, authentication configuration, service profiles, and administrative operations, **refer to the OmniHSS documentation**.

Multi-IMSI Support

OmniHSS natively supports Multi-IMSI configurations, allowing a single subscriber (identified by MSISDN) to have multiple IMSIs. This enables:

- **International Roaming Profiles**: Different IMSIs for different regions to reduce roaming costs
- **eSIM Multi-Profile**: Multiple network profiles on a single eSIM-capable device
- **Network Switching**: Seamless switching between networks without changing MSISDN
- **Dual SIM Coordination**: Coordination across multiple physical or virtual SIMs
- **Testing & Development**: Multiple test IMSIs pointing to the same subscriber

How it works:

- Each IMSI has its own authentication credentials (Ki, OPc, algorithm)
- Each IMSI can have independent circuit and packet session registrations
- Subscriber services and profiles can be shared or customized per-IMSI
- OmniSS7 queries OmniHSS by IMSI, and OmniHSS returns the appropriate subscriber data
- Billing systems can track usage per-IMSI while associating all IMSIs to a single account

Example Multi-IMSI scenario:

```
Subscriber MSISDN: +1-555-123-4567
├─ IMSI 1: 310260123456789 (US Home Network - Milenage auth)
├─ IMSI 2: 208011234567890 (France Roaming Profile - Milenage auth)
└─ IMSI 3: 440201234567891 (UK Roaming Profile - COMP128 auth)
```

All three IMSIs can be used independently for network registration, but they all belong to the same subscriber account. OmniHSS manages the IMSI-to-subscriber mapping and ensures proper authentication and provisioning for each IMSI.

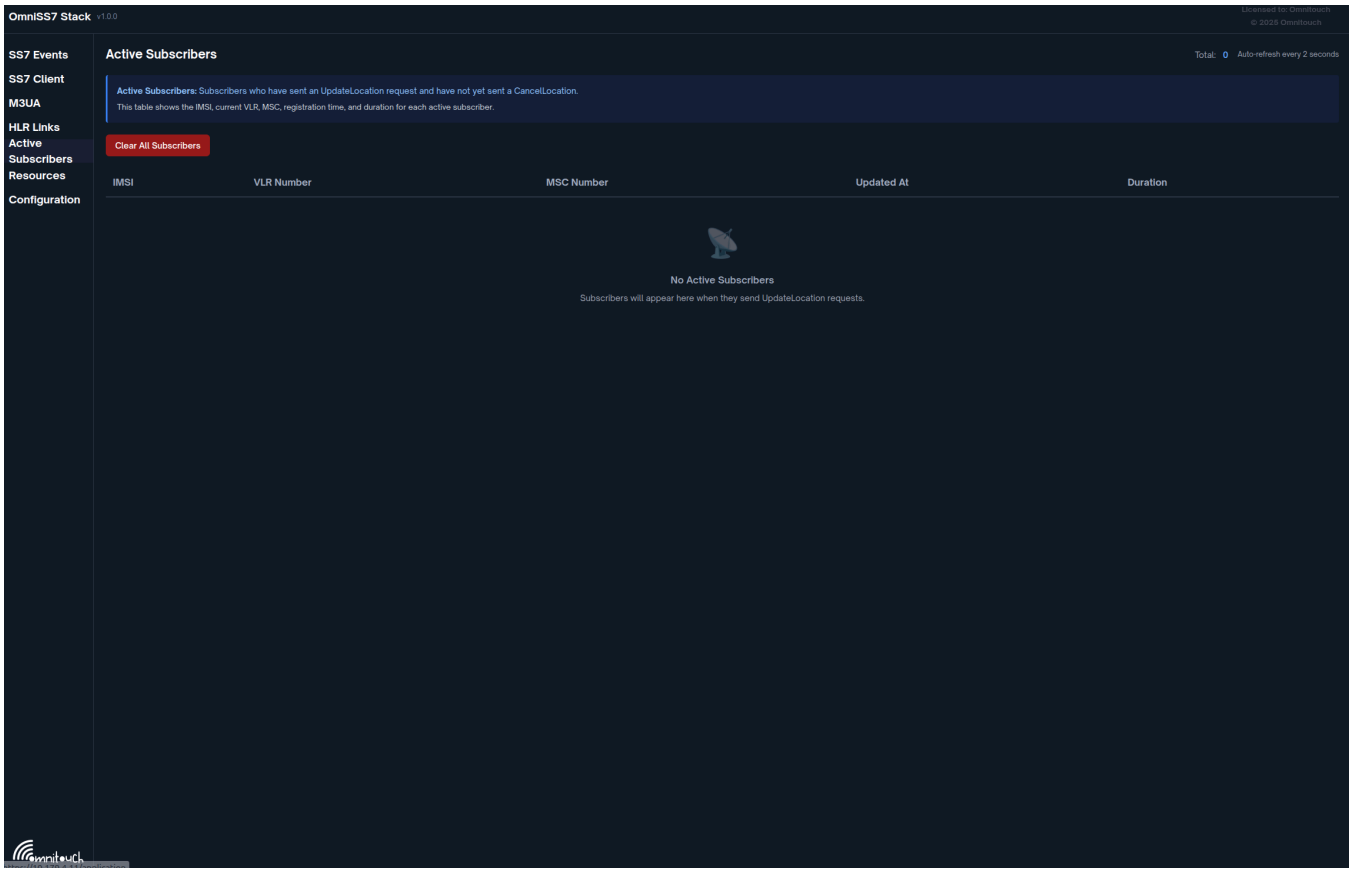


Table of Contents

What is HLR Mode?

HLR Mode enables OmniSS7 to function as a Home Location Register for:

- **Subscriber Management**: Store and manage subscriber data
- **Authentication**: Generate authentication vectors for network access
- **Location Tracking**: Process location updates from VLRs
- **Routing Information**: Provide routing info for calls and SMS

HLR Architecture

Enabling HLR Mode

OmniSS7 can operate in different modes. To use it as an HLR, you need to enable HLR mode in the configuration.

Switching to HLR Mode

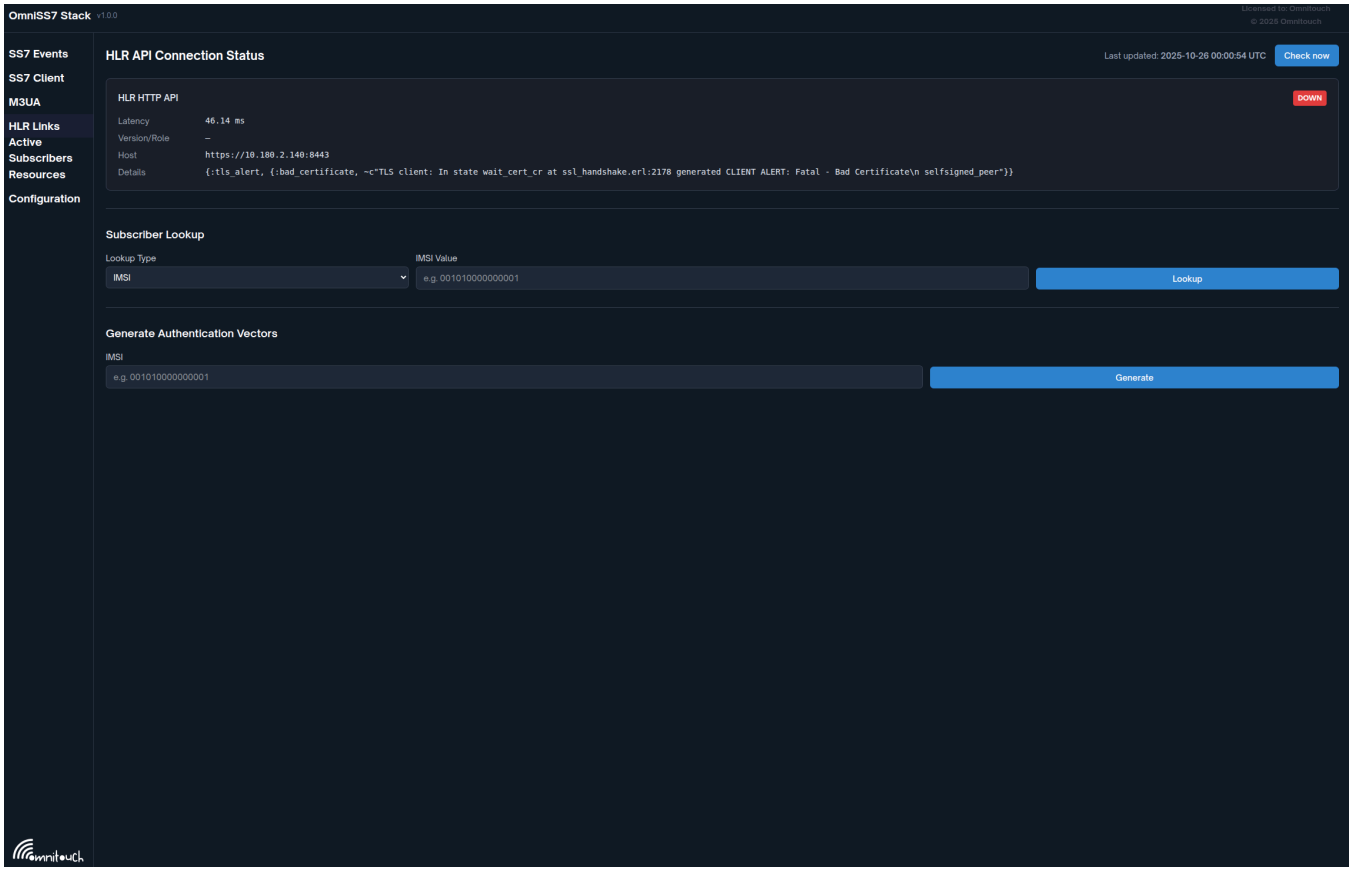
OmniSS7's `config/runtime.exs` contains three pre-configured operational modes. To enable HLR mode:

1. **Open** `config/runtime.exs`
2. **Find** the three configuration sections (lines 53-174):
   - Configuration 1: STP Mode (lines 53-85)
   - Configuration 2: HLR Mode (lines 87-123)
   - Configuration 3: SMSc Mode (lines 125-174)
3. **Comment out** the currently active configuration (add `#` to each line)
4. **Uncomment** the HLR configuration (remove `#` from lines 87-123)
5. **Customize** the configuration parameters as needed
6. **Restart** the application: `iex -S mix`

HLR Mode Configuration

The complete HLR configuration looks like this:

```
config :omniss7,
  # Mode flags - Enable HLR features only
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  # OmniHSS Backend API Configuration
  hlr_api_base_url: "https://10.180.2.140:8443",

  # HLR Service Center GT Address for SMS operations
  hlr_service_center_gt_address: "1234567890",

  # MSISDN ↔ IMSI Mapping Configuration
  # See: MSISDN ↔ IMSI Mapping section for details
  hlr_imsi_plmn_prefix: "50557",
  hlr_msisdn_country_code: "61",
  hlr_msisdn_nsn_offset: 0,
  hlr_msisdn_nsn_length: 9,

  # InsertSubscriberData Configuration
  # Network Access Mode: :packetAndCircuit, :packetOnly, or :circuitOnly
  isd_network_access_mode: :packetAndCircuit,

  # Send ISD #2 (Supplementary Services data)
  isd_send_ss_data: true,

  # Send ISD #3 (Call Barring data)
  isd_send_call_barring: true,

  # CAMEL Configuration (for SendRoutingInfo responses)
  # Service Key for CAMEL service initiation
  camel_service_key: 11_110,

  # CAMEL Trigger Detection Point
  # Options: :termAttemptAuthorized, :tBusy, :tNoAnswer, :tAnswer
  camel_trigger_detection_point: :termAttemptAuthorized,

  # Home VLR Prefixes
  # List of VLR address prefixes that are considered "home" network
  # If subscriber's VLR starts with one of these prefixes, use standard SRI response
  # Otherwise, subscriber is roaming and we need to send PRN to get MSRN
  home_vlr_prefixes: ["123456"],

  # M3UA Connection Configuration
  # Connect as ASP for receiving MAP operations (UpdateLocation, SendAuthInfo, etc.)
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # Local endpoint (HLR system)
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    # Remote STP endpoint
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```



Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

Parameter	Type	Default	Description	Example
`hlr_api_base_url`	String	*Required*	**OmniHSS** backend API endpoint	`"https://10.179.3.219:8443"`
`hlr_service_center_gt_address`	String	*Required*	HLR GT address used in UpdateLocation responses	`"5551234568"`
`smsc_service_center_gt_address`	String	*Required*	SMSC GT address returned in SRI-for-SM responses	`"5551234567"`
`hlr_smsc_alert_gts`	List	`[]`	List of SMSc GTs to send alertServiceCenter after UpdateLocation	`["15559876543", "15559876544"]`
`hlr_alert_location_expiry_seconds`	Integer	`172800`	Location expiry time (seconds) when SMSc receives alertServiceCenter	`86400`
`hlr_imsi_plmn_prefix`	String	`"50557"`	PLMN prefix (MCC+MNC) for MSISDN→IMSI mapping (see MSISDN ↔ IMSI Mapping)	`"001001"`
`hlr_msisdn_country_code`	String	`"61"`	Country code prefix for reverse IMSI→MSISDN mapping (see MSISDN ↔ IMSI Mapping)	`"1"`
`hlr_msisdn_nsn_offset`	Integer	`0`	Offset into MSISDN for NSN extraction (see MSISDN ↔ IMSI Mapping)	`0`
`hlr_msisdn_nsn_length`	Integer	`9`	Length of National Subscriber Number to extract (see MSISDN ↔ IMSI Mapping)	`10`
`isd_network_access_mode`	Atom	`:packetAndCircuit`	Network access mode for InsertSubscriberData	`:packetOnly`
`isd_send_ss_data`	Boolean	`true`	Send ISD #2 with Supplementary Services data	`false`
`isd_send_call_barring`	Boolean	`true`	Send ISD #3 with Call Barring data	`false`
`camel_service_key`	Integer	`11_110`	CAMEL service key for SendRoutingInfo responses	`100`
`camel_trigger_detection_point`	Atom	`:termAttemptAuthorized`	CAMEL trigger detection point	`:tBusy`
`home_vlr_prefixes`	List	`["5551231"]`	List of VLR address prefixes considered "home" network	`["555123"]`
`local_ip`	Tuple	*Required*	Your HLR system's IP address	`{10, 179, 4, 12}`
`local_port`	Integer	`2905`	Local SCTP port	`2905`
`remote_ip`	Tuple	*Required*	STP IP address for SS7 connectivity	`{10, 179, 4, 10}`
`remote_port`	Integer	`2905`	Remote SCTP port	`2905`
`routing_context`	Integer	`1`	M3UA routing context ID	`1`

What Happens When HLR Mode is Enabled

When `hlr_mode_enabled: true`, the web UI will show:

- ◇ **SS7 Events** - Event logging
- ◇ **SS7 Client** - MAP operation testing
- ◇ **M3UA** - Connection status
- ◇ **HLR Links** - HLR API status + subscriber management ← *HLR-specific*
- ◇ **Resources** - System monitoring
- ◇ **Configuration** - Config viewer

The **Routing**, **Routing Test**, and **SMSc Links** tabs will be hidden.

Important Notes

- **Required Configuration**: The `hlr_service_center_gt_address` parameter is **mandatory**. The application will fail to start if it is not configured.
- **OmniHSS Backend**: The OmniHSS API backend must be accessible at the configured `hlr_api_base_url`
- **API Request Timeout**: All OmniHSS API requests have a **hardcoded 5-second timeout**
- **MAP Request Timeout**: All MAP requests (SRI, UpdateLocation, SendAuthInfo, etc.) have a **hardcoded 10-second timeout**
- **ISD Timeout**: Each InsertSubscriberData (ISD) message in an UpdateLocation sequence has a **hardcoded 10-second timeout**
- M3UA connection to STP is required for receiving MAP operations
- After changing modes, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for information on using the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

Subscriber Database

OmniHSS manages all subscriber data including identities, authentication credentials, service profiles, and location information. OmniSS7 retrieves this data via RESTful API calls.

OmniHSS Subscriber Model

OmniHSS stores comprehensive subscriber information:

- **Multiple IMSIs per subscriber**: Support for Multi-IMSI configurations (eSIM, roaming profiles, network switching)
- **Authentication credentials**: Ki, OPc, and algorithm selection (Milenage or COMP128)
- **Service profiles**: Subscriber category, allowed services, QoS parameters
- **Location tracking**: Current VLR/MSC (circuit session) and SGSN/GGSN (packet session) independent tracking
- **CAMEL subscription data**: Service keys, trigger points, and gsmSCF addresses
- **Supplementary services**: Call forwarding, barring, waiting, CLIP/CLIR configurations
- **Administrative state**: Enabled/disabled, service restrictions, expiration dates

---

Authentication Vectors

Generate Auth Vectors

OmniHSS generates authentication vectors using the Milenage or COMP128 algorithms based on each subscriber's configured authentication method. When OmniSS7 receives **sendAuthenticationInfo** MAP requests:

1. OmniSS7 extracts the IMSI from the MAP request
2. OmniSS7 calls the OmniHSS API to generate authentication vectors
3. OmniHSS retrieves the subscriber's Ki and OPc credentials
4. OmniHSS generates the requested number of vectors (RAND, XRES, CK, IK, AUTN)
5. OmniSS7 encodes the vectors into MAP format and returns them to the requesting VLR/SGSN

OmniHSS API Integration

OmniSS7 communicates with OmniHSS via HTTPS REST API to retrieve subscriber information, update location data, and generate authentication vectors:

```
config :omniss7,
  hlr_api_base_url: "https://omnihss-server:8443"
```

When OmniSS7 receives MAP operations from the SS7 network, it queries OmniHSS to:

- **Retrieve subscriber data** by IMSI or MSISDN
- **Generate authentication vectors** using stored Ki/OPc credentials
- **Update circuit session location** when subscribers perform UpdateLocation
- **Check subscriber status** and service entitlements

Location Updates

Update Location Processing

When receiving **updateLocation** MAP requests, OmniSS7 coordinates with OmniHSS to register the subscriber at a new VLR:

1. **Extract location info** from UpdateLocation request (IMSI, new VLR GT, new MSC GT)
2. **Query OmniHSS** to verify subscriber exists and is enabled
3. **Update circuit session** in OmniHSS with new VLR/MSC location
4. **Send InsertSubscriberData (ISD)** messages to provision the subscriber at the new VLR
5. **Return UpdateLocation response** to VLR (includes HLR GT from `hlr_service_center_gt_address`)
6. **Send alertServiceCenter** to configured SMSc GTs (if `hlr_smsc_alert_gts` is populated)

Note: The `hlr_service_center_gt_address` configuration parameter specifies the HLR's Global Title that is returned in UpdateLocation responses. This allows the VLR/MSC to identify and route messages back to this HLR.

Alert Service Center Integration

After a successful UpdateLocation, the HLR can automatically notify SMSc systems that a subscriber is now reachable by sending **alertServiceCenter** (MAP opcode 64) messages. For information on how the SMSc handles these alerts, see Alert Service Center Handling in SMSc Guide.

Configuration

Configure the list of SMSc Global Titles to notify:

```
config :omniss7,
  # List of SMSc GTs to send alertServiceCenter after UpdateLocation
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],

  # Location expiry time when SMSc receives alertServiceCenter (default: 48 hours)
  hlr_alert_location_expiry_seconds: 172800
```

Flow Diagram

Behavior

When a subscriber performs UpdateLocation:

1. HLR sends alertServiceCenter to **each** SMSc GT in the `hlr_smsc_alert_gts` list
2. Message includes the subscriber's MSISDN
3. HLR uses `hlr_service_center_gt_address` as the calling party GT
4. SCCP addressing: calling SSN=6 (HLR), called SSN=8 (SMSc)

The SMSc receives the alert and:

- **Strips TON/NPI prefix** from MSISDN (e.g., "19123123213" → "123123213")
- Marks the subscriber as reachable in its location database (via POST to /api/location)
- **Sets `user_agent` field** to the HLR GT when calling the API (for tracking which HLR sent the alert)
- Sets location expiry time based on `hlr_alert_location_expiry_seconds`
- Tracks the subscriber in the SMSc Subscriber Tracker for monitoring

Testing

Use the **Active Subscribers** page in the Web UI to manually send alertServiceCenter messages for testing:

1. Navigate to the "Active Subscribers" tab
2. Find the "Test Alert Service Center" section
3. Enter MSISDN, SMSc GT, and HLR GT (defaults are pre-populated from config)
    - SMSc GT defaults to first entry in `hlr_smsc_alert_gts`
    - HLR GT defaults to `hlr_service_center_gt_address`
4. Click "Send alertServiceCenter"

This is useful for testing SMSc alert handling without requiring a full UpdateLocation flow. The form uses `phx-blur` validation to avoid showing errors while typing.

InsertSubscriberData (ISD) Configuration

After a successful UpdateLocation, the HLR sends subscriber provisioning data to the VLR using **InsertSubscriberData** (ISD) messages. The ISD configuration allows you to customize what data is sent and how.

For configuration parameter reference, see ISD Configuration in Configuration Reference.

ISD Sequence

The HLR can send up to 3 sequential ISD messages:

1. **ISD #1** (Always sent) - Basic subscriber data:
    - IMSI
    - MSISDN
    - Subscriber category
    - Subscriber status (serviceGranted)
    - Bearer service list
    - Teleservice list
    - Network access mode
2. **ISD #2** (Optional) - Supplementary Services (SS) data:
    - Call forwarding settings (unconditional, busy, no reply, not reachable)
    - Call waiting
    - Call hold
    - Multi-party service
    - Supplementary service status and features
3. **ISD #3** (Optional) - Call Barring data:
    - Barring of all outgoing calls (BAOC)
    - Barring of outgoing international calls (BOIC)
    - Access restriction data

Configuration Options

```
# InsertSubscriberData Configuration
# Network Access Mode: :packetAndCircuit, :packetOnly, or :circuitOnly
isd_network_access_mode: :packetAndCircuit,

# Send ISD #2 (Supplementary Services data)
isd_send_ss_data: true,

# Send ISD #3 (Call Barring data)
isd_send_call_barring: true,
```

Network Access Mode

The `isd_network_access_mode` parameter controls what type of network access the subscriber is allowed:

Value	Description	Use Case
`:packetAndCircuit`	Both packet-switched (GPRS/LTE) and circuit-switched (voice)	Default - Full service subscribers
`:packetOnly`	Packet-switched only (data/LTE)	Data-only SIM cards, IoT devices
`:circuitOnly`	Circuit-switched only (voice/SMS)	Legacy devices, voice-only plans

Controlling ISD Messages

You can control which ISD messages are sent based on your network requirements:

Send all ISDs (Default - Full feature set):

```
isd_send_ss_data: true,
isd_send_call_barring: true,
```

Send only basic subscriber data (Minimal provisioning):

```
isd_send_ss_data: false,
isd_send_call_barring: false,
```

Send basic + supplementary services (No call barring):

```
isd_send_ss_data: true,
isd_send_call_barring: false,
```

ISD Flow Example

When UpdateLocation is received:

```
VLR → HLR: UpdateLocation (BEGIN)
HLR → VLR: InsertSubscriberData #1 (CONTINUE) - Basic data
VLR → HLR: ISD #1 ACK (CONTINUE)
```

```
HLR → VLR: InsertSubscriberData #2 (CONTINUE) - SS data [if enabled]
VLR → HLR: ISD #2 ACK (CONTINUE)
HLR → VLR: InsertSubscriberData #3 (CONTINUE) - Call barring [if enabled]
VLR → HLR: ISD #3 ACK (CONTINUE)
HLR → VLR: UpdateLocation Response (END)
```

If `isd_send_ss_data` or `isd_send_call_barring` are set to `false`, those ISD messages are skipped, and the UpdateLocation END is sent sooner.

Best Practices

- **Default Configuration**: Use `:packetAndCircuit` and enable all ISDs for maximum compatibility
- **IoT/M2M**: Use `:packetOnly` and disable SS data/call barring for data-only devices
- **Interoperability**: Some older VLRs may not support all supplementary services - disable `isd_send_ss_data` if encountering issues
- **Performance**: Disabling unused ISDs reduces message overhead and speeds up location updates

---

CAMEL Integration

CAMEL Configuration for SendRoutingInfo

When responding to **SendRoutingInfo** (SRI) requests from a GMSC (Gateway MSC), the HLR can instruct the GMSC to invoke CAMEL services for intelligent call routing and service control.

For configuration parameter reference, see CAMEL Configuration in Configuration Reference.

What is CAMEL?

CAMEL (Customized Applications for Mobile network Enhanced Logic) is a protocol that enables intelligent network services in GSM/UMTS networks. It allows network operators to implement value-added services like:

- Prepaid billing
- Call screening and barring
- Virtual Private Networks (VPN)
- Premium rate services
- Call forwarding with custom logic
- Location-based services

Configuration Options

```
# CAMEL Configuration (for SendRoutingInfo responses)
# Service Key for CAMEL service initiation
camel_service_key: 11_110,

# CAMEL Trigger Detection Point
# Options: :termAttemptAuthorized, :tBusy, :tNoAnswer, :tAnswer
camel_trigger_detection_point: :termAttemptAuthorized,
```

Service Key

The `camel_service_key` identifies which CAMEL service should be invoked at the gsmSCF (Service Control Function). This is a numeric identifier configured in your network:

Service Key	Typical Use Case
`11_110`	Prepaid terminating call control (default)
`100`	Originating prepaid service
`200`	Call forwarding with custom logic
`300`	Virtual Private Network (VPN)
Custom	Operator-specific services

Configuration Example:

```
# For prepaid terminating call control
camel_service_key: 11_110,

# For VPN service
camel_service_key: 300,
```

Trigger Detection Point

The `camel_trigger_detection_point` specifies when the CAMEL service should be triggered during call setup:

Detection Point	Description	When Triggered
`:termAttemptAuthorized`	Call attempt authorized (default)	Before call is routed to subscriber
`:tBusy`	Terminating busy	When subscriber is busy
`:tNoAnswer`	Terminating no answer	When subscriber doesn't answer
`:tAnswer`	Terminating answer	When subscriber answers the call

Configuration Examples:

Standard prepaid control (trigger before routing):

```
camel_trigger_detection_point: :termAttemptAuthorized,
```

Custom busy handling (trigger when busy):

```
camel_trigger_detection_point: :tBusy,
```

Answer-based billing (trigger on answer):

```
camel_trigger_detection_point: :tAnswer,
```

SRI Response with CAMEL

When configured, SendRoutingInfo responses include CAMEL subscription information:

```
GMSC → HLR: SendRoutingInfo (BEGIN)
HLR → GMSC: SRI Response (END) with:
  - IMSI
  - VLR number
  - Subscriber state
  - CAMEL routing info:
    * Service Key: 11_110
    * gsmSCF Address: <configured address>
    * Trigger Detection Point: termAttemptAuthorized
    * Default Call Handling: continueCall

GMSC contacts gsmSCF at trigger point to execute CAMEL service
```

Best Practices

- **Production Networks**: Use standardized service keys agreed upon with your gsmSCF provider
- **Testing**: Use `:termAttemptAuthorized` for most comprehensive testing
- **Prepaid Services**: Service key `11_110` is a common industry standard for prepaid terminating calls
- **Fallback Handling**: `defaultCallHandling: :continueCall` ensures calls proceed if gsmSCF is unreachable

---

Roaming Subscriber Handling

Home VLR vs Roaming VLR Detection

When the HLR receives a **SendRoutingInfo** (SRI) request, it needs to determine whether the subscriber is on a "home" VLR (within your network) or on a roaming VLR (visiting another network). The behavior differs based on this determination:

For configuration parameter reference, see Home VLR Prefixes in Configuration Reference.

- **Home VLR**: Return standard SRI response with CAMEL routing information
- **Roaming VLR**: Send a Provide Roaming Number (PRN) request to obtain an MSRN, then return it in the SRI response

Configuration

```
# Home VLR Prefixes
# List of VLR address prefixes that are considered "home" network
# If subscriber's VLR address starts with one of these prefixes, use standard SRI response
# Otherwise, subscriber is roaming and we need to send PRN to get MSRN
home_vlr_prefixes: ["555123"],
```

Configuration Example:

```
# Single home network
home_vlr_prefixes: ["555123"],

# Multiple home networks (e.g., different regions or subsidiaries)
home_vlr_prefixes: ["555123", "555124", "555125"],
```

How It Works

1. Home Subscriber Flow (Standard)

When the subscriber's VLR address starts with a configured home prefix:

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR queries backend API for subscriber data
HLR checks VLR address: "5551234567"
HLR determines: VLR starts with "555123" → Home network
HLR → GMSC: SRI Response with CAMEL routing info:
  - IMSI
  - VLR number: "5551234567"
  - gsmSCF address (MSC): "5551234501"
  - CAMEL service key: 11_110
  - Trigger detection point: termAttemptAuthorized
```

2. Roaming Subscriber Flow (PRN Required)

When the subscriber's VLR address does NOT match any home prefix:

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR queries backend API for subscriber data
HLR checks VLR address: "49170123456"
HLR determines: VLR doesn't start with "555123" → Roaming
HLR → MSC: ProvideRoamingNumber (PRN):
  - MSISDN: "1234567890"
  - IMSI: "999999876543210"
  - MSC number: "49170123456"
  - GMSC address: "5551234501"
MSC → HLR: PRN Response with MSRN: "49170999888777"
HLR → GMSC: SRI Response with routing info:
  - IMSI
  - VLR number: "49170123456"
  - Roaming Number (MSRN): "49170999888777"
```

Response Structure Differences

Home Subscriber SRI Response

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :camelRoutingInfo, %{
      gmscCamelSubscriptionInfo: %{
        "t-CSI": %{
          serviceKey: 11_110,
          "gsmSCF-Address": "5551234501",
          defaultCallHandling: :continueCall,
          "t-BcsmTriggerDetectionPoint": :termAttemptAuthorized
        }
      }
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "5551234567"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

Roaming Subscriber SRI Response

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :routingInfo, %{
      roamingNumber: "49170999888777"  # MSRN from PRN
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "49170123456"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

Provide Roaming Number (PRN) Operation

PRN Request Structure

The PRN request sent to the MSC/VLR contains:

Field	Source	Description
MSISDN	SRI request	Subscriber's phone number
IMSI	HLR API	Subscriber's IMSI
MSC Number	HLR API	MSC serving the roaming subscriber (`serving_msc`)
GMSC Address	SRI request	GMSC making the original SRI request
Call Reference Number	Static	Call reference identifier
Supported CAMEL Phases	Static	CAMEL phases supported by GMSC

PRN Response Handling

The HLR expects a PRN response containing:

- **MSRN** (Mobile Station Roaming Number): A temporary number allocated by the visited network for routing the call

Error Handling:

- If PRN times out → Returns error 27 (Absent Subscriber) in SRI response
- If PRN fails → Returns error 27 (Absent Subscriber) in SRI response
- If MSRN cannot be extracted → Returns error 27 (Absent Subscriber) in SRI response

Configuration Examples

Single Home Network Operator

```
# All VLR addresses starting with "555123" are considered home
home_vlr_prefixes: ["555123"],
```

- VLR `5551234567` → Home (CAMEL response)
- VLR `5551235001` → Home (CAMEL response)
- VLR `49170123456` → Roaming (PRN + MSRN response)

Multi-Region Operator

```
# Multiple home networks across different regions
home_vlr_prefixes: ["555123", "555124", "555125"],
```

- VLR 5551234567 → Home (region 1)
- VLR 5552341234 → Home (region 2)
- VLR 5553411111 → Home (region 3)
- VLR `44201234567` → Roaming (international)

Testing Configuration

For testing PRN functionality, set an empty list to treat all VLRs as roaming:

```
# All VLRs are treated as roaming (for testing PRN flow)
home_vlr_prefixes: [],
```

Best Practices

- **Prefix Selection**: Use the shortest unique prefix that identifies your network's VLRs (e.g., country code + network code)
- **Multiple Prefixes**: Include all VLR prefixes in your network, including different regions and subsidiaries
- **Roaming Agreements**: Ensure PRN is properly supported by roaming partner networks
- **Testing**: Test both home and roaming scenarios thoroughly before production deployment
- **Monitoring**: Monitor PRN timeout rates to identify connectivity issues with roaming partners

Troubleshooting

Symptom: All subscribers treated as roaming

- **Cause**: `home_vlr_prefixes` not configured or prefixes don't match VLR addresses
- **Solution**: Check VLR addresses in your database and update prefixes accordingly

Symptom: PRN requests timing out

- **Cause**: Network connectivity issues to roaming partner MSC/VLR
- **Solution**: Verify M3UA/SCCP routing to remote MSC addresses

Symptom: Invalid MSRN in SRI response

- **Cause**: PRN response format from roaming partner doesn't match expected structure
- **Solution**: Review PRN response logs and adjust `extract_msrn_from_prn/1` if needed

---

HLR Operations

Supported MAP Operations

- `updateLocation` (Opcode 2) - Register VLR location
- `sendAuthenticationInfo` (Opcode 56) - Generate auth vectors
- `sendRoutingInfo` (Opcode 22) - Provide MSRN for calls with CAMEL support
- `sendRoutingInfoForSM` (Opcode 45) - Provide MSC GT for SMS
- `cancelLocation` (Opcode 3) - Deregister from old VLR
- `insertSubscriberData` (Opcode 7) - Push subscriber profile

Response Field Mapping

This section details where each field in HLR responses comes from.

SendRoutingInfo (SRI) Response

Purpose: Provides routing information for incoming calls to a subscriber.

The HLR provides two different response types based on whether the subscriber is on a home VLR or roaming:

Home Subscriber Response (CAMEL Routing)

Used when the subscriber's VLR address starts with a configured `home_vlr_prefixes` value.

Response Structure:

Field	Source	Description	Example
IMSI	OmniHSS API	Subscriber's IMSI from OmniHSS database	`"999999876543210"`
VLR Number	OmniHSS API	Current VLR serving the subscriber (`circuit_session.assigned_vlr`)	`"5551234567"`
Subscriber State	Static	Always `notProvidedFromVLR`	`:notProvidedFromVLR`
extendedRoutingInfo	-	Type: `camelRoutingInfo`	-
gsmSCF Address	OmniHSS API	MSC serving the subscriber (`circuit_session.assigned_msc`)	`"5551234501"`
Service Key	runtime.exs	CAMEL service identifier (`camel_service_key`)	`11_110`
Trigger Detection Point	runtime.exs	When to trigger CAMEL (`camel_trigger_detection_point`)	`:termAttemptAuthorized`
CAMEL Capability Handling	Static	CAMEL phase support level	`3`
Default Call Handling	Static	Fallback if gsmSCF unreachable	`:continueCall`

Roaming Subscriber Response (MSRN Routing)

Used when the subscriber's VLR address does NOT match any configured `home_vlr_prefixes` value.

Response Structure:

Field	Source	Description	Example
IMSI	OmniHSS API	Subscriber's IMSI from OmniHSS database	`"999999876543210"`
VLR Number	OmniHSS API	Current VLR serving the subscriber (`circuit_session.assigned_vlr`)	`"49170123456"`
Subscriber State	Static	Always `notProvidedFromVLR`	`:notProvidedFromVLR`
extendedRoutingInfo	-	Type: `routingInfo`	-
Roaming Number (MSRN)	PRN Response	MSRN obtained from ProvideRoamingNumber request	`"49170999888777"`

Routing Decision Logic:

```
1. OmniSS7 receives SendRoutingInfo request
2. OmniSS7 queries subscriber data from OmniHSS API
3. OmniSS7 checks VLR address against home_vlr_prefixes:

   If VLR starts with home prefix:
     → Return CAMEL routing info (home subscriber flow)

   If VLR does NOT match any home prefix:
     → Send ProvideRoamingNumber (PRN) to MSC
     → Extract MSRN from PRN response
     → Return routing info with MSRN (roaming subscriber flow)
```

Data Flow:

- OmniSS7 queries OmniHSS for subscriber information
- OmniHSS returns IMSI, current VLR/MSC location, and subscriber state
- OmniSS7 uses this data to construct the MAP response

Configuration Requirements:

```
# In runtime.exs
home_vlr_prefixes: ["555123"],  # List of home VLR prefixes
```

Error Responses:

- If `serving_vlr` and `serving_msc` are `null`: Returns error 27 (Absent Subscriber)
- If subscriber not found: Returns error 1 (Unknown Subscriber)
- If PRN request times out (roaming case): Returns error 27 (Absent Subscriber)
- If PRN response invalid (roaming case): Returns error 27 (Absent Subscriber)

UpdateLocation Response with InsertSubscriberData

Purpose: Registers subscriber at new VLR and provisions subscriber data.

UpdateLocation END Response

Field	Source	Description	Example
HLR Number	runtime.exs	This HLR's Global Title (`hlr_service_center_gt_address`)	`"5551234568"`
TCAP Message Type	Static	Final response after all ISDs	`END`

InsertSubscriberData #1 (Basic Subscriber Data)

Field	Source	Description	Example
IMSI	Request	From UpdateLocation request	`"999999876543210"`
MSISDN	OmniHSS API	Subscriber's phone number from OmniHSS	`"555123456"`
Category	Static	Subscriber category	`"\n"` (0x0A)
Subscriber Status	Static	Service status	`:serviceGranted`
Bearer Service List	Static	Supported bearer services	`[<<31>>]`
Teleservice List	Static	Supported teleservices	`[<<17>>, "!", "\""]`
Network Access Mode	runtime.exs	Packet/circuit access (`isd_network_access_mode`)	`:packetAndCircuit`

InsertSubscriberData #2 (Supplementary Services) - Optional

Field	Source	Description	Controlled By
Provisioned SS	Static	Supplementary services data	`isd_send_ss_data: true`
Call Forwarding	Static	Forwarding configurations (unconditional, busy, no reply, not reachable)	Config enabled
Call Waiting	Static	Call waiting service status	Config enabled
Multi-party Service	Static	Conference call support	Config enabled

ISD #2 includes:

- Call forwarding unconditional (SS code 21)
- Call forwarding on busy (SS code 41)
- Call forwarding on no reply (SS code 42)
- Call forwarding on not reachable (SS code 62)
- Call waiting (SS code 43)
- Multi-party service (SS code 51)
- CLIP/CLIR services

InsertSubscriberData #3 (Call Barring) - Optional

Field	Source	Description	Controlled By
Call Barring Info	Static	Call barring configurations	`isd_send_call_barring: true`
BAOC	Static	Barring of All Outgoing Calls (SS code 146)	Config enabled
BOIC	Static	Barring of Outgoing International Calls (SS code 147)	Config enabled
Access Restriction Data	Static	Network access restrictions	Config enabled

ISD Sequence Control:

- ISD #1: **Always sent** - Contains essential subscriber data
- ISD #2: Sent only if `isd_send_ss_data: true` in runtime.exs
- ISD #3: Sent only if `isd_send_call_barring: true` in runtime.exs

SendRoutingInfoForSM (SRI-for-SM) Response

Purpose: Provides MSC/SMSC routing information for SMS delivery. When an SMSc needs to deliver an SMS to a subscriber, it sends a SRI-for-SM request to the HLR to determine where to route the message.

Response Structure:

Field	Source	Description	How Generated	Example
IMSI	Calculated	Synthetic IMSI derived from MSISDN	`PLMN_PREFIX + zero_padded_MSISDN`	`"001001555123456"`
Network Node Number	runtime.exs	SMSC GT address for SMS routing	`smsc_service_center_gt_address`	`"5551234567"`

Configuration Parameters (from `runtime.exs`):

```
# Service Center GT Address (returned in SRI-for-SM responses)
# This tells the requesting SMSc where to send MT-ForwardSM messages
smsc_service_center_gt_address: "5551234567",  # Required

# MSISDN ↔ IMSI Mapping Configuration
# PLMN prefix: MCC (001 = Test Network) + MNC (01 = Test Operator)
hlr_imsi_plmn_prefix: "001001",                  # Only config parameter needed!
```

MSISDN ↔ IMSI Mapping

Configuration Parameters:

These parameters control how OmniSS7 generates synthetic IMSIs from MSISDNs for SRI-for-SM responses:

- **`hlr_imsi_plmn_prefix`**: The MCC+MNC prefix to use when constructing synthetic IMSIs (e.g., "50557" for MCC=505, MNC=57)
- **`hlr_msisdn_country_code`**: Country code to prepend when doing reverse IMSI→MSISDN mapping (e.g., "61" for Australia, "1" for USA/Canada)
- **`hlr_msisdn_nsn_offset`**: Character position where the National Subscriber Number (NSN) starts within the MSISDN (typically 0 if MSISDN doesn't include country code, or length of country code if it does)
- **`hlr_msisdn_nsn_length`**: Number of digits to extract from the MSISDN as the NSN

For additional configuration details, see MSISDN ↔ IMSI Mapping in Configuration Reference.

Why is MSISDN to IMSI Mapping Needed?

The MAP protocol for **SendRoutingInfoForSM** (SRI-for-SM) requires the HLR to return an **IMSI** (International Mobile Subscriber Identity) in its response. However, the requesting SMSc only knows the subscriber's **MSISDN** (phone number).

In a traditional network:

- The SMSc sends SRI-for-SM with the destination MSISDN (e.g., "5551234567")
- The HLR must look up the subscriber in its database to find their IMSI
- The HLR returns the IMSI in the SRI-for-SM response
- The SMSc then uses this IMSI when sending MT-ForwardSM to the MSC/VLR

OmniSS7's Approach - Synthetic IMSIs:

Instead of maintaining a full subscriber database with MSISDN-to-IMSI mappings, OmniSS7 uses a simple encoding scheme to **calculate** synthetic IMSIs directly from the MSISDN. This approach provides two key benefits:

1. **Privacy**: Real subscriber IMSIs stored in the HLR database are never exposed in SRI-for-SM responses sent over the SS7 network
2. **Simplicity**: No need to query the HLR database for IMSI lookups during SRI-for-SM operations - the IMSI is calculated on-the-fly from the MSISDN

How It Works:

MSISDNs are encoded directly into the subscriber portion of the IMSI (the digits after MCC+MNC):

```
IMSI = PLMN_PREFIX + zero_padded_MSISDN
```

Where:

- **PLMN_PREFIX**: MCC + MNC (e.g., "001001" for Test Network)
- **MSISDN**: All numeric digits from the phone number
- **Zero Padding**: Left-padded with zeros to fill IMSI to exactly 15 digits

Step-by-Step Example:

```
# Configuration
plmn_prefix = "001001"  # MCC 001 + MNC 01

# Input: MSISDN from SRI-for-SM request (TBCD decoded)
msisdn = "555123456"    # 9 digits

# Step 1: Calculate available space for subscriber number
subscriber_digits = 15 - String.length("001001")  # = 9 digits

# Step 2: Left-pad MSISDN with zeros to fill subscriber portion
padded_msisdn = String.pad_leading("555123456", 9, "0")  # = "555123456" (no padding needed)

# Step 3: Concatenate PLMN prefix + padded MSISDN
imsi = "001001" <> "555123456"  # = "001001555123456" (exactly 15 digits)
```

Complete Examples:

Input MSISDN	PLMN Prefix	Subscriber Digits Available	Padded MSISDN	Final IMSI	Notes
"555123456"	"001001"	(6) 9	"555123456"	"001001555123456"	Exact fit, no padding
"99"	"001001"	(6) 9	"000000099"	"001001000000099"	Left-padded with zeros
"999999999"	"001001"	(6) 9	"999999999"	"001001999999999"	Exact fit
"91123456789"	"001001"	(6) 9	"555123456"	"001001555123456"	Too long, rightmost 9 digits kept

Edge Case Handling:

- **Short MSISDNs**: Left-padded with zeros (e.g., "99" → "000000099")
- **Long MSISDNs**: Rightmost digits are kept, leftmost digits are truncated (e.g., "91123456789" → "555123456")
- **IMSI Length**: Always exactly 15 digits

Reverse Mapping (IMSI → MSISDN):

The SMSc can reverse this mapping to convert IMSIs back to MSISDNs:

```
# Input: IMSI from SRI-for-SM response
imsi = "001001555123456"

# Step 1: Strip PLMN prefix
plmn_prefix = "001001"
subscriber_portion = String.slice(imsi, 6, 9)  # = "555123456"

# Step 2: Remove leading zeros to get actual MSISDN
msisdn = String.replace_leading(subscriber_portion, "0", "")  # = "555123456"
```

Reverse Mapping Examples:

Input IMSI	PLMN Prefix	Subscriber Portion	Remove Leading Zeros	Final MSISDN
"001001555123456"	"001001"	"555123456"	"555123456"	"555123456"
"001001000000099"	"001001"	"000000099"	"99"	"99"
"001001999999999"	"001001"	"999999999"	"999999999"	"999999999"

Properties of This Mapping:

- ✅ **Deterministic**: Same MSISDN always produces same IMSI
- ✅ **Reversible**: Can convert back from IMSI to MSISDN
- ✅ **Minimal Configuration**: Only requires `hlr_imsi_plmn_prefix`
- ✅ **Privacy-Preserving**: Real IMSIs never exposed
- ✅ **No Database Lookup**: Fast calculation, no API calls needed
- ✅ **Always 15 Digits**: IMSI is always exactly 15 digits

MSISDN Input Handling:

When the HLR receives a SRI-for-SM request, the MSISDN undergoes TBCD decoding:

1. **TBCD Decode**: Convert binary TBCD to string (may include TON/NPI prefix like "91")
2. **Extract Digits**: Keep only numeric digits, strip any non-digit characters
3. **Normalize**: If longer than available space, take rightmost digits; if shorter, left-pad with zeros
4. **Encode**: Concatenate PLMN prefix + normalized MSISDN

Security Considerations:

The synthetic IMSIs returned in SRI-for-SM responses are purely for routing purposes. They are NOT the real IMSIs stored in the HLR subscriber database. This provides an additional layer of privacy protection, as real subscriber IMSIs are only exposed when absolutely necessary (e.g., during UpdateLocation or SendAuthenticationInfo operations that require real authentication vectors).

Response Flow:

```
1. SMSc → HLR: SRI-for-SM Request
   - MSISDN (TBCD): "91123456789" (includes TON/NPI)

2. HLR Processing:
   - TBCD decode: "91123456789"
   - Extract digits: "91123456789" (11 digits)
   - Fit to 9 digits: "555123456" (rightmost 9)
   - Add PLMN: "001001" + "555123456" = "001001555123456"
   - Get SMSC GT from config: "5551234567"

3. HLR → SMSc: SRI-for-SM Response
   - IMSI: "001001555123456" (synthetic, always 15 digits)
   - Network Node Number: "5551234567" (where to send MT-ForwardSM)

4. SMSc sends MT-ForwardSM to "5551234567" with IMSI "001001555123456"
```

Configuration:

The following parameters are used in `runtime.exs`:

```
# PLMN prefix: MCC (001 = Test Network) + MNC (01 = Test Operator)
hlr_imsi_plmn_prefix: "001001",

# NSN Extraction (if MSISDNs include country code)
hlr_msisdn_country_code: "1",       # Used for reverse mapping (IMSI→MSISDN)
hlr_msisdn_nsn_offset: 1,           # Skip 1-digit country code
hlr_msisdn_nsn_length: 10           # Extract 10-digit NSN
```

NSN Extraction Configuration:

If your MSISDNs include the country code (e.g., "68988000088" instead of just "88000088"), you must configure NSN extraction:

- **`hlr_msisdn_nsn_offset`**: Position where NSN starts (typically the length of your country code)
- **`hlr_msisdn_nsn_length`**: Number of digits in the NSN

Examples:

Example	Country Code	MSISDN Example	nsn_offset	nsn_length	NSN Extracted
1-digit CC	"9"	"95551234567"	1	10	"5551234567"
2-digit CC	"99"	"99412345678"	2	9	"412345678"
3-digit CC	"999"	"99988000088"	3	8	"88000088"

How It Works:

1. **MSISDN → IMSI**: Extract NSN from MSISDN, pad with leading zeros, concatenate with PLMN prefix

   ```
   MSISDN: "99988000088"
   NSN: String.slice("99988000088", 3, 8) = "88000088"
   Padded NSN: "088000088" (9 digits)
   IMSI: "547050" + "088000088" = "547050088000088"
   ```

2. **IMSI → MSISDN**: Strip PLMN prefix, remove leading zeros, prepend country code

   ```
   IMSI: "547050088000088"
   Subscriber portion: "088000088"
   Remove zeros: "88000088"
   MSISDN: "+999" + "88000088" = "+99988000088"
   ```

API Requirements: None - SRI-for-SM uses calculated values and config only. No backend API calls are required.

Field Source Summary

Source Type	Description	Examples
OmniHSS API	Dynamic data from OmniHSS subscriber database	IMSI, MSISDN, serving VLR/MSC from circuit_session
runtime.exs	OmniSS7 configuration parameters	`smsc_service_center_gt_address`, `camel_service_key`, `isd_network_access_mode`
Static	Hardcoded values in response generator	Subscriber status, bearer services, SS codes
Request	Fields extracted from incoming MAP request	IMSI from UpdateLocation, MSISDN from SRI
Calculated	Derived values using logic	Synthetic IMSI in SRI-for-SM (hlr_imsi_prefix + NSN)

Configuration Dependencies

Required in runtime.exs:

- `hlr_service_center_gt_address` - Used in UpdateLocation responses
- `smsc_service_center_gt_address` - Used in SRI-for-SM responses (where MT-ForwardSM should be routed)

Optional in runtime.exs (with defaults):

- `camel_service_key` - Default: `11_110`
- `camel_trigger_detection_point` - Default: `:termAttemptAuthorized`
- `isd_network_access_mode` - Default: `:packetAndCircuit`
- `isd_send_ss_data` - Default: `true`
- `isd_send_call_barring` - Default: `true`
- `hlr_imsi_plmn_prefix` - Default: `"001001"` (PLMN prefix for MSISDN↔IMSI mapping)

Required from OmniHSS:

OmniHSS must provide REST API endpoints for:

- Subscriber lookup by IMSI and MSISDN
- Circuit session location updates (VLR/MSC assignment)
- Authentication vector generation
- Subscriber status and service profile queries

Related Documentation

OmniSS7 Documentation:

- ← Back to Main Documentation
- Common Features Guide
- MAP Client Guide
- Technical Reference
- Configuration Reference

OmniHSS Documentation: For subscriber management, provisioning, authentication configuration, and administrative operations, refer to the **OmniHSS product documentation**. OmniHSS contains all the subscriber database logic, authentication algorithms, service provisioning rules, and Multi-IMSI management capabilities.

MAP Client Configuration Guide

← Back to Main Documentation

This guide provides detailed configuration for using OmniSS7 as a **MAP Client** to send MAP protocol requests to network elements.

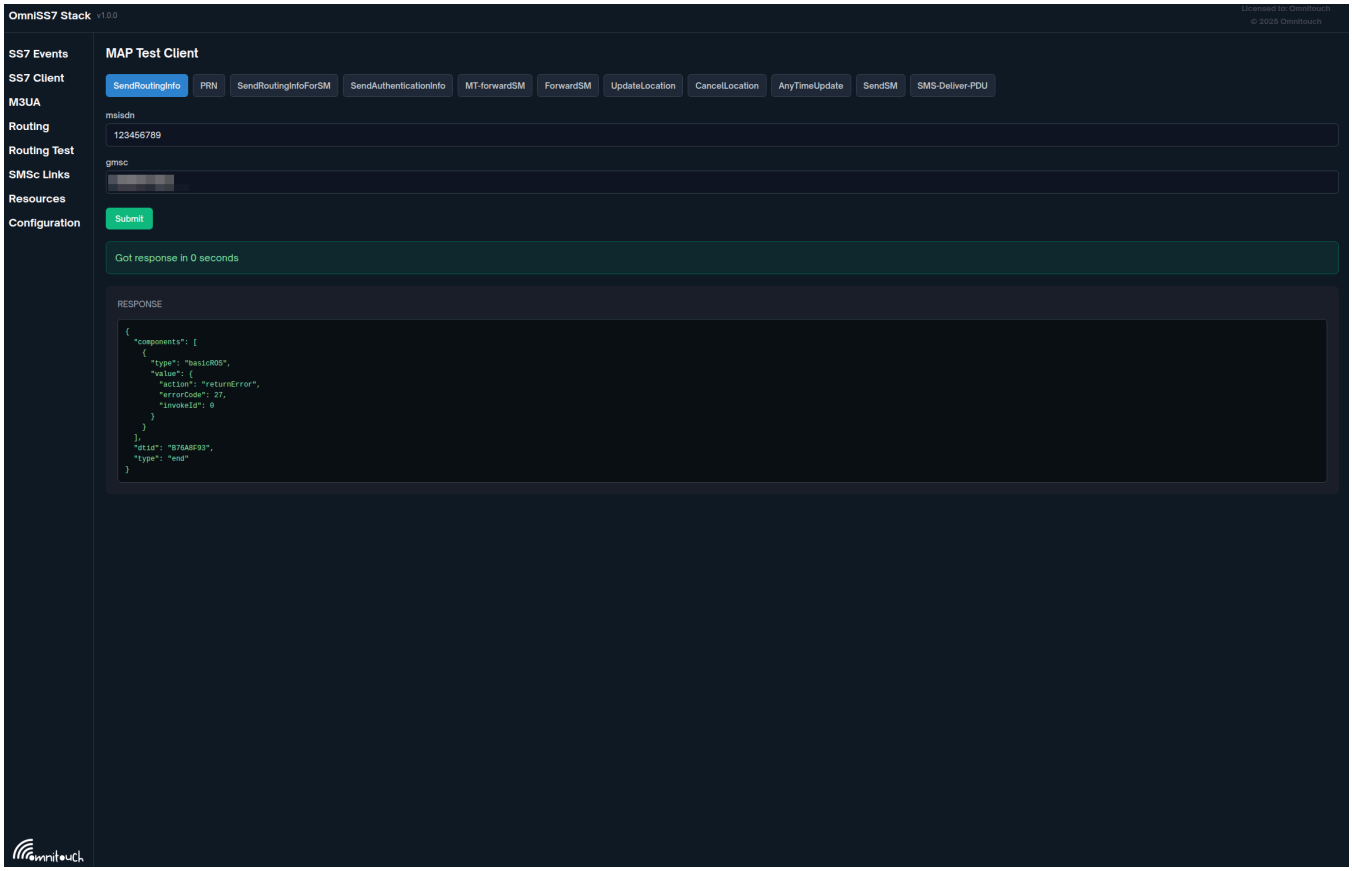


Table of Contents

What is MAP Client Mode?

MAP Client Mode allows OmniSS7 to connect as an **Application Server Process (ASP)** to an M3UA peer (STP or SGP) and send/receive **MAP (Mobile Application Part)** messages for services like:

- **HLR Queries**: SRI (Send Routing Info), SRI-for-SM, Authentication Info
- **Location Updates**: Update Location, Cancel Location
- **Subscriber Management**: Provide Roaming Number (PRN), Insert Subscriber Data

Network Architecture

Enabling MAP Client Mode

Edit `config/runtime.exs` and configure MAP client settings. For complete configuration reference, see M3UA Connection Parameters in Configuration Reference.

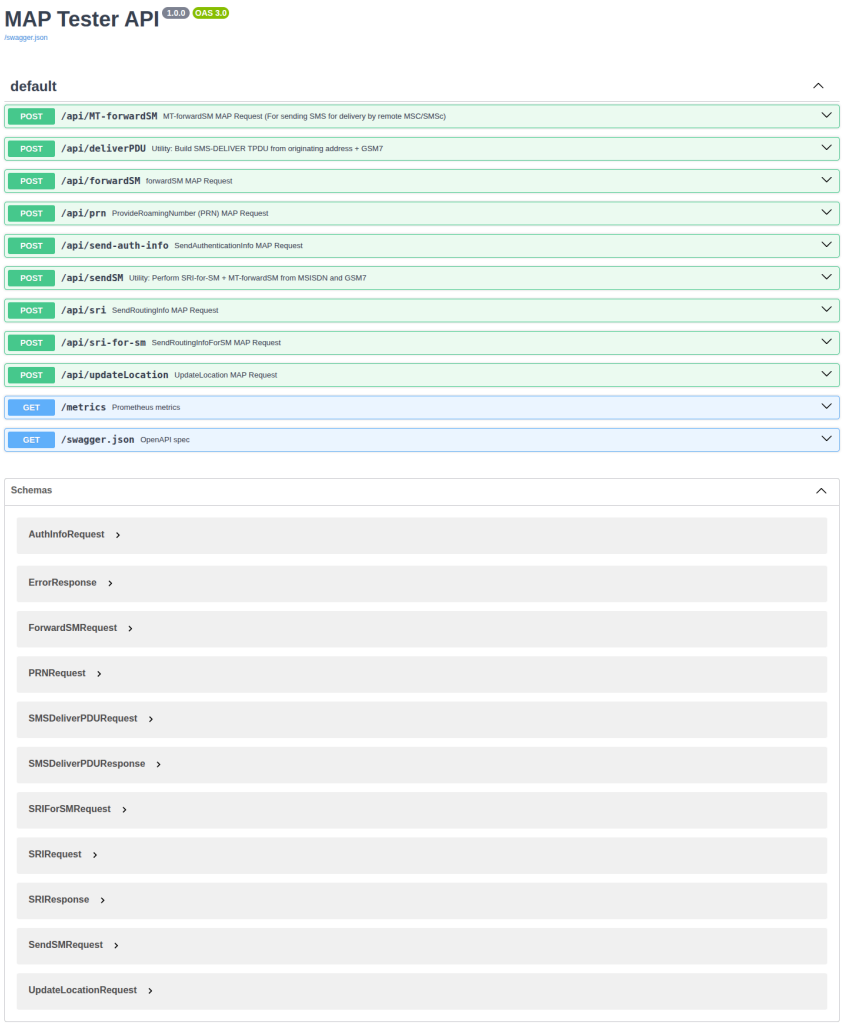
Basic Configuration

```
config :omniss7,
  # Enable MAP Client mode
  map_client_enabled: true,

  # M3UA Connection for MAP Client (connects as ASP to remote STP/SGP)
  map_client_m3ua: %{
    mode: "ASP",                        # M3UA mode: "ASP" (client) or "SGP" (server)
    callback: {MapClient, :handle_payload, []},  # Callback for incoming messages
    process_name: :map_client_asp,    # Registered process name
    local_ip: {10, 0, 0, 100},       # Local IP address
    local_port: 2905,                  # Local SCTP port
    remote_ip: {10, 0, 0, 1},        # Remote STP/SGP IP
    remote_port: 2905,                 # Remote STP/SGP port
    routing_context: 1                 # M3UA routing context
  }
```

Production Configuration Example

```
config :omniss7,
  # Enable MAP Client for production
  map_client_enabled: true,

  # Production M3UA connection
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},        # Production STP IP
    remote_port: 2905,
    routing_context: 1
  }

config :control_panel,
  web: %{
    listen_ip: "0.0.0.0",
    port: 443,
    hostname: "ss7-gateway.example.com",
    enable_tls: true,
    tls_cert: "/etc/ssl/certs/gateway.crt",
    tls_key: "/etc/ssl/private/gateway.key"
  }
```

Available MAP Operations



1. Send Routing Info for SM (SRI-for-SM)

Queries the HLR to determine the serving MSC for SMS delivery. For detailed information on how the HLR processes SRI-for-SM requests, see SRI-for-SM in HLR Guide.

API Endpoint: `POST /api/sri-for-sm`

Request:

```
{
  "msisdn": "447712345678",
  "serviceCenter": "447999123456"
}
```

Response:

```
{
  "result": {
    "imsi": "234509876543210",
    "locationInfoWithLMSI": {
      "networkNode-Number": "447999555111"
    }
  }
}
```

cURL Example:

```
curl -X POST http://localhost/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "447712345678",
    "serviceCenter": "447999123456"
  }'
```

---

2. Send Routing Info (SRI)

Queries the HLR for voice call routing information.

API Endpoint: `POST /api/sri`

Request:

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456"
}
```

Response:

```
{
  "result": {
    "imsi": "234509876543210",
    "extendedRoutingInfo": {
      "routingInfo": {
        "roamingNumber": "447999555222"
      }
    }
  }
}
```

---

3. Provide Roaming Number (PRN)

Requests a temporary roaming number (MSRN) from the serving MSC.

API Endpoint: `POST /api/prn`

Request:

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456",
  "msc_number": "447999555111",
  "imsi": "234509876543210"
}
```

---

4. Send Authentication Info

Requests authentication vectors from the HLR for subscriber authentication.

API Endpoint: `POST /api/send-auth-info`

Request:

```
{
  "imsi": "234509876543210",
  "vectors": 5
}
```

Response:

```
{
  "result": {
    "authenticationSetList": [
      {
        "rand": "0123456789ABCDEF0123456789ABCDEF",
        "xres": "ABCDEF0123456789",
        "ck": "0123456789ABCDEF0123456789ABCDEF",
        "ik": "FEDCBA9876543210FEDCBA9876543210",
        "autn": "0123456789ABCDEF0123456789ABCDEF"
      }
    ]
  }
}
```

---

5. Update Location

Registers a subscriber's current location with the HLR. For detailed information on UpdateLocation processing and InsertSubscriberData sequences, see Location Updates in HLR Guide.

API Endpoint: `POST /api/updateLocation`

Request:

```
{
  "imsi": "234509876543210",
  "vlr": "447999555111"
}
```

MAP Operations Summary

Sending Requests via API

Using Swagger UI

The Swagger UI provides an interactive interface for sending SS7 requests.

Access Swagger UI:

1. Navigate to `http://your-server/swagger`
2. Browse the available API endpoints
3. Click on any endpoint to expand its details

Sending a Request:

1. Click on the endpoint you want to use (e.g., `/api/sri-for-sm`)
2. Click the "Try it out" button
3. Fill in the required parameters in the request body
4. Click "Execute"
5. View the response below

API Response Codes

- **200** - Success, result returned in response body
- **400** - Bad Request, invalid parameters
- **504** - Gateway Timeout, no response from SS7 network within 10 seconds

---

MAP Client Metrics

Available Metrics

Request Metrics:

- `map_requests_total` - Total number of MAP requests sent
  - Labels: `operation` (values: `sri`, `sri_for_sm`, `prn`, `authentication_info`, etc.)
- `map_request_errors_total` - Total number of MAP request errors
  - Labels: `operation`
- `map_request_duration_milliseconds` - Histogram of MAP request durations
  - Labels: `operation`
- `map_pending_requests` - Current number of pending MAP requests (gauge)

Example Prometheus Queries

```
# Total SRI-for-SM requests in the last hour
increase(map_requests_total{operation="sri_for_sm"}[1h])

# Average response time for SRI requests
rate(map_request_duration_milliseconds_sum{operation="sri"}[5m]) /
rate(map_request_duration_milliseconds_count{operation="sri"}[5m])

# Error rate for all MAP operations
sum(rate(map_request_errors_total[5m])) by (operation)

# Current pending requests
map_pending_requests
```

Troubleshooting MAP Client

Issue: Requests Timeout

Symptoms:

- API returns 504 Gateway Timeout
- No response from HLR/MSC

Checks:

1. Verify M3UA connection is ACTIVE:

   ```
   # In IEx console
   :sys.get_state(:map_client_asp)
   ```

2. Check network connectivity to STP
3. Verify routing context and SCCP addressing
4. Check logs for SCCP errors

Issue: SCCP Errors

Symptoms:

- API returns SCCP error responses
- Logs show "SCCP unitdata service" messages

Common SCCP Error Codes:

- **No Translation**: Global Title not found in STP routing table
- **Subsystem Failure**: Destination subsystem (HLR SSN 6) is unavailable
- **Network Failure**: Network congestion or failure

Solutions:

- Contact STP administrator to verify routing configuration
- Verify destination Global Title is reachable
- Check if destination subsystem is operational

Related Documentation

- ← Back to Main Documentation
- Common Features Guide - Web UI, API, Monitoring
- STP Guide - Routing configuration
- SMS Center Guide - SMS delivery
- Technical Reference - Protocol specifications

SMS Center (SMSc) Configuration Guide

← Back to Main Documentation

This guide provides detailed configuration for using OmniSS7 as an **SMS Center (SMSc)** frontend with **OmniMessage** as the backend message store and delivery platform.

OmniMessage Integration

OmniSS7 SMSc mode functions as an SS7 signaling frontend that interfaces with **OmniMessage**, a carrier-grade SMS platform. This architecture separates concerns:

- **OmniSS7 (SMSc Frontend)**: Handles all SS7/MAP protocol signaling, SCCP routing, and network communication
- **OmniMessage (SMS Backend)**: Manages message storage, queuing, retry logic, delivery tracking, and routing decisions

Why OmniMessage?

OmniMessage provides carrier-grade SMS messaging capabilities with features including:

- **Message Queue Management**: Persistent storage with configurable retry logic and priority queuing
- **Delivery Tracking**: Real-time delivery status, delivery reports (DLR), and failure reason tracking
- **Multi-SMSc Support**: Multiple frontend instances can connect to a single OmniMessage backend for load balancing and redundancy
- **Routing Intelligence**: Advanced routing rules based on destination, sender, message content, and time of day
- **Rate Limiting**: Per-route TPS (transactions per second) controls to prevent network congestion
- **API-First Design**: RESTful HTTP API for integration with billing systems, customer portals, and third-party applications
- **Analytics & Reporting**: Message volume statistics, delivery success rates, and performance metrics

All message data, delivery state, and routing configurations are stored and managed in OmniMessage. OmniSS7 queries OmniMessage via HTTPS API calls to retrieve pending messages, update delivery status, and register as an active frontend.

Important: OmniSS7 SMSc mode is a **signaling frontend only**. All message routing logic, queue management, retry algorithms, delivery tracking, and business rules are handled by OmniMessage. This guide covers the SS7/MAP protocol configuration in OmniSS7. For information about message routing, queue configuration, delivery reports, rate limiting, and analytics, **refer to the OmniMessage documentation**.

Table of Contents

What is SMS Center Mode?

Note: This section covers OmniSS7's SS7 signaling configuration only. For message routing rules, queue management, delivery tracking, and business logic configuration, see the **OmniMessage product documentation**.

SMS Center Mode enables OmniSS7 to function as an SMSc for:

- **MT-SMS Delivery**: Mobile-Terminated SMS delivery to subscribers
- **MO-SMS Handling**: Mobile-Originated SMS reception and routing
- **Message Queuing**: Database-backed message queue with retry logic
- **Auto-Flush**: Automatic SMS delivery from queue
- **Delivery Reports**: Track message delivery status

SMS Center Architecture

Enabling SMSc Mode

OmniSS7 can operate in different modes. To use it as an SMSc, you need to enable SMSc mode in the configuration.

Switching to SMSc Mode

OmniSS7's `config/runtime.exs` contains three pre-configured operational modes. To enable SMSc mode:

1. **Open** `config/runtime.exs`
2. **Find** the three configuration sections (lines 53-204):
    - Configuration 1: STP Mode (lines 53-95)
    - Configuration 2: HLR Mode (lines 97-142)
    - Configuration 3: SMSc Mode (lines 144-204)
3. **Comment out** any other active configuration (add # to each line)
4. **Uncomment** the SMSc configuration (remove # from lines 144-204)
5. **Customize** the configuration parameters as needed
6. **Restart** the application: `iex -S mix`

SMSc Mode Configuration

The complete SMSc configuration looks like this:

```
config :omniss7,
  # Mode flags - Enable STP + SMSc features
  # Note: map_client_enabled is true because SMSc needs routing capabilities
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  # OmniMessage Backend API Configuration
  smsc_api_base_url: "https://10.179.3.219:8443",
  # SMSc identification for registration with backend
  smsc_name: "ipsmgw",
  # Service Center GT Address for SMS operations
  smsc_service_center_gt_address: "5551234567",

  # Auto Flush Configuration (background SMS queue processing)
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  # M3UA Connection Configuration
  # Connect as ASP for sending/receiving MAP SMS operations
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    # Local endpoint (SMSc system)
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    # Remote STP endpoint
    remote_ip: {10, 179, 4, 10},
```

```
    remote_port: 2905,
    routing_context: 1
  }

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.TestClientLive, "/client", "SS7 Client"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.RoutingLive, "/routing", "Routing"},
    {SS7.Web.RoutingTestLive, "/routing_test", "Routing Test"},
    {SS7.Web.SmscLinksLive, "/smsc_links", "SMSc Links"}
  ],
  page_order: ["/events", "/client", "/m3ua", "/routing", "/routing_test", "/smsc_links",
"/application", "/configuration"]
```

Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

Parameter	Type	Default	Description	Example
`smsc_api_base_url`	String	*Required*	**OmniMessage** backend API endpoint	`"https://10.179.3.219:8443"`
`smsc_name`	String	`"{hostname}_SMSc"`	Your SMSc identifier for registration	`"ipsmgw"`
`smsc_service_center_gt_address`	String	*Required*	Service Center Global Title	`"5551234567"`
`auto_flush_enabled`	Boolean	`true`	Enable automatic queue processing	`false`
`auto_flush_interval`	Integer	`10_000`	Queue processing interval in milliseconds	`5_000`
`auto_flush_dest_smsc`	String	*Required*	Destination SMSC name for auto-flush	`"ipsmgw"`
`auto_flush_tps`	Integer	`10`	Message processing rate (transactions/second)	`20`
`local_ip`	Tuple	*Required*	Your SMSc system's IP address	`{10, 179, 4, 12}`
`local_port`	Integer	`2905`	Local SCTP port	`2905`
`remote_ip`	Tuple	*Required*	STP IP address for SS7 connectivity	`{10, 179, 4, 10}`
`remote_port`	Integer	`2905`	Remote SCTP port	`2905`
`routing_context`	Integer	`1`	M3UA routing context ID	`1`

What Happens When SMSc Mode is Enabled

When `smsc_mode_enabled: true` and `map_client_enabled: true`, the web UI will show:

- 🔀 **SS7 Events** - Event logging
- 🔀 **SS7 Client** - MAP operation testing
- 🔀 **M3UA** - Connection status

- 🔀 **Routing** - Route table management (STP enabled)
- 🔀 **Routing Test** - Route testing (STP enabled)
- 🔀 **SMSc Links** - SMSc API status + SMS queue management ← *SMSc-specific*
- 🔀 **Resources** - System monitoring
- 🔀 **Configuration** - Config viewer

The **HLR Links** tab will be hidden.

Important Notes

- SMSc mode requires `map_client_enabled: true` for routing capabilities
- **OmniMessage Backend**: The OmniMessage API backend must be accessible at the configured `smsc_api_base_url`
- **Frontend Registration**: The system automatically registers with OmniMessage every **5 minutes** via the `SMS.FrontendRegistry` module
- **API Request Timeout**: All OmniMessage API requests have a **hardcoded 5-second timeout**
- **MAP Request Timeout**: All MAP requests (SRI-for-SM, MT-ForwardSM, etc.) have a **hardcoded 10-second timeout**
- Auto-flush automatically processes the SMS queue in the background
- M3UA connection to STP is required for sending/receiving MAP SMS operations
- After changing modes, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for information on using the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

HTTP API Configuration

OmniMessage Backend Setup

OmniSS7 communicates with OmniMessage via HTTPS REST API to manage message delivery, track subscriber state, and register as an active frontend:

```
config :omniss7,
  # OmniMessage API base URL
  smsc_api_base_url: "https://10.5.198.200:8443",
  # SMSC name identifier for registration (defaults to hostname_SMSc if empty)
  smsc_name: "omni-smsc01",
  # Service Center GT Address for SMS operations
  smsc_service_center_gt_address: "5551234567"
```

Configuration Parameters:

Parameter	Type	Required	Default	Description
`smsc_api_base_url`	String	Yes	`"https://localhost:8443"`	Base URL for OmniMessage API
`smsc_name`	String	No	`""` (uses `"{hostname}_SMSc"`)	SMSC identifier for registration and queue management
`smsc_service_center_gt_address`	String	No	`"5551234567"`	Service Center GT Address returned in SRI-for-SM responses. This tells other network elements where to route MT-ForwardSM messages. See SRI-for-SM Guide for details.

Frontend Registration

The system automatically registers itself with OmniMessage on startup and **re-registers every 5 minutes** via the `SMS.FrontendRegistry` module. This allows OmniMessage to:

- Track active frontends for load balancing
- Monitor uptime and health status
- Collect configuration information

- Manage distributed SMS routing across multiple frontends

Implementation Details:

- **Registration Interval**: 5 minutes (hardcoded)
- **Process**: Started automatically when `smsc_mode_enabled: true`

Registration Payload:

```
{
  "frontend_name": "omni-smsc01",
  "configuration": "{...}",
  "frontend_type": "SS7",
  "hostname": "smsc-server01",
  "uptime_seconds": 12345
}
```

Note: The frontend name is taken from the `smsc_name` configuration parameter. If not set, it defaults to `"{hostname}_SMSc"`.

OmniMessage API Communication

When OmniSS7 receives MAP operations from the SS7 network or processes the message queue, it communicates with OmniMessage to:

- **Register as an active frontend** and report health status
- **Submit mobile-originated (MO) messages** received from subscribers
- **Retrieve mobile-terminated (MT) messages** from the queue for delivery
- **Update delivery status** with success/failure reports
- **Query routing information** for message forwarding

Endpoint	Method	Purpose	Request Body
`/api/frontends`	POST	Register frontend instance	`{"frontend_name": "...", "frontend_type": "SMSc", "hostname": "...", "uptime_seconds": ...}`
`/api/messages_raw`	POST	Insert new SMS message	`{"source_msisdn": "...", "source_smsc": "...", "message_body": "..."}`
`/api/messages`	GET	Get message queue	Header: `smsc: <smsc_name>`
`/api/messages/{id}`	PATCH	Mark message as delivered	`{"deliver_time": "...", "dest_smsc": "..."}`
`/api/messages/{id}`	PUT	Update message status	`{"dest_smsc": null}`
`/api/locations`	POST	Insert/update subscriber location	`{"msisdn": "...", "imsi": "...", "location": "...", "ims_capable": true, "csfb": false, "expires": "...", "user_agent": "...", "ran_location": "...", "imei": "...", "registered": "..."}`
`/api/events`	POST	Add event tracking	`{"message_id": ..., "name": "...", "description": "..."}`
`/api/status`	GET	Health check	-

API Response Format

All API responses use JSON format with the following conventions:

- **Success responses**: HTTP 200-201 with JSON body containing result data
- **Error responses**: HTTP 4xx/5xx with error details in response body
- **Timestamps**: ISO 8601 format (e.g., `"2025-10-21T12:34:56Z"`)
- **Message IDs**: Integer or string identifiers

API Client Modules

The SMS system consists of three main modules:

1. SMSc.APIClient

Main API client module providing all HTTP API communication with OmniMessage:

- `frontend_register/4` - Register frontend with OmniMessage
- `insert_message/3` - Insert raw SMS message (Python-compatible 3-parameter version)
- `insert_location/9` - Insert/update subscriber location data
- `get_message_queue/2` - Retrieve pending messages from queue
- `mark_dest_smsc/3` - Mark message as delivered or failed
- `add_event/3` - Add event tracking for messages
- `flush_queue/2` - Process pending messages (SRI-for-SM + MT-forwardSM)
- `auto_flush/2` - Continuous queue processing loop

2. SMS.FrontendRegistry

Handles periodic frontend registration with the backend:

- Automatically registers on startup
- Re-registers every 5 minutes
- Uses `smsc_name` from config (falls back to hostname)
- Collects system configuration and uptime information

3. SMS.Utils

Utility functions for SMS operations:

- `generate_tp_scts/0` - Generate SMS timestamp in TPDU format

SMS Message Flows

Incoming SMS Flow (Mobile-Originated)

Outgoing SMS Flow (Mobile-Terminated)

Key Steps Explained:

- **SRI-for-SM Request**: The SMSc queries the HLR with the destination MSISDN to determine where to route the SMS message. The HLR responds with:
  - A synthetic IMSI (calculated from the MSISDN for privacy) - see MSISDN ↔ IMSI Mapping
  - The SMSC GT address (network node number) where the MT-ForwardSM should be sent
  - For complete details on how this works, see SRI-for-SM in HLR Guide
- **MT-forwardSM Request**: Once routing info is obtained, the SMSc sends the actual SMS message to the MSC/VLR serving the subscriber

SMS TPDU Structure

Alert Service Center Handling

The SMSc can receive **alertServiceCenter** messages from the HLR to track subscriber reachability status.

For information on how the HLR sends alertServiceCenter messages, see Alert Service Center Integration in HLR Guide.

What is alertServiceCenter?

When a subscriber performs an UpdateLocation at the HLR (i.e., registers with a new VLR/MSC), the HLR can

notify SMSc systems that the subscriber is now reachable by sending an **alertServiceCenter** (MAP opcode 64) message.

Configuration

The location expiry time is configured in the HLR:

```
config :omniss7,
  # Location expiry time when SMSc receives alertServiceCenter (default: 48 hours)
  hlr_alert_location_expiry_seconds: 172800
```

Behavior

When the SMSc receives an alertServiceCenter message:

1. **Decode MSISDN**: Extract the subscriber's MSISDN from the message (TBCD format)
2. **Strip TON/NPI prefix**: Remove common prefixes like "19", "11", "91" (e.g., "19123123213" → "123123213")
3. **Calculate IMSI**: Generate synthetic IMSI using same mapping as SRI-for-SM
4. **POST to /api/location**: Update location database with:
   - `msisdn`: Subscriber's phone number (cleaned)
   - `imsi`: Synthetic IMSI
   - `location`: SMSc name (e.g., "ipsmgw")
   - `expires`: Current time + `hlr_alert_location_expiry_seconds`
   - `csfb`: true (subscriber reachable via Circuit-Switched Fallback)
   - `ims_capable`: false (this is 2G/3G CS registration, not IMS/VoLTE)
   - `user_agent`: HLR GT that sent the alert (for tracking)
   - `ran_location`: "SS7"
5. **Track in SMSc Subscriber Tracker**: Record the subscriber with HLR GT, status=active, message counters at 0
6. **Send ACK**: Reply to HLR with alertServiceCenter acknowledgment

Absent Subscriber Handling

When the SMSc attempts to deliver a message and receives an "absent subscriber" error during SRI-for-SM (for more on SRI-for-SM, see SRI-for-SM in HLR Guide):

1. **Detect absence**: SRI-for-SM returns `absentSubscriberDiagnosticSM` error
2. **Expire location**: POST to /api/location with `expires=0` to mark subscriber as unreachable
3. **User agent**: Set to "SS7_AbsentSubscriber" to identify the source
4. **Update tracker**: Mark subscriber as `failed` in SMSc Subscriber Tracker

This ensures the location database and tracker accurately reflect subscriber reachability status.

Flow Diagram

API Endpoint

POST /api/location

```
{
  "msisdn": "15551234567",
  "imsi": "001010123456789",
  "location": "ipsmgw",
  "ims_capable": false,
  "csfb": true,
  "expires": "2025-11-01T12:00:00Z",
  "user_agent": "15551111111",
  "ran_location": "SS7",
  "imei": "",
  "registered": "2025-10-30T12:00:00Z"
}
```

Note: The `user_agent` field contains the HLR GT that sent the alertServiceCenter, allowing the SMSc to track which HLR is providing location updates.

For absent subscribers, `expires` is set to current time (immediate expiry).

Loop Prevention

The SMSc implements **automatic loop prevention** to avoid infinite message routing loops when messages originate from SS7 networks.

Why Loop Prevention is Important

When the SMSc receives mobile-originated (MO) SMS messages from the SS7 network, it inserts them into the message queue with a `source_smsc` field identifying their origin (e.g., `"SS7_GT_15551234567"`). Without loop prevention, these messages could be:

1. Received from SS7 network → Queued with `source_smsc` containing "SS7"
2. Retrieved from queue → Processed for delivery
3. Sent back to SS7 network → Creating a loop

How It Works

The SMSc automatically detects and prevents loops during message processing:

Implementation

When processing messages from the queue, the SMSc checks the `source_smsc` field:

- **If `source_smsc` contains "SS7"**:
  - Message is skipped
  - Event added: `"Loop Prevention"` with description explaining the skip reason
  - Message marked as failed via PUT request
  - Logged with warning level
- **Otherwise**:
  - Message processed normally
  - SRI-for-SM and MT-ForwardSM operations proceed

Source SMSC Values

Messages can have various `source_smsc` values:

Source	Example Value	Action
SS7 Network (MO-FSM)	`"SS7_GT_15551234567"`	**Skipped** - Loop prevention
External API/SMPP	`"ipsmgw"` or `"api_gateway"`	Processed normally
Other SMSc	`"smsc-node-01"`	Processed normally

Event Tracking

When a message is skipped due to loop prevention, an event is recorded:

```
{
  "message_id": 12345,
  "name": "Loop Prevention",
  "description": "Message skipped - source_smsc 'SS7_GT_15551234567' contains 'SS7', preventing
message loop"
}
```

This event is visible in:

- **Web UI**: SS7 Events page (`/events`)
- **Database**: `events` table via API
- **Logs**: Warning level log entries

Configuration

Loop prevention is **always enabled** and cannot be disabled. This is a critical safety feature to prevent network disruption from message loops.

Example Scenario

Scenario: Mobile subscriber sends SMS via SS7 network

```
1. Mobile phone → MSC/VLR → SMSc (via MO-ForwardSM)
2. SMSc receives MO-FSM from GT 15551234567
3. SMSc inserts to queue: source_smsc = "SS7_GT_15551234567"
4. Auto-flush retrieves message from queue
5. SMSc detects "SS7" in source_smsc → SKIP
6. Event logged: "Loop Prevention"
7. Message marked as failed
8. No SRI-for-SM or MT-ForwardSM sent (loop prevented)
```

Without loop prevention, step 8 would send the message back to the SS7 network, potentially creating an infinite loop.

SMSc Subscriber Tracking

The SMSc includes a **Subscriber Tracker** GenServer that maintains real-time state for subscribers based on alertServiceCenter messages and message delivery attempts.

Purpose

The tracker provides:

- **Reachability monitoring**: Which subscribers are currently reachable
- **HLR tracking**: Which HLR sent the alertServiceCenter for each subscriber
- **Message counters**: Number of messages sent/received per subscriber
- **Failure tracking**: Mark subscribers as failed when delivery attempts fail
- **Web UI visibility**: Real-time dashboard showing all tracked subscribers

Tracked Information

For each subscriber, the tracker stores:

Field	Description	Example
`msisdn`	Subscriber's phone number (key)	"15551234567"
`imsi`	Subscriber's IMSI	"001010123456789"
`hlr_gt`	HLR GT that sent alertServiceCenter	"15551111111"
`messages_sent`	Count of MT-FSM messages sent	5
`messages_received`	Count of MO-FSM messages received	2
`status`	`:active` or `:failed`	`:active`
`updated_at`	Unix timestamp of last update	1730246400

State Transitions

Behavior

When alertServiceCenter is received:

- Create or update subscriber entry
- Set `status = :active`
- Record HLR GT
- Reset or preserve message counters

When SRI-for-SM succeeds:

- Increment `messages_sent` counter

- Update `updated_at` timestamp

When SRI-for-SM fails:

- Set `status = :failed`
- Keep in tracker for monitoring

When subscriber is removed:

- Delete from ETS table
- No longer appears in Web UI

Web UI - SMSc Subscribers Page

Path: `/smsc_subscribers` **Auto-refresh**: Every 2 seconds

Note: This page is only available when running in SMSc mode. After uncommenting the SMSc configuration in `config/runtime.exs`, you must restart the application for the route to become available.

The **SMSc Subscribers** page provides real-time monitoring of all tracked subscribers:

Features

1. **Subscriber Table**
   - MSISDN, IMSI, HLR GT
   - Messages sent/received counters
   - Status badge (Active/Failed) with color coding
   - Last updated timestamp and duration
   - Remove button for individual subscribers
2. **Summary Statistics**
   - Total tracked subscribers
   - Count of active subscribers
   - Count of failed subscribers
   - Number of unique HLRs
3. **Actions**
   - Clear All: Remove all tracked subscribers
   - Remove: Remove individual subscriber

Example View

```
┌──────────────────────────────────────────────────────────────────────────┐
│ SMSc Tracked Subscribers                              Total: 3           │
├──────────────────────────────────────────────────────────────────────────┤
│ MSISDN          IMSI              HLR GT          Msgs  Status           │
│                                                    S/R                   │
├──────────────────────────────────────────────────────────────────────────┤
│ 15551234567     001010123456789   15551111111    5/2   ● Active        │
│ 15559876543     001010987654321   15551111111    0/0   ● Active        │
│ 15551112222     001010111222233   15552222222    3/1   ○ Failed        │
└──────────────────────────────────────────────────────────────────────────┘

Summary:  Total: 3  |  Active: 2  |  Failed: 1  |  Unique HLRs: 2
```

API Functions

The tracker exposes these functions for programmatic access:

```
# Called when alertServiceCenter is received
SMSc.SubscriberTracker.alert_received(msisdn, imsi, hlr_gt)

# Increment message counters
```

```
SMSc.SubscriberTracker.message_sent(msisdn)
SMSc.SubscriberTracker.message_received(msisdn)

# Mark as failed (SRI-for-SM failure)
SMSc.SubscriberTracker.mark_failed(msisdn)

# Remove from tracking
SMSc.SubscriberTracker.remove_subscriber(msisdn)

# Query functions
SMSc.SubscriberTracker.get_active_subscribers()
SMSc.SubscriberTracker.get_subscriber(msisdn)
SMSc.SubscriberTracker.count_subscribers()
SMSc.SubscriberTracker.clear_all()
```

Integration

The tracker is automatically integrated with:

- **alertServiceCenter handler**: Calls `alert_received/3` on successful location update
- **SRI-for-SM handler**: Increments `messages_sent` on successful routing
- **Absent subscriber handler**: Calls `mark_failed/1` when subscriber is absent
- **Unknown subscriber errors**: Calls `mark_failed/1` when SRI-for-SM fails

---

Auto-Flush SMS Queue

The **Auto-Flush** service automatically processes pending SMS messages.

For configuration parameter reference, see Auto-Flush Configuration in Configuration Reference.

Configuration

```
config :omniss7,
  auto_flush_enabled: true,           # Enable/disable auto-flush
  auto_flush_interval: 10_000,        # Poll interval in milliseconds
  auto_flush_dest_smsc: nil,          # Filter: nil = all
  auto_flush_tps: 10                  # Max transactions per second
```

How It Works

1. **Polling**: Every `auto_flush_interval` milliseconds, queries API for pending messages
2. **Filtering**: Optionally filter by `auto_flush_dest_smsc`
3. **Rate Limiting**: Process up to `auto_flush_tps` messages per cycle
4. **Delivery**: For each message:
    - Send **SRI-for-SM** (Send Routing Info for Short Message) to HLR to get routing info
        - The HLR returns a synthetic IMSI calculated from the MSISDN
        - The HLR returns the SMSC GT address where MT-ForwardSM should be sent
        - See SRI-for-SM Details in HLR Guide for complete documentation
    - On success, send **MT-forwardSM** to MSC/VLR
    - Update message status via API (delivered/failed)
    - Add event tracking via API

> 🔷 **Technical Deep Dive**: For a complete explanation of how SRI-for-SM works, including MSISDN to IMSI mapping, service center GT address configuration, and the privacy-preserving synthetic IMSI generation, see the SRI-for-SM section in the HLR Configuration Guide.

SMSc Metrics

Available Metrics

SMS Queue Metrics:

- `smsc_queue_depth` - Current number of pending messages
- `smsc_messages_delivered_total` - Total messages successfully delivered
- `smsc_messages_failed_total` - Total messages that failed delivery
- `smsc_delivery_duration_milliseconds` - Histogram of delivery times

Example Queries:

```
# Current queue depth
smsc_queue_depth

# Delivery success rate (last 5 minutes)
rate(smsc_messages_delivered_total[5m]) /
(rate(smsc_messages_delivered_total[5m]) + rate(smsc_messages_failed_total[5m]))

# Average delivery time
rate(smsc_delivery_duration_milliseconds_sum[5m]) /
rate(smsc_delivery_duration_milliseconds_count[5m])
```

Troubleshooting SMSc

Issue: Messages Not Delivering

Checks:

1. Verify auto-flush is enabled
2. Check database connection
3. Monitor logs for errors
4. Verify M3UA connection is ACTIVE
5. Check TPS limits

Issue: High Queue Depth

Possible Causes:

- TPS limit too low
- HLR timeout issues
- Network connectivity problems
- Invalid destination numbers

Solutions:

- Increase `auto_flush_tps`
- Check HLR availability
- Review failed message logs

MT-forwardSM API

Send SMS via API

API Endpoint: `POST /api/MT-forwardSM`

Request:

```
{
  "imsi": "234509876543210",
  "destination_serviceCentre": "447999555111",
  "originating_serviceCenter": "447999123456",
  "smsPDU": "040B917477218345F60000157030185714 0C0BD4F29C0E9281C4E1F11A"
}
```

Response:

```
{
```

```
  "result": "success",
  "message_id": "12345"
}
```

Related Documentation

OmniSS7 Documentation:

- ← Back to Main Documentation
- HLR Configuration Guide - HLR mode setup and operations
  - **SRI-for-SM Technical Details** - Complete documentation on MSISDN to IMSI mapping and service center configuration
- Common Features Guide - Web UI, API, Monitoring
- MAP Client Guide - MAP operations
- Technical Reference - Protocol specifications

OmniMessage Documentation: For message routing configuration, queue management, delivery tracking, rate limiting, and analytics, refer to the **OmniMessage product documentation**. OmniMessage contains all the message routing logic, queue retry algorithms, delivery report handling, and business rules engine.

M3UA STP Configuration Guide

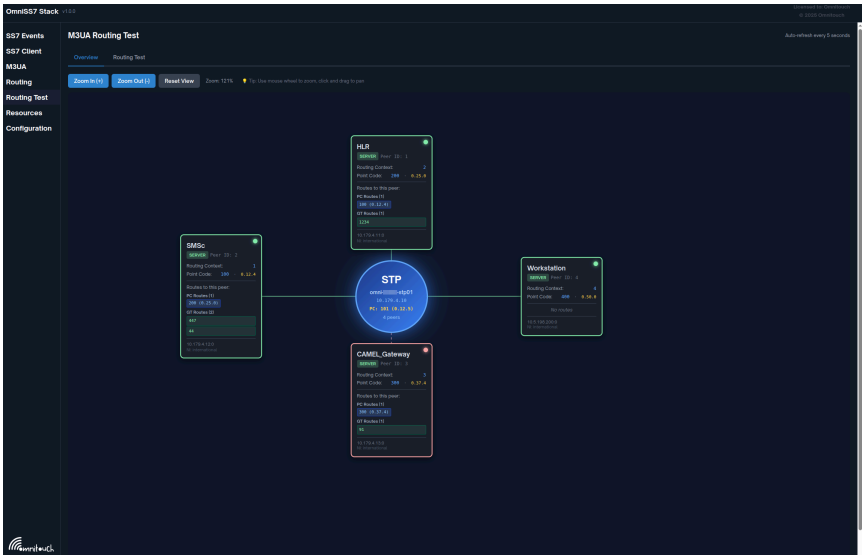
← Back to Main Documentation

This guide provides detailed configuration for using OmniSS7 as a **Signaling Transfer Point (STP)**.

Table of Contents

What is a Signaling Transfer Point (STP)?

A **Signaling Transfer Point (STP)** is a critical network element in SS7 and IP-based signaling networks that routes signaling messages between network nodes.



STP Functions

- **Message Routing**: Routes SS7 signaling traffic based on destination Point Code (PC) or Global Title (GT)
- **Protocol Translation**: Bridges traditional SS7 networks with IP-based M3UA/SCTP networks
- **Load Distribution**: Distributes traffic across multiple destinations using priority-based routing
- **Network Gateway**: Connects different signaling networks and service providers
- **Topology Hiding**: Can rewrite addresses to hide internal network topology

STP Network Diagram

STP Network Roles Explained

ASP (Application Server Process)

- **Role**: Client connecting to a remote SGP/STP
- **Direction**: Outbound connection
- **Use Case**: Your STP connects to a partner network's STP

SGP (Signaling Gateway Process)

- **Role**: Server accepting connections from ASPs
- **Direction**: Inbound connection
- **Use Case**: Partner networks connect to your STP

AS (Application Server)

- **Definition**: Logical grouping of one or more ASPs
- **Purpose**: Provides redundancy and load sharing
- **Use Case**: Multiple ASPs serving the same destination

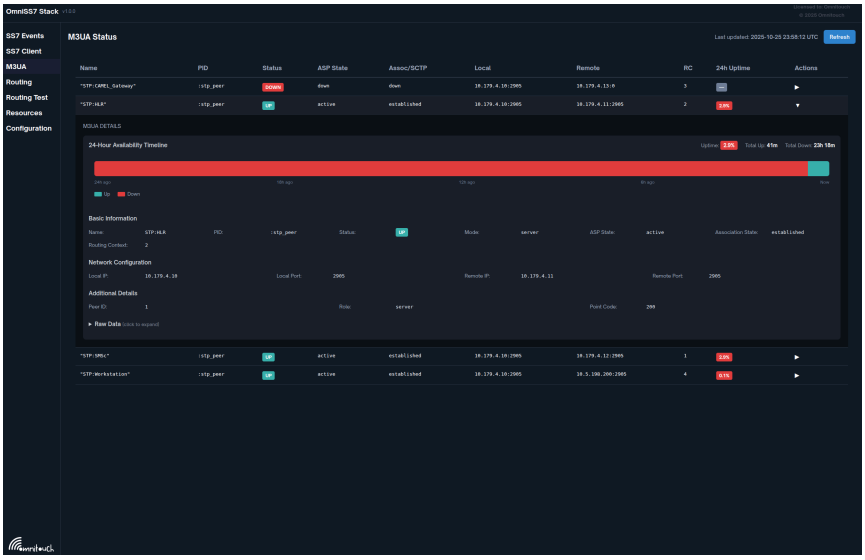
Enabling M3UA STP Mode

OmniSS7 can operate in different modes. To use it as an STP, you need to enable STP mode in the configuration.

Switching to STP Mode

OmniSS7's `config/runtime.exs` contains three pre-configured operational modes. To enable STP mode:

1. **Open** `config/runtime.exs`
2. **find** the three configuration sections (lines 53-174):
   - Configuration 1: STP Mode (lines 53-85)
   - Configuration 2: HLR Mode (lines 87-123)
   - Configuration 3: SMSc Mode (lines 125-174)
3. **Comment out** the currently active configuration (add `#` to each line)
4. **Uncomment** the STP configuration (remove `#` from lines 53-85)
5. **Customize** the configuration parameters as needed
6. **Restart** the application: `iex -S mix`



STP Mode Configuration

The complete STP configuration looks like this:

```
config :omniss7,
  # Mode flags - Enable STP features only
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,
```

```
# M3UA Connection Configuration
# Connect as ASP (Application Server Process) to remote STP/SGW
map_client_m3ua: %{
  mode: "ASP",
  callback: {MapClient, :handle_payload, []},
  process_name: :stp_client_asp,
  # Local endpoint (this system)
  local_ip: {10, 179, 4, 10},
  local_port: 2905,
  # Remote STP/SGW endpoint
  remote_ip: {10, 179, 4, 11},
  remote_port: 2905,
  routing_context: 1
}
```

Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

Parameter	Type	Default	Description	Example
`map_client_enabled`	Boolean	`true`	Enable MAP client and routing capabilities	`true`
`local_ip`	Tuple	*Required*	Your system's IP address	`{10, 179, 4, 10}`
`local_port`	Integer	2905	Local SCTP port	2905
`remote_ip`	Tuple	*Required*	Remote STP/SGW IP address	`{10, 179, 4, 11}`
`remote_port`	Integer	2905	Remote SCTP port	2905
`routing_context`	Integer	1	M3UA routing context ID	1
`enable_gt_routing`	Boolean	`false`	Enable Global Title routing (in addition to PC routing)	`true`

What Happens When STP Mode is Enabled

When `map_client_enabled: true`, the web UI will show:

- ✅ **SS7 Events** - Event logging
- ✅ **SS7 Client** - MAP operation testing
- ✅ **M3UA** - Connection status
- ✅ **Routing** - Route table management ← *STP-specific*
- ✅ **Routing Test** - Route testing ← *STP-specific*
- ✅ **Resources** - System monitoring
- ✅ **Configuration** - Config viewer

The **HLR Links** and **SMSc Links** tabs will be hidden.

Important Notes

- SCTP protocol (IP protocol 132) must be allowed through firewalls
- Default M3UA port is 2905 (industry standard)
- Ensure sufficient system resources for handling routing traffic
- **Routing Persistence**: All routes configured via the Web UI or API are stored in **Mnesia database** and **survive restarts**
- **Configuration Merge**: Routes from `runtime.exs` are loaded at startup and merged with Mnesia routes
- After changing modes, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for managing routes via the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

Standalone STP Mode

In addition to the STP routing capabilities available when `map_client_enabled: true`, you can run a **standalone M3UA STP server** that listens for incoming connections.

Enabling Standalone STP

Add this configuration to `config/runtime.exs`:

```
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},    # IP address to listen on
    local_port: 2905,             # Port to listen on
    point_code: 100               # This STP's own point code
  }
```

STP Configuration Parameters

Parameter	Type	Default	Description	Example
`enabled`	Boolean	`false`	Enable standalone STP server	`true`
`local_ip`	Tuple	`{127, 0, 0, 1}`	IP address to listen for connections	`{0, 0, 0, 0}`
`local_port`	Integer	2905	Port to listen on	2905
`point_code`	Integer	*Required*	This STP's own SS7 point code	100

When to Use Standalone STP

- **Pure Routing**: When you only need M3UA routing without MAP client functionality
- **Central STP**: To create a central signaling router for multiple network elements
- **Hub Architecture**: Connect multiple HLRs, MSCs, and SMSCs through a central STP

Note: You can enable both `map_client_m3ua` and `m3ua_stp` simultaneously if you need both outbound connections and inbound STP functionality.

Routing Table Persistence (Mnesia)

All routing tables (peers, Point Code routes, and Global Title routes) are stored in a **Mnesia database** for persistence.

How Routing Works

1. **Runtime.exs Routes**: Routes defined in `config/runtime.exs` under `m3ua_peers`, `m3ua_routes`, and `m3ua_gt_routes` are loaded at application startup
2. **Web UI Routes**: Routes added via the Web UI Routing page are stored in Mnesia
3. **Route Merge**: On restart, runtime.exs routes are merged with existing Mnesia routes (no duplicates)
4. **Persistence**: All routes configured via Web UI **survive application restarts**

Mnesia Storage Type

Control how routing tables are stored. For more details on database configuration, see Database Parameters in Configuration Reference.

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # or :ram_copies for testing
```

Storage Type	Description	Persistence	Use Case
`:disc_copies`	Disk-backed storage (default)	**Survives restarts**	Production environments
`:ram_copies`	In-memory only	Lost on restart	Testing, development

Default: `:disc_copies`

Mnesia Database Location

Mnesia stores routing tables in the application's Mnesia directory:

- **Location**: `Mnesia.{node_name}/` (e.g., `Mnesia.nonode@nohost/`)
- **Tables**: `m3ua_peer`, `m3ua_route`, `m3ua_gt_route`

Managing Routes

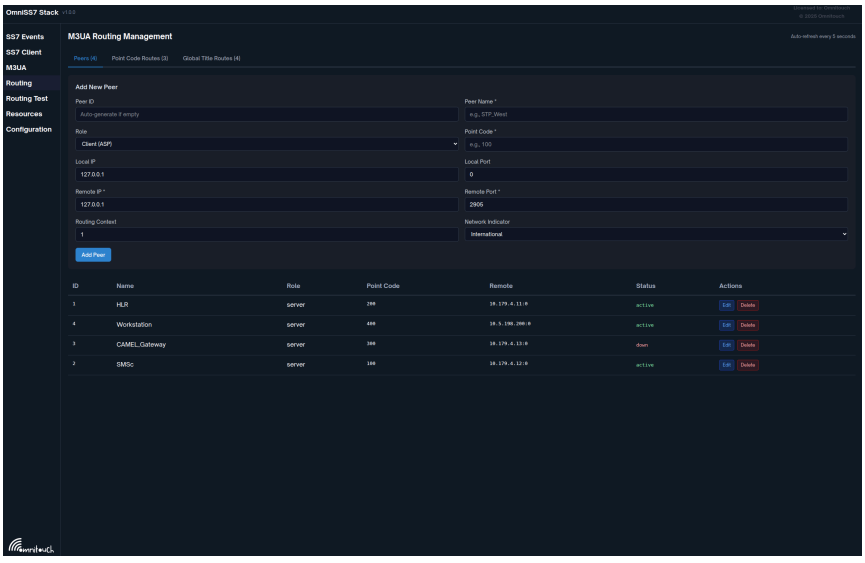
You have three options for managing routes:

1. **Runtime.exs** - Static configuration loaded at startup
2. **Web UI** - Interactive route management (see Web UI Guide)
3. **REST API** - Programmatic route management (see API Guide)

Best Practice: Use `runtime.exs` for base configuration and the Web UI for dynamic route changes during operation.

Configuring M3UA Peers

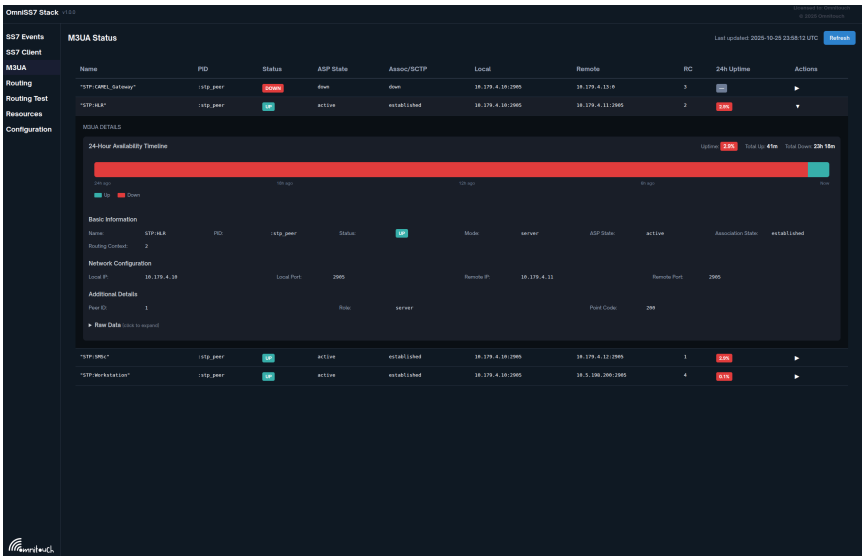
Peers represent M3UA connection endpoints (other STPs, HLRs, MSCs, SMSCs). Add peers to `config/runtime.exs`.



Peer Configuration Example

```
config :omniss7,
  m3ua_peers: [
    # Outbound connection to Partner STP (role: :client)
    %{
      peer_id: 1,                    # Unique identifier
      name: "Partner_STP_West",      # Descriptive name
      role: :client,                 # :client for outbound, :server for inbound
      local_ip: {10, 0, 0, 1},       # Local IP to bind
      local_port: 0,                 # 0 = dynamic port assignment
      remote_ip: {10, 0, 0, 10},     # Remote peer IP
      remote_port: 2905,             # Remote peer port
      routing_context: 1,            # M3UA routing context
      point_code: 100,               # Point code of this peer
      network_indicator: :international    # :international or :national
    },

    # Connection to Local HLR (role: :client)
    %{
      peer_id: 2,
      name: "Local_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 20},
      remote_port: 2905,
      routing_context: 2,
      point_code: 200,
      network_indicator: :international
    },

    # Inbound connection from Remote MSC (role: :server)
    # For :server role, STP waits for incoming connection
    %{
      peer_id: 3,
      name: "Remote_MSC",
      role: :server,                 # Accept inbound connection
      remote_ip: {10, 0, 0, 30},     # Expected source IP
      remote_port: 2905,             # Expected source port (0 = accept from any port)
      routing_context: 3,
```

```
    point_code: 300,
    network_indicator: :international
  },

  # Inbound connection with dynamic source port (no port filtering)
  %{
    peer_id: 4,
    name: "Dynamic_Client",
    role: :server,
    remote_ip: {10, 0, 0, 40},          # Expected source IP
    remote_port: 0,                      # 0 = accept connections from any source port
    routing_context: 4,
    point_code: 400,
    network_indicator: :international
  }
]
```



Peer Configuration Parameters

Parameter	Type	Required	Description
`peer_id`	Integer	Yes	Unique numeric identifier for the peer
`name`	String	Yes	Human-readable name for logs and monitoring
`role`	Atom	Yes	`:client` (outbound) or `:server` (inbound)
`local_ip`	Tuple	Yes (client)	Local IP address to bind
`local_port`	Integer	Yes (client)	Local port (0 for dynamic)
`remote_ip`	Tuple	Yes	Remote peer IP address
`remote_port`	Integer	Yes	Remote peer port (0 for inbound = accept any source port)
`routing_context`	Integer	Yes	M3UA routing context identifier
`point_code`	Integer	Yes	SS7 point code of this peer
`network_indicator`	Atom	No	`:international` or `:national`

Source Port Filtering for Inbound Connections

For **inbound connections** (role: `:server`), the `remote_port` parameter controls source port filtering:

- **Specific Port** (e.g., `remote_port: 2905`): Only accept connections from that exact source port
  - Provides additional security by validating the source port
  - Use when the remote peer uses a fixed source port
- **Any Port** (`remote_port: 0`): Accept connections from any source port
  - Useful when the remote peer uses dynamic/ephemeral source ports
  - Only validates the source IP address
  - More flexible but slightly less secure

Example:

```
# Accept only from 10.5.198.200:2905 (specific port)
%{
  peer_id: 1,
  name: "Strict_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 2905,
  # ... other config
}

# Accept from 10.5.198.200 with any source port
%{
  peer_id: 2,
  name: "Flexible_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 0,  # Accept from any source port
  # ... other config
}
```

M2PA Protocol Support

OmniSS7 supports both **M3UA** and **M2PA** protocols for SS7 signaling transport.

What is M2PA?

M2PA (MTP2 User Peer-to-Peer Adaptation Layer) is an IETF-standardized protocol (RFC 4165) for transporting SS7 MTP3 messages over IP networks using SCTP.

M3UA vs M2PA: Key Differences

Feature	M3UA	M2PA
Architecture	Client/Server (ASP/SGW)	Peer-to-Peer
Use Case	Gateway between SS7 and IP	Direct point-to-point links
Link State Management	Application-level (ASPUP/ASPAC)	MTP2-style (Alignment, Proving, Ready)
Sequence Numbers	No inherent sequencing	24-bit BSN/FSN for ordered delivery
Typical Deployment	SS7-to-IP gateway, STP	Direct signaling links between nodes
RFC	RFC 4666	RFC 4165

Protocol Selection Guidance

Recommendation: Use M3UA by default. Only use M2PA when specifically required.

When to Use M3UA (Recommended)

M3UA is the recommended protocol for most deployments:

- **STP Deployments**: Standard signaling transfer point implementations
- **Gateway Functions**: Bridging SS7 networks with IP-based signaling
- **Network Element Connections**: Connecting HLRs, MSCs, SMSCs, and other network elements to your STP
- **Signaling Gateway (SGW)**: Central gateway accepting connections from multiple Application Servers
- **Flexible Topologies**: Client/server architectures with centralized control
- **Multi-vendor Networks**: Widely supported industry standard (RFC 4666)

Use M3UA for connecting network elements (HLR, MSC, SMSC, VLR, etc.) to your STP.

When to Use M2PA (Special Cases Only)

M2PA should only be used in specific scenarios:

- **STP-to-STP Links**: Direct point-to-point connections between Signal Transfer Points in a multi-STP network
- **Legacy TDM Replacement**: Replacing traditional SS7 TDM links when the remote system specifically requires M2PA
- **MTP2 Compatibility Required**: When connecting to legacy systems that mandate MTP2-style link state management
- **Partner Requirement**: When a partner or interconnect specifically requires M2PA protocol

Important: Do not use M2PA for connecting network elements (HLR, MSC, SMSC) to your STP - use M3UA instead. M2PA is designed for STP-to-STP interconnections where both sides operate as routing nodes.

Configuring M2PA Peers

M2PA peers are configured the same way as M3UA peers, with an additional `protocol` parameter.

M2PA Peer Configuration

Add M2PA peers to your `m3ua_peers` configuration in `config/runtime.exs` (yes, they share the same configuration section despite being different protocols):

Key Parameters for M2PA:

Parameter	Value	Description
`protocol`	`:m2pa`	Specifies M2PA protocol (defaults to `:m3ua` if omitted)
`role`	`:client` or `:server`	Connection direction
`local_port`	Integer	Local SCTP port (standard M2PA port is **3565**)
`remote_port`	Integer	Remote SCTP port (standard M2PA port is **3565**)
`point_code`	Integer	Your point code
`adjacent_point_code`	Integer	Remote peer's point code (M2PA-specific)

Note: M2PA uses **port 3565** as the industry standard (different from M3UA's port 2905).

M2PA Link States

M2PA links progress through several states during initialization:

1. **Down** - No connection established
2. **Alignment** - Initial synchronization phase (~1 second)
3. **Proving** - Link quality verification (~2 seconds)
4. **Ready** - Link active and ready for traffic

The link state progression ensures reliable signaling before traffic is exchanged.

Managing M2PA Peers via Web UI

The **Routing** page in the Web UI provides full support for managing M2PA peers:

1. **Navigate** to the Routing page
2. **Select** the "Peers" tab
3. **Click** "Add New Peer"
4. **Choose** "M2PA (RFC 4165)" from the Protocol dropdown
5. **Fill in** the peer configuration:
   - Peer Name (descriptive identifier)
   - Protocol: M2PA
   - Role: client or server
   - Point Code (your PC)
   - Local/Remote IP addresses
   - Local/Remote ports (typically 3565 for M2PA)
   - Network Indicator (international or national)
6. **Click** "Save Peer"

The peers table displays the protocol type with color coding:

- **Blue** - M3UA peers
- **Green** - M2PA peers

M2PA Routing Behavior

M2PA peers integrate seamlessly with OmniSS7's routing system:

- **Point Code Routes**: Work identically for M2PA and M3UA
- **Global Title Routes**: Fully supported on M2PA links

- **Route Priority**: M2PA and M3UA peers can be mixed in the same routing tables
- **Message Relay**: Messages can arrive on M2PA and be routed to M3UA, and vice versa

M2PA Metrics

M2PA provides comprehensive Prometheus metrics for monitoring link health and traffic:

Traffic Metrics:

- `m2pa_messages_sent_total` - Total MTP3 messages sent per link
- `m2pa_messages_received_total` - Total MTP3 messages received per link
- `m2pa_bytes_sent_total` - Total bytes sent over M2PA
- `m2pa_bytes_received_total` - Total bytes received over M2PA

All traffic metrics are labeled by: `link_name`, `point_code`, `adjacent_pc`

Link State Metrics:

- `m2pa_link_state_changes_total` - Link state transitions (DOWN → ALIGNMENT → PROVING → READY)
  - Labels: `link_name`, `from_state`, `to_state`

Error Metrics:

- `m2pa_errors_total` - Total errors by type
  - `decode_error` - M2PA message decode failures
  - `encode_error` - M2PA message encode failures
  - `sctp_send_error` - SCTP transmission failures
  - Labels: `link_name`, `error_type`

Access Metrics:

- Prometheus endpoint: `http://your-server:8080/metrics`
- Metrics auto-register on application startup

M2PA Best Practices

1. **Port Selection**: Use port 3565 for M2PA (industry standard)
2. **Link Monitoring**: Monitor link state changes via metrics
3. **Firewall Rules**: Ensure SCTP (IP protocol 132) is allowed
4. **Point Codes**: Ensure adjacent point codes are correctly configured on both sides
5. **Network Indicator**: Must match between peers (international or national)
6. **Testing**: Use the Routing Test page to verify connectivity after configuration

---

Configuring Point Code Routing

Point Code routing directs messages based on the **Destination Point Code (DPC)** in the MTP3 header.

Understanding Point Codes in SS7 Protocol Stack

Point codes exist at different layers of the SS7 protocol stack. Understanding this distinction is important:

Protocol Stack Layers:

```
┌─────────────────────────────────┐
│ Application Layer (SCCP/TCAP/MAP)│
├─────────────────────────────────┤
│ MTP3 Layer                      │ ← User Data Routing
│ - Routing Label: DPC, OPC, SLS  │ ← Used for STP routing
│ - Service Information Octet (SIO)│
├─────────────────────────────────┤
│ M3UA or M2PA (Adaptation Layer) │ ← Transport Protocol
│ - Protocol Data (contains MTP3) │
│ - Network Management (DUNA/DAVA)│ ← Network Status
├─────────────────────────────────┤
│ SCTP (Transport)                │
└─────────────────────────────────┘
```

Two Types of Point Codes:

1. **MTP3 Layer Point Codes** (Used for Routing):
   - Located in the MTP3 routing label (DPC, OPC)
   - Present in M3UA Protocol Data parameter (tag 528)
   - Present in M2PA User Data messages
   - **STP uses these DPC values for routing decisions**
   - These determine where the message is ultimately delivered

2. **M3UA Layer Point Codes** (Used for Network Management):
   - Present in M3UA management messages (DUNA, DAVA, SCON, DUPU)
   - Indicate affected point codes for network status
   - Tell peers which destinations are available/unavailable
   - Not used for routing user data

How STP Routing Works:

- **For M3UA DATA messages**: STP extracts the MTP3 message from the Protocol Data parameter (tag 528), which contains the MTP3 routing label (DPC, OPC, SLS). The DPC from the MTP3 layer is used to look up routes.
- **For M2PA User Data messages**: STP extracts the MTP3 message from the M2PA user data field, then reads the DPC from the MTP3 routing label.
- **M3UA management messages**: Network management messages (DUNA, DAVA, SCON) contain affected point codes at the M3UA layer for signaling network status between peers.

Basic Point Code Routes

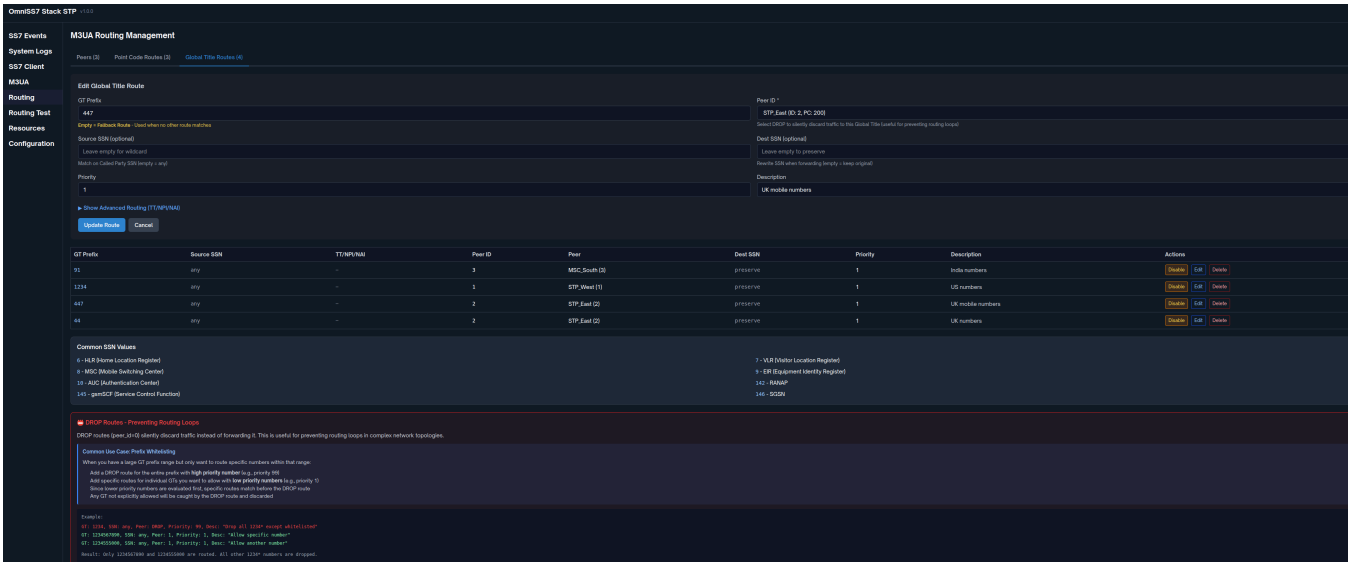
Add routes to `config/runtime.exs`:

```
config :omniss7,
  m3ua_routes: [
    # Route all traffic for PC 100 to peer 1 (Partner STP)
    %{
      dest_pc: 100,                  # Destination point code
      peer_id: 1,                    # Peer to route through
      priority: 1,                   # Priority (lower = higher priority)
      network_indicator: :international
      # mask: 14                     # Optional: defaults to 14 (exact match)
    },

    # Route all traffic for PC 200 to peer 2 (Local HLR)
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international
    },

    # Load balancing example: PC 300 with primary and backup routes
    %{
      dest_pc: 300,
      peer_id: 3,                    # Primary route
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 300,
      peer_id: 4,                    # Backup route (higher priority number)
      priority: 2,
      network_indicator: :international
    }
  ]
```

Note: The `mask` field is optional and defaults to 14 (exact point code match). Only specify `mask` when you need range-based routing (see Point Code Masks section below).

Routing Logic

1. STP receives M3UA DATA or M2PA User Data message
2. STP extracts the **MTP3 message** from the Protocol Data (M3UA) or User Data (M2PA) field
3. STP reads the **Destination Point Code (DPC)** from the MTP3 routing label
4. Looks up routing table for matching DPC (considering masks)
5. If multiple routes exist, selects the route with **most specific mask** (highest mask value), then **lowest priority number**
6. Wraps the MTP3 message in M3UA DATA or M2PA User Data for the destination peer
7. Routes the message to the corresponding peer
8. If the selected peer is down, tries the next highest priority route



Point Code Masks

Point codes are 14-bit values (range 0-16383). By default, routes match a single point code exactly (mask /14). However, you can use **point code masks** to create routes that match **ranges** of point codes.

Understanding Masks

The mask specifies how many **most significant bits** must match between the route's destination PC and the incoming message's DPC. The remaining bits can be any value, creating a range of matching point codes.

Mask Reference Table:

Mask	Point Codes Matched	Use Case
/14	1 PC (exact match)	Single destination (default)
/13	2 PCs	Small range
/12	4 PCs	Small range
/11	8 PCs	Small range
/10	16 PCs	Medium range
/9	32 PCs	Medium range
/8	64 PCs	Medium range
/7	128 PCs	Medium-large range
/6	256 PCs	Large range
/5	512 PCs	Large range
/4	1,024 PCs	Very large range
/3	2,048 PCs	Very large range
/2	4,096 PCs	Extremely large range
/1	8,192 PCs	Half of all PCs
/0	16,384 PCs	All PCs (default/fallback route)

Point Code Mask Examples

Note: The `mask` field is **optional** in all examples. If omitted, it defaults to 14 (exact match).

Example 1: Single Point Code (Default Behavior)

```
# Without mask field (recommended for single PC)
```

```
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  network_indicator: :international
}
# Mask defaults to 14 - Matches: Only PC 1000

# Explicit mask (same result)
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  mask: 14,                          # Explicit exact match
  network_indicator: :international
}
# Matches: Only PC 1000
```

Example 2: Small Range

```
%{
  dest_pc: 1000,
  peer_id: 2,
  priority: 1,
  mask: 12,                          # Matches 4 PCs
  network_indicator: :international
}
# Matches: PC 1000, 1001, 1002, 1003
```

Example 3: Medium Range

```
%{
  dest_pc: 1000,
  peer_id: 3,
  priority: 1,
  mask: 8,                           # Matches 64 PCs
  network_indicator: :international
}
# Matches: PC 1000-1063 (64 consecutive point codes)
```

Example 4: Default/Fallback Route

```
%{
  dest_pc: 0,
  peer_id: 4,
  priority: 10,                      # Low priority (high number)
  mask: 0,                           # Matches all PCs
  network_indicator: :international
}
# Matches: All point codes (0-16383)
# Use as a catch-all/default route with low priority
```

Combining Specific and Masked Routes

You can combine specific routes with masked routes for flexible routing:

```
config :omniss7,
  m3ua_routes: [
    # Specific route for PC 1000 (takes precedence)
    %{
      dest_pc: 1000,
      peer_id: 1,
      priority: 1,
      network_indicator: :international
      # mask defaults to 14 (exact match)
    },

    # Range route for PCs 1000-1063
    %{
      dest_pc: 1000,
      peer_id: 2,
      priority: 1,
      mask: 8,                       # Matches 64 PCs
      network_indicator: :international
    },

    # Default/fallback route for all other PCs
    %{
      dest_pc: 0,
      peer_id: 3,
      priority: 10,                  # Low priority
      mask: 0,                       # Matches all PCs
      network_indicator: :international
    }
  ]
```

Routing Decision for DPC 1000:

1. Matches mask `/14` route (PC 1000 exactly) - **Selected** (most specific)
2. Also matches mask `/8` route (PC 1000-1063 range) - Ignored (less specific)
3. Also matches mask `/0` route (all PCs) - Ignored (least specific)

Routing Decision for DPC 1015:

1. Does not match mask `/14` route (PC 1000 only)
2. Matches mask `/8` route (PC 1000-1063 range) - **Selected** (most specific match)
3. Also matches mask `/0` route (all PCs) - Ignored (less specific)

Routing Decision for DPC 5000:

1. Does not match mask `/14` route
2. Does not match mask `/8` route
3. Matches mask `/0` route (all PCs) - **Selected** (only match, fallback route)

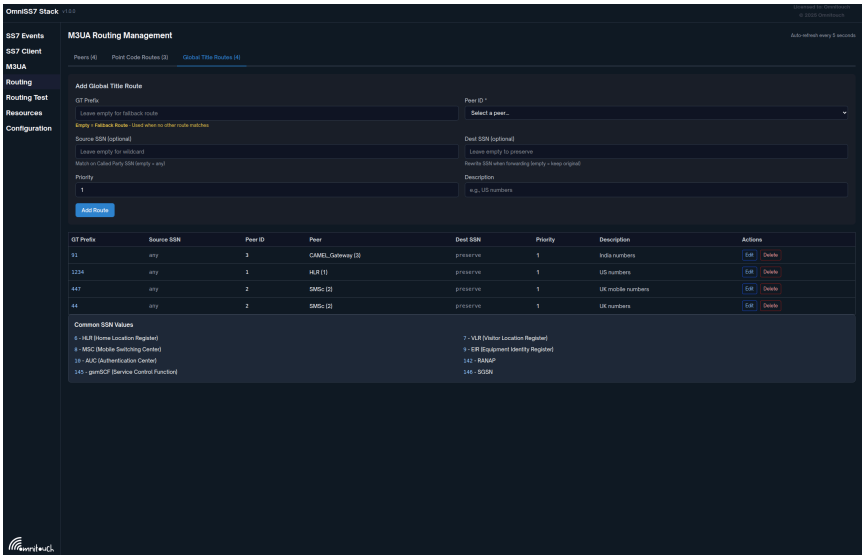
Best Practices

1. **Omit mask for Single Destinations**: For exact point code matches, omit the `mask` field entirely (defaults to `/14`)
2. **Use `/14` Explicitly Only When Needed**: Only specify `mask: 14` when you need to make it clear in documentation or when mixing with range routes
3. **Use Range Masks for Network Blocks**: Route entire network segments to specific peers with masks `/8` through `/13`
4. **Use `/0` as Fallback**: Create a default route with low priority to catch unmatched traffic
5. **Most Specific Wins**: The routing engine always selects the most specific (highest mask value) matching route first
6. **Priority as Tiebreaker**: If multiple routes have the same mask, lowest priority number wins

---

Configuring Global Title (GT) Routing

Global Title routing enables **content-based routing** using phone numbers or IMSI values instead of point codes. For advanced Global Title address translation based on calling/called party, see the Global Title NAT Guide.



Prerequisites

- Enable GT routing: `enable_gt_routing: true` in `config/runtime.exs`

GT Route Configuration

```
config :omniss7,
  # Enable GT routing
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # Route all UK numbers (prefix 44) to peer 1
    %{
      gt_prefix: "44",               # Global Title prefix to match
      peer_id: 1,                    # Destination peer
      priority: 1,                   # Priority (lower = higher)
      description: "UK numbers"      # Description for logging
    },

    # Route US numbers (prefix 1) to peer 2
    %{
      gt_prefix: "1",
      peer_id: 2,
      priority: 1,
      description: "US numbers"
    },

    # More specific route: UK mobile numbers starting with 447
    %{
      gt_prefix: "447",              # Longest prefix match wins
      peer_id: 3,
      priority: 1,
      description: "UK mobile numbers"
    },

    # SSN-specific routing (optional)
    %{
      gt_prefix: "555",
      source_ssn: 8,                 # Only match if source SSN = 8 (SMSC)
      peer_id: 4,
      dest_ssn: 6,                   # Rewrite destination SSN to 6 (HLR)
      priority: 1,
      description: "SMS traffic for 61 prefix"
    }
  ]
```

GT Routing Logic

The GT routing algorithm follows this decision process:

Routing Steps:

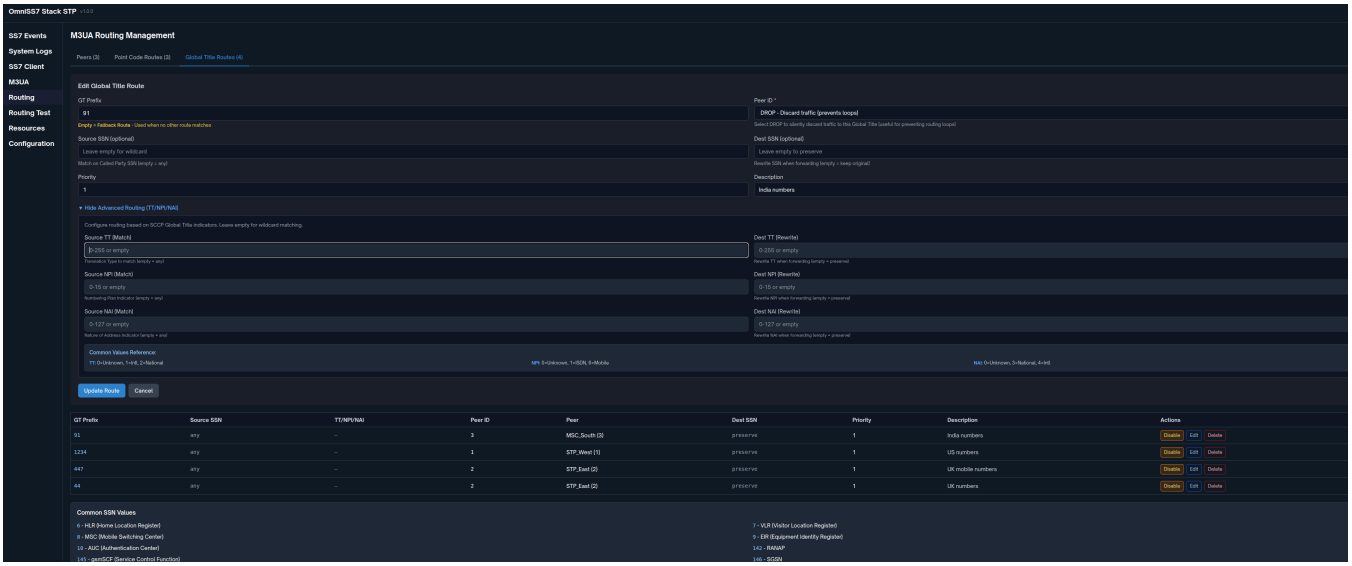
1. **Longest Prefix Match**: The STP finds all GT routes where the prefix matches the beginning of the Global Title
   - Example: GT "447712345678" matches both "44" and "447", but "447" wins (longest match)
2. **SSN Matching** (Optional):
   - If `source_ssn` is specified, the route only matches when the SCCP Called Party SSN equals that value
   - If `source_ssn` is `nil`, the route matches any SSN (wildcard)

3. **TT/NPI/NAI Matching** (Optional):
   - If `source_tt`, `source_npi`, or `source_nai` are specified, routes must match those indicators
   - `nil` values act as wildcards (match any value)
4. **Specificity-Based Selection**:
   - Routes with more specific matching criteria win over wildcards
   - Priority order: GT Prefix Length → SSN → TT → NPI → NAI → Priority Number
5. **Indicator Rewriting** (Optional):
   - If `dest_ssn`, `dest_tt`, `dest_npi`, or `dest_nai` are specified, the STP rewrites those indicators
   - Useful for protocol normalization and network interconnection
6. **Fallback to Point Code**:
   - If no GT route matches, the STP falls back to Point Code routing using the DPC

Advanced GT Routing: Translation Type, NPI, and NAI

In addition to GT prefix and SSN matching, the STP supports routing and transformation based on SCCP Global Title indicators:

- **Translation Type (TT)**: Identifies the numbering plan and address type
- **Numbering Plan Indicator (NPI)**: Defines the numbering plan (e.g., ISDN, Data, Telex)
- **Nature of Address Indicator (NAI)**: Specifies the address format (e.g., International, National, Subscriber)



Matching (Source Indicators)

Routes can match on incoming message indicators:

- `source_tt`: Match messages with specific Translation Type
- `source_npi`: Match messages with specific Numbering Plan Indicator
- `source_nai`: Match messages with specific Nature of Address Indicator
- `nil` value = wildcard (matches any value)

Transformation (Destination Indicators)

Routes can rewrite indicators when forwarding:

- `dest_tt`: Transform Translation Type to new value
- `dest_npi`: Transform Numbering Plan Indicator to new value
- `dest_nai`: Transform Nature of Address Indicator to new value
- `nil` value = preserve original value (no transformation)

Specificity-Based Selection

When multiple routes match, the most specific route is selected using this priority order:

1. Longest GT prefix match
2. Specific source SSN over wildcard SSN
3. Specific source TT over wildcard TT
4. Specific source NPI over wildcard NPI
5. Specific source NAI over wildcard NAI
6. Lowest priority number

Configuration Examples

```
config :omniss7,
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # Example 1: Match and transform Translation Type
    %{
      gt_prefix: "44",
      peer_id: 1,
      source_tt: 0,      # Match TT=0 (Unknown)
      dest_tt: 3,        # Transform to TT=3 (National)
      priority: 1,
      description: "UK numbers: TT 0→3 transformation"
    },

    # Example 2: Match specific NPI and transform NAI
    %{
      gt_prefix: "1",
      peer_id: 2,
      source_npi: 1,     # Match NPI=1 (ISDN/Telephony)
      source_nai: 4,     # Match NAI=4 (International)
      dest_nai: 3,       # Transform to NAI=3 (National)
      priority: 1,
      description: "US numbers: International→National NAI"
    },

    # Example 3: Combined SSN and indicator routing
    %{
      gt_prefix: "33",
      source_ssn: 8,     # Match SMSC traffic
      source_tt: 0,      # Match TT=0
      dest_ssn: 6,       # Rewrite SSN to HLR
      dest_tt: 2,        # Transform to TT=2
      dest_npi: 1,       # Set NPI=1 (ISDN)
      dest_nai: 4,       # Set NAI=4 (International)
      peer_id: 3,
      priority: 1,
      description: "French SMS: Full normalization"
    },

    # Example 4: Wildcard TT, specific NPI
    %{
      gt_prefix: "49",
      source_tt: nil,    # Match any TT (wildcard)
      source_npi: 6,     # Match NPI=6 (Data)
      dest_npi: 1,       # Transform to NPI=1 (ISDN)
      peer_id: 4,
      priority: 1,
      description: "German data network normalization"
    }
  ]
```

Common TT/NPI/NAI Values

Translation Type (TT):

- `0` = Unknown
- `1` = International
- `2` = National
- `3` = Network Specific

Numbering Plan Indicator (NPI):

- `0` = Unknown
- `1` = ISDN/Telephony (E.164)
- `3` = Data (X.121)
- `4` = Telex (F.69)
- `6` = Land Mobile (E.212)

Nature of Address Indicator (NAI):

- `0` = Unknown
- `1` = Subscriber Number
- `2` = Reserved for National Use
- `3` = National Significant Number
- `4` = International Number

Routing Decision Example

For an incoming message with:

- GT: "447712345678"
- SSN: 8
- TT: 0
- NPI: 1
- NAI: 4

With these configured routes:

```
# Route A: Wildcard TT
%{gt_prefix: "447", peer_id: 1, priority: 1}

# Route B: Specific TT
%{gt_prefix: "447", source_tt: 0, peer_id: 2, priority: 1}

# Route C: Specific TT + NPI
%{gt_prefix: "447", source_tt: 0, source_npi: 1, peer_id: 3, priority: 1}
```

Result: Route C is selected (most specific: matches GT + TT + NPI)

The message is forwarded with indicators transformed per Route C's `dest_tt`, `dest_npi`, `dest_nai` values.

GT Routing Examples

Called GT	Source SSN	TT	NPI	NAI	Matched Route	Reason
447712345678	-	-	-	-	"447" → peer 3	Longest prefix match
4412345678906	-	-	-	-	"44" → peer 1	Prefix match, no more specific route
12125551234	6	-	-	-	"1" → peer 2	Prefix match for US numbers
5558812345678	-	-	-	-	"555" (SSN 8) → peer 4	GT + SSN match, rewrites SSN to 6
5558812345676	-	-	-	-	"555" (SSN wildcard) → peer X	GT match, no SSN rewrite
4412345678906	0	1	4	-	"44" (TT=0) → peer 1	GT + TT match, transforms TT to 3
12125551234	8	0	1	4	"1" (TT=0, NPI=1, NAI=4)	Most specific: GT+TT+NPI+NAI match

Practical Use Cases for TT/NPI/NAI Routing

1. **Network Interconnection Normalization**
   - Different networks may use different indicator conventions
   - Transform indicators at the interconnection point to ensure compatibility
   - Example: Partner network uses TT=0 for international, your network uses TT=1

2. **Protocol Conversion**
   - Convert between numbering plans when routing between different network types
   - Example: Route from mobile network (NPI=6) to PSTN (NPI=1)
3. **Address Format Standardization**
   - Normalize all incoming traffic to use consistent NAI values
   - Example: Convert all international format (NAI=4) to national format (NAI=3) for domestic routing
4. **Carrier-Specific Routing**
   - Route based on translation type to different service providers
   - Example: TT=0 routes to Carrier A, TT=2 routes to Carrier B
5. **Legacy System Integration**
   - Modern systems might use different indicator values than legacy systems
   - Transform at the STP to maintain backward compatibility

Route Management Features

Disabling Routes

Routes can be temporarily disabled without deleting them. This is useful for testing, maintenance, or traffic management.

Enabled Flag

Both Point Code and Global Title routes support an optional `enabled` flag:

```
config :omniss7,
  m3ua_routes: [
    # Active route
    %{
      dest_pc: 100,
      peer_id: 1,
      priority: 1,
      network_indicator: :international,
      enabled: true  # Route is active (default if omitted)
    },

    # Disabled route (not evaluated during routing)
    %{
      dest_pc: 200,
      peer_id: 1,
      priority: 1,
      network_indicator: :international,
      enabled: false  # Route is disabled
    }
  ],

  m3ua_gt_routes: [
    # Disabled GT route
    %{
      gt_prefix: "44",
      peer_id: 1,
      priority: 1,
      description: "UK numbers - temporarily disabled",
      enabled: false
    }
  ]
```

Default Behavior:

- If `enabled` is not specified, routes default to `enabled: true`
- Disabled routes are completely skipped during route lookup
- Use the Web UI to toggle routes on/off without editing config

Use Cases:

- Testing traffic flow changes
- Temporary maintenance windows
- A/B testing different routing paths
- Gradual rollout of new routes

DROP Routes - Preventing Routing Loops

DROP routes (with `peer_id: 0`) silently discard traffic instead of forwarding it. This prevents routing loops and enables advanced traffic filtering.

Configuring DROP Routes

```
config :omniss7,
  m3ua_routes: [
    # DROP route for specific point code
    %{
      dest_pc: 999,
      peer_id: 0,        # peer_id=0 means DROP
      priority: 1,
      network_indicator: :international
    }
  ],

  m3ua_gt_routes: [
    # DROP route for GT prefix
    %{
      gt_prefix: "999",
      peer_id: 0,        # peer_id=0 means DROP
      priority: 99,
      description: "Block test range"
    }
  ]
```

How DROP Routes Work

When a message matches a DROP route:

1. The routing engine identifies `peer_id: 0`
2. The message is **silently discarded** (not forwarded)
3. An **INFO log** is generated: `"DROP route matched for DPC 999"` or `"DROP route matched for GT 999"`
4. The routing lookup returns `{:error, :dropped}`

Important: Dropped traffic is logged at INFO level for monitoring and troubleshooting.

Common Use Case: Prefix Whitelisting

One of the most powerful uses of DROP routes is **prefix whitelisting** - allowing only specific numbers within a large range while blocking all others.

The Pattern:

1. Create a DROP route for the entire prefix with **high priority number** (e.g., 99)
2. Create specific allow routes for individual numbers with **low priority numbers** (e.g., 1)
3. Since lower priority numbers are evaluated first, allowed routes match before the DROP route
4. Any number not explicitly allowed gets caught by the DROP route

Example Scenario:

You have a GT prefix `1234` that represents a range of 10,000 numbers (1234000000 - 1234999999), but you only want to route 3 specific numbers: `1234567890`, `1234555000`, and `1234111222`.

```
config :omniss7,
  m3ua_gt_routes: [
    # DROP route with HIGH priority number (evaluated last)
    %{
      gt_prefix: "1234",
      peer_id: 0,                # DROP
      priority: 99,              # High number = low priority = evaluated last
      description: "Block all 1234* except whitelisted numbers"
    },

    # Specific allow routes with LOW priority numbers (evaluated first)
    %{
      gt_prefix: "1234567890",
      peer_id: 1,                # Route to peer 1
      priority: 1,               # Low number = high priority = evaluated first
      description: "Allowed number 1"
    },
    %{
      gt_prefix: "1234555000",
      peer_id: 1,
      priority: 1,
      description: "Allowed number 2"
    },
    %{
      gt_prefix: "1234111222",
      peer_id: 1,
      priority: 1,
      description: "Allowed number 3"
    }
  ]
```

Routing Behavior:

Incoming GT	Matching Routes	Selected Route	Action
1234567890	✧ "1234567890" (priority 1) , "1234567890" (most specific, highest priority) ✧ "1234" DROP (priority 99)		**Routed to peer 1**
1234555000	✧ "1234555000" (priority 1) , "1234555000" (most specific, highest priority) ✧ "1234" DROP (priority 99)		**Routed to peer 1**
1234111222	✧ "1234111222" (priority 1) , "1234111222" (most specific, highest priority) ✧ "1234" DROP (priority 99)		**Routed to peer 1**
1234999999	✧ "1234" DROP (priority 99)	"1234" DROP (only match)	**Dropped + logged**
1234000000	✧ "1234" DROP (priority 99)	"1234" DROP (only match)	**Dropped + logged**

Result:

- ✧ Only 3 specific numbers are routed to peer 1
- ✧ All other `1234*` numbers are silently dropped
- ✧ All dropped traffic is logged for monitoring

Logs Generated:

```
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for GT 1234000000
```

DROP Routes for Point Codes

The same whitelist pattern works for Point Code routing:

```
config :omniss7,
  m3ua_routes: [
    # DROP entire range /8 (64 point codes: 1000-1063)
    %{
      dest_pc: 1000,
      peer_id: 0,
      priority: 99,
      mask: 8,
      network_indicator: :international
    },

    # Allow specific PCs
    %{dest_pc: 1010, peer_id: 1, priority: 1, network_indicator: :international},
    %{dest_pc: 1020, peer_id: 1, priority: 1, network_indicator: :international},
    %{dest_pc: 1030, peer_id: 1, priority: 1, network_indicator: :international}
  ]
```

Result: Only PCs 1010, 1020, and 1030 are routed. All other PCs in the 1000-1063 range are dropped.

Monitoring DROP Routes

Check Logs:

```
# Monitor for dropped traffic
tail -f logs/app.log | grep "DROP route matched"

# Expected output:
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for DPC 1050
```

Via Web UI:

- Navigate to **System Logs** tab
- Filter by `INFO` level
- Search for `"DROP route matched"`

Best Practices:

1. ⚠ Monitor logs regularly to ensure DROP routes aren't blocking legitimate traffic
2. ✧ Use descriptive `description` fields to document why routes are dropped
3. ✧ Use high priority numbers (90-99) for DROP routes to ensure they're catch-all routes
4. ✧ Test DROP route behavior before deploying to production
5. ✧ Set up alerts for unexpected increases in dropped traffic

---

Advanced Routing: SSN-Based Routing and Rewriting

Subsystem Numbers (SSN)

Subsystem Numbers identify the application layer:

- **SSN 6**: HLR (Home Location Register)
- **SSN 7**: VLR (Visitor Location Register)
- **SSN 8**: MSC (Mobile Switching Center) / SMSC (SMS Center)
- **SSN 9**: GMLC (Gateway Mobile Location Center)

SSN-Based Routing Example

Route SMS traffic to different HLR based on number prefix:

```
m3ua_gt_routes: [
  # Route SMS for UK numbers to UK HLR, rewrite SSN from 8 (SMSC) to 6 (HLR)
  %{
    gt_prefix: "44",
    source_ssn: 8,                    # Match incoming SSN 8 (SMSC)
    peer_id: 1,
    dest_ssn: 6,                      # Rewrite to SSN 6 (HLR)
    priority: 1,
    description: "UK SMS to HLR"
```

```
  },

  # Route voice traffic for UK numbers (SSN 6) without rewriting
  %{
    gt_prefix: "44",
    source_ssn: 6,                  # Match incoming SSN 6 (HLR)
    peer_id: 1,
    dest_ssn: nil,                  # No SSN rewrite
    priority: 1,
    description: "UK voice traffic"
  }
]
```

Testing STP Routing Configuration

After configuring peers and routes, verify your configuration:

1. Check Peer Status

Via Web UI:

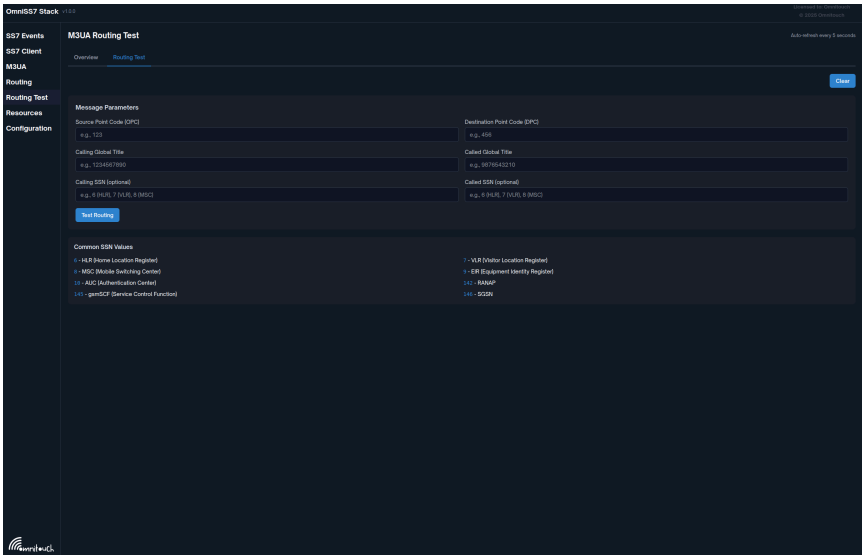
- Navigate to http://localhost
- Check M3UA Status page
- Verify peers show **Status: ACTIVE**

Via IEx Console:

```
# Get all peer statuses
M3UA.STP.get_peers_status()

# Expected output:
# [
#   %{peer_id: 1, name: "Partner_STP_West", status: :active, point_code: 100, ...},
#   %{peer_id: 2, name: "Local_HLR", status: :active, point_code: 200, ...}
# ]
```

2. Test Point Code Routing

```
# Send test M3UA message to DPC 100
test_payload = <<1, 2, 3, 4>>  # Dummy payload
M3UA.STP.route_by_pc(100, test_payload, 0)

# Check logs for routing decision
# Expected log: "Routing message: OPC=... -> DPC=100 via peer 1"
```



3. Test Global Title Routing

```
# Look up GT route manually
M3UARouting.lookup_peer_by_gt("447712345678")

# Expected output:
# {:ok, {:m3ua_peer, 3, "UK_Mobile_Peer", ...}, nil}

# Look up GT route with SSN
M3UARouting.lookup_peer_by_gt("555881234567", 8)

# Expected output with SSN rewrite:
# {:ok, {:m3ua_peer, 4, "SMS_HLR_Peer", ...}, 6}
```

4. Monitor Routing Metrics

Access Prometheus metrics at `/metrics`

Key metrics:

```
# Messages received per peer
m3ua_stp_messages_received_total{peer_name="Partner_STP_West",point_code="100"} 1523

# Messages sent per peer
m3ua_stp_messages_sent_total{peer_name="Local_HLR",point_code="200"} 1498

# Routing failures
m3ua_stp_routing_failures_total{reason="no_route"} 5
m3ua_stp_routing_failures_total{reason="no_gt_route"} 2
```

STP Metrics and Monitoring

Available Metrics

Per-Peer Traffic Metrics:

- `m3ua_stp_messages_received_total` - Total messages received from each peer
  - Labels: `peer_name`, `point_code`
- `m3ua_stp_messages_sent_total` - Total messages forwarded to each peer
  - Labels: `peer_name`, `point_code`

Routing Failure Metrics:

- `m3ua_stp_routing_failures_total` - Count of routing failures by reason
  - Labels: `reason` (values: `no_route`, `no_gt_route`)

Metric Interpretation

- **High message counts**: Indicates active traffic flow
- **Routing failures**: Indicates missing routes or misconfiguration
  - `no_route`: No Point Code route found for destination
  - `no_gt_route`: No Global Title route found, and PC routing also failed

Troubleshooting with Metrics

Scenario: No traffic reaching destination

1. Check if messages are being received:
   ```
   m3ua_stp_messages_received_total{peer_name="Source_Peer"} > 0
   ```
2. Check if messages are being sent:
   ```
   m3ua_stp_messages_sent_total{peer_name="Dest_Peer"} > 0
   ```
3. Check for routing failures:
   ```
   m3ua_stp_routing_failures_total{reason="no_route"} > 0
   ```

Solution: If routing failures are high, add missing routes in configuration.

M3UA Peer Status Monitoring

Understanding M3UA

M3UA (MTP3 User Adaptation Layer) is a protocol that allows SS7 signaling to be transported over IP networks using SCTP.

M3UA Connection States

M3UA connections progress through several states:

State Descriptions:

- **DOWN** - No SCTP connection
- **CONNECTING** - SCTP connection in progress
- **ASPUP_SENT** - Waiting for ASPUP acknowledgment
- **INACTIVE** - ASP is up but not active
- **ASPAC_SENT** - Waiting for ASPAC acknowledgment
- **ACTIVE** - Ready for traffic, fully operational
- **ASPDOWN_SENT** - Graceful shutdown in progress

Monitoring M3UA Peers via Web UI

The Web UI provides real-time monitoring of M3UA peer connections.

Accessing M3UA Status Page:

1. Navigate to the Web UI home page
2. Click on "M3UA Status" in the navigation menu
3. The page auto-refreshes every second

M3UA Status Table:

Column	Description
Name	Connection name (e.g., `testASP`)
PID	Process identifier
Status	UP (green) or DOWN (red)
ASP State	Current M3UA state (e.g., ACTIVE, INACTIVE)
Assoc/SCTP	SCTP association state
Local	Local IP:Port
Remote	Remote IP:Port
RC	Routing Context ID

Status Indicators:

- **Green (UP)** - Connection is active and healthy
- **Red (DOWN)** - Connection is down or unavailable
- **ASP State** - Shows current M3UA connection state
- **Assoc/SCTP** - Shows SCTP association status

M3UA Message Flow

Troubleshooting M3UA Connections

Issue: Connection Won't Establish

Symptoms:

- Status shows DOWN
- No SCTP association

Checks:

1. Verify network connectivity: `ping remote_ip`
2. Check firewall allows SCTP (protocol 132)
3. Verify remote STP/SGP is listening on correct port
4. Check `remote_ip` and `remote_port` in config
5. Review application logs for SCTP errors

Issue: Connection Established but ASP Not Active

Symptoms:

- SCTP association exists
- ASP state stuck in INACTIVE or ASPUP_SENT

Checks:

1. Verify routing context matches remote configuration
2. Check remote STP accepts your point code
3. Review logs for ASPUP/ASPAC rejections
4. Verify no authentication/security requirements

Issue: Data Not Flowing

Symptoms:

- ASP state shows ACTIVE
- No messages being routed

Checks:

1. Verify routing context in messages
2. Check SCCP addressing (GT format, SSN values)
3. Verify routing tables configured correctly
4. Review `/events` page for SCCP errors
5. Check point code routing at STP level

M2PA Peer Status Monitoring

Understanding M2PA

M2PA (MTP2 User Peer-to-Peer Adaptation Layer) is a protocol defined in RFC 4165 that provides point-to-point MTP3 message transport over SCTP. Unlike M3UA which uses an ASP/SGP architecture, M2PA provides peer-to-peer links similar to traditional TDM SS7 links.

M2PA Link States

M2PA links progress through several states during establishment:

State Descriptions:

- **DOWN** - No SCTP connection, link inactive
- **CONNECTING** - SCTP association in progress
- **ALIGNMENT** - Link Status messages exchanged (~1 second)
- **PROVING** - Link proving period, testing link integrity (~2 seconds)
- **READY** - Link operational, ready for MTP3 user data transfer
- **ALIGNMENT (re-entry)** - Link status change requires re-alignment

Link State Progression:

1. **SCTP Connection**: Establishes SCTP association (DOWN → CONNECTING)
2. **Alignment**: Exchanges Link Status messages to synchronize (CONNECTING → ALIGNMENT)
3. **Proving**: Tests link reliability and sequence number synchronization (ALIGNMENT → PROVING)
4. **Ready**: Link becomes operational for data transfer (PROVING → READY)

M2PA Message Flow

Monitoring M2PA Peers via Web UI

The Web UI provides real-time monitoring of M2PA peer connections.

Accessing Routing Management Page:

1. Navigate to the Web UI home page
2. Click on "Routing Management" in the navigation menu
3. View the "M3UA/M2PA Peers" table

M2PA Peer Table:

Column	Description
Peer ID	Unique peer identifier
Name	Peer name (e.g., M2PA_Link_STP_A)
Protocol	Shows "M2PA" in green
Point Code	Local point code
Adj. PC	Adjacent peer point code
Local	Local IP:Port (typically port 3565)
Remote	Remote IP:Port
Status	Link state (e.g., READY, ALIGNMENT, DOWN)

Status Indicators:

- **READY (Green)** - Link is operational and passing traffic
- **ALIGNMENT (Yellow)** - Link is aligning, not yet ready
- **PROVING (Yellow)** - Link is in proving state
- **DOWN (Red)** - Link is down or unavailable

Troubleshooting M2PA Connections

Issue: Link Stuck in ALIGNMENT

Symptoms:

- Link state shows ALIGNMENT for extended period
- No progression to PROVING or READY

Checks:

1. Verify both sides are configured with correct point codes
2. Check SCTP firewall allows protocol 132
3. Verify `point_code` and `adjacent_point_code` are correctly set
4. Review application logs for Link Status message errors
5. Ensure remote peer is also in ALIGNMENT state

Issue: Link Stuck in PROVING

Symptoms:

- Link reaches PROVING but doesn't transition to READY
- Proving period exceeds 2-3 seconds

Checks:

1. Verify network stability (no packet loss)
2. Check for SCTP association errors
3. Review logs for sequence number mismatches
4. Ensure remote peer is also in PROVING state
5. Verify SCTP multihoming isn't causing routing issues

Issue: Link Flapping (DOWN ↔ READY)

Symptoms:

- Link repeatedly cycles between READY and DOWN
- Frequent re-alignments

Checks:

1. Check network connectivity stability
2. Verify SCTP heartbeat settings
3. Review firewall session timeout settings
4. Check for MTU/fragmentation issues
5. Verify no duplicate IP addresses

Issue: Data Not Flowing

Symptoms:

- Link state shows READY
- No MTP3 messages being transferred

Checks:

1. Verify routing tables include routes to this peer
2. Check MTP3 point code routing is configured
3. Review DPC values in messages match expected routes
4. Check `/events` page for routing errors
5. Verify sequence numbers (BSN/FSN) are incrementing

Related Documentation

- ← Back to Main Documentation
- Common Features Guide - Web UI, API, Monitoring
- MAP Client Guide - Sending MAP requests
- SMS Center Guide - SMS delivery
- Technical Reference - Protocol specifications

Web UI Guide

← Back to Main Documentation

This guide provides comprehensive documentation for using the OmniSS7 **Web UI** (Phoenix LiveView interface).

Table of Contents

Overview

The OmniSS7 Web UI is a **Phoenix LiveView** application that provides real-time monitoring and management capabilities. The available pages depend on which operational mode is active (STP, HLR, or SMSc).

Web UI Architecture

Server Configuration

- **Protocol**: HTTPS
- **Port**: 443 (configured in `config/runtime.exs`)
- **Default IP**: 0.0.0.0 (listens on all interfaces)
- **Certificates**: Located in `priv/cert/`

Access URL: `https://[server-ip]:443`

Accessing the Web UI

Prerequisites

1. **SSL Certificates**: Ensure valid SSL certificates are present in `priv/cert/`:
    - `omnitouch.crt` - Certificate file

    - `omnitouch.pem` - Private key file

2. **Application Running**: Start the application with `iex -S mix`
3. **Firewall**: Ensure port 443 is open for HTTPS traffic

Available Pages by Mode

Page	STP Mode	HLR Mode	SMSc Mode	Description
SS7 Events	✓	✓	✓	Event logging and SCCP message capture
SS7 Client	✓	✓	✓	Manual MAP operation testing
M3UA	✓	✓	✓	M3UA connection status
Routing	✓	✓	✓	M3UA routing table management
Routing Test	✓	✓	✓	Route testing and validation
HLR Links	✓	✓	✓	HLR API status and subscriber management
Active Subscribers	✓	✓	✓	Real-time subscriber location tracking (HLR)
SMSc Links	✓	✓	✓	SMSc API status and queue management
SMSc Subscribers	✓	✓	✓	Real-time subscriber tracking (SMSc)
Application	✓	✓	✓	System resources and monitoring
Configuration	✓	✓	✓	Configuration viewer

Routing Management

Page: `/routing` **Modes**: STP, SMSc **Auto-Refresh**: Every 5 seconds

The Routing Management page provides a tabbed interface for managing M3UA routing tables.

Page Layout

Peers Tab

Manage M3UA peer connections (other STPs, HLRs, MSCs, SMSCs).

Peer Table Columns

Column	Description	Example
ID	Unique peer identifier	1
Name	Human-readable peer name	`"STP_West"`

Column	Description	Example
Role	Connection role	`client`, `server`, `stp`
Point Code	Peer's SS7 point code	`100`
Remote	Remote IP:Port	`10.0.0.10:2905`
Status	Connection status	`active`, `aspup`, `down`
Actions	Edit/Delete buttons	-

Adding a Peer

1. **Click** the Peers tab
2. **Fill in** the form fields:
   - **Peer ID**: Auto-generated if left empty
   - **Peer Name**: Descriptive name (required)
   - **Role**: Select `client`, `server`, or `stp`
   - **Point Code**: SS7 point code (required)
   - **Local IP**: Your system's IP address
   - **Local Port**: 0 for dynamic port assignment
   - **Remote IP**: Peer's IP address
   - **Remote Port**: Peer's port (typically 2905)
   - **Routing Context**: M3UA routing context ID
   - **Network Indicator**: `international` or `national`
3. **Click** "Add Peer"

Persistence: Peer is immediately saved to Mnesia and survives restart.

Editing a Peer

1. **Click** the "Edit" button on the peer row
2. **Modify** the form fields as needed
3. **Click** "Update Peer"

Note: If you change the Peer ID, the old peer is deleted and a new one is created.

Deleting a Peer

1. **Click** the "Delete" button on the peer row
2. **Confirm** the deletion (all routes using this peer will also be removed)

Peer Status Indicators

Status	Color	Description
`active`	🟢 Green	Peer is connected and routing messages
`aspup`	🟡 Yellow	ASP is up but not yet active
`down`	🔴 Red	Peer is disconnected

Point Code Routes Tab

Configure routing rules based on destination Point Codes.

Route Table Columns

Column	Description	Example
Destination PC	Target point code (zone.area.id format)	`1.2.3` (100)
Mask	Subnet mask for PC matching	`/14` (exact), `/8` (range)
Peer ID	Target peer for this route	`1`
Peer Name	Name of target peer	`"STP_West"`
Priority	Route priority (1 = highest)	`1`
Network	Network indicator	`international`
Actions	Edit/Delete buttons	-

Adding a Point Code Route

1. **Click** the "Point Code Routes" tab
2. **Fill in** the form fields:
    - **Destination Point Code**: Enter as `zone.area.id` (e.g., `1.2.3`) or integer (0-16383)
    - **Mask**: Select mask `/14` for exact match, lower values for ranges
    - **Peer ID**: Select target peer from dropdown
    - **Priority**: Enter priority (1 = highest, default)
    - **Network Indicator**: Select `international` or `national`
3. **Click** "Add Route"

Point Code Format: You can enter point codes in two formats:

- **3-8-3 Format**: `zone.area.id` (e.g., `1.2.3`)
- **Integer Format**: `0-16383` (e.g., `1100`)

The system automatically converts between formats.

Understanding Masks

Point codes are 14-bit values (0-16383). The mask specifies how many most significant bits must match:

Mask	PCs Matched	Use Case
`/14`	1 (exact match)	Route to specific destination
`/13`	2 PCs	Small range
`/8`	64 PCs	Medium range
`/0`	All 16,384 PCs	**Default/fallback route**

Examples:

- `PC 1000 /14` → Matches only PC 1000
- `PC 1000 /8` → Matches PC 1000-1063 (64 consecutive PCs)
- `PC 0 /0` → Matches all point codes (default route)

Point Code Mask Reference Card

The web page includes an interactive reference showing all mask values and their ranges.

Global Title Routes Tab

Configure routing rules based on SCCP Global Title addresses.

Requirement: Global Title routing must be enabled in configuration:

```
config :omniss7,
  enable_gt_routing: true
```

Route Table Columns

Column	Description	Example
GT Prefix	Called party GT prefix (empty = fallback)	`"1234"`, `""`
Source SSN	Match on called party SSN (optional)	`6` (HLR), `any`
Peer ID	Target peer	`1`
Peer	Peer name	`"HLR_West (1)"`
Dest SSN	Rewrite SSN when forwarding (optional)	`6`, `preserve`
Priority	Route priority	`1`
Description	Route description	`"US numbers"`
Actions	Edit/Delete buttons	-

Adding a Global Title Route

1. **Click** the "Global Title Routes" tab
2. **Fill in** the form fields:
   - **GT Prefix**: Leave empty for fallback route, or enter digits (e.g., "1234")
   - **Source SSN**: Optional - filter by called party SSN
   - **Peer ID**: Select target peer
   - **Dest SSN**: Optional - rewrite SSN when forwarding
   - **Priority**: Route priority (1 = highest)
   - **Description**: Human-readable description
3. **Click** "Add Route"

Fallback Routes: If GT Prefix is empty, the route acts as a catch-all for GTs that don't match any other route.

Common SSN Values

The page includes a reference card with common SSN values:

SSN	Network Element
6	HLR (Home Location Register)
7	VLR (Visitor Location Register)
8	MSC (Mobile Switching Center)
9	EIR (Equipment Identity Register)
10	AUC (Authentication Center)
142	RANAP
145	gsmSCF (Service Control Function)
146	SGSN

SSN Rewriting

- **Source SSN**: Match on the Called Party SSN in incoming messages
- **Dest SSN**: If set, rewrites the Called Party SSN when forwarding
  - Empty = preserve original SSN
  - Value = replace with this SSN

Use Case: Route messages with SSN=6 (HLR) to a peer, and rewrite to SSN=7 (VLR) on the outgoing side.

Routing Table Persistence

All routes are stored in Mnesia and survive application restarts.

How Routes Persist

1. **Web UI Changes**: All add/edit/delete operations are immediately saved to Mnesia
2. **Application Restart**: Routes are loaded from Mnesia on startup
3. **Runtime.exs Merge**: Static routes from `config/runtime.exs` are merged with Mnesia routes (no duplicates)

Route Priority

When multiple routes match a destination:

1. **More Specific First**: Higher mask values (more specific) take precedence
2. **Priority Field**: Lower priority numbers route first (1 = highest priority)
3. **Peer Status**: Only routes to `active` peers are used

Active Subscribers

Page: `/subscribers` **Mode**: HLR only **Auto-Refresh**: Every 2 seconds

Displays real-time tracking of subscribers who have sent UpdateLocation requests.

Page Features

Subscriber Table Columns

Column	Description	Example
IMSI	Subscriber IMSI	`"50557123456789"`
VLR Number	Current VLR GT address	`"555123155"`
MSC Number	Current MSC GT address	`"555123155"`
Updated At	Last UpdateLocation timestamp	`"2025-10-25 14:23:45 UTC"`
Duration	Time since registration	`"2h 15m 34s"`

Statistics Summary

When subscribers are present, a summary card displays:

- **Total Active**: Total number of registered subscribers
- **Unique VLRs**: Number of distinct VLR addresses
- **Unique MSCs**: Number of distinct MSC addresses

Clearing Subscribers

Clear All Button: Removes all active subscribers from the tracker.

Confirmation: Requires confirmation before clearing (cannot be undone).

Use Case: Clear stale subscriber records after network maintenance or testing.

Auto-Refresh

The page automatically refreshes every **2 seconds** to show real-time subscriber updates.

SMSc Subscribers

Page: `/smsc_subscribers` **Mode**: SMSc only **Auto-Refresh**: Every 2 seconds

Displays real-time tracking of subscribers based on alertServiceCenter messages received from HLRs, message delivery status, and failure tracking.

Page Features

Subscriber Table Columns

Column	Description	Example
MSISDN	Subscriber's phone number	`"15551234567"`
IMSI	Subscriber IMSI	`"001010123456789"`
HLR GT	HLR GT that sent alertServiceCenter	`"15551111111"`
Msgs Sent	Count of MT-FSM messages sent	`5`
Msgs Rcvd	Count of MO-FSM messages received	`2`
Status	`Active` or `Failed` (color-coded)	`● Active`
Last Updated	Last update timestamp	`"2025-10-30 14:23:45 UTC"`
Duration	Time since last update	`"15m 34s"`

Status Indicators

- ● **Active** (Green): Subscriber is reachable, last alertServiceCenter received successfully
- ○ **Failed** (Red): Last delivery attempt failed (SRI-for-SM or absent subscriber error)

Statistics Summary

When subscribers are present, a summary card displays:

- **Total Tracked**: Total number of tracked subscribers
- **Active**: Number of subscribers with active status
- **Failed**: Number of subscribers with failed status
- **Unique HLRs**: Number of distinct HLRs sending alerts

Managing Subscribers

Remove Button: Removes individual subscriber from tracking.

Clear All Button: Removes all tracked subscribers.

Confirmation: Clear All requires confirmation before clearing (cannot be undone).

Use Case:

- Remove stale entries after network issues
- Clear test data after development
- Monitor which HLRs are sending alerts

Message Counters

The tracker automatically increments counters:

- **Messages Sent**: Incremented when SRI-for-SM succeeds and MT-FSM is sent
- **Messages Received**: Incremented when MO-FSM is received from subscriber

Auto-Refresh

The page automatically refreshes every **2 seconds** to show real-time subscriber and status updates.

Common Operations

Searching and Filtering

Currently, the Web UI does not include built-in search/filter functionality. To find specific routes:

1. Use your browser's find function (Ctrl+F / Cmd+F)
2. Search for peer names, point codes, or GT prefixes

Bulk Operations

To perform bulk route changes:

1. **Option 1**: Use the REST API for programmatic access
2. **Option 2**: Edit `config/runtime.exs` and restart the application
3. **Option 3**: Use the Web UI for individual route changes

Export/Import

Note: The Web UI does not currently support exporting or importing routing tables. Routes are:

- Stored in Mnesia database files
- Configured in `config/runtime.exs`

To backup routes:

1. **Mnesia**: Backup the `Mnesia.{node_name}/` directory
2. **Config**: Version control `config/runtime.exs`

Auto-Refresh Behavior

Different pages have different refresh intervals:

Page	Refresh Interval	Reason
Routing Management	5 seconds	Route changes are infrequent
Active Subscribers	2 seconds	Subscriber state changes frequently
M3UA Status	Varies by page	Connection state monitoring

WebSocket Connection: All pages use Phoenix LiveView WebSocket connections for real-time updates.

Network Interruption: If the WebSocket connection is lost, the page will attempt to reconnect automatically.

Troubleshooting

Page Not Loading

1. **Check HTTPS Certificate**: Ensure `priv/cert/omnitouch.crt` and `.pem` are present
2. **Verify Port 443**: Check firewall rules allow HTTPS traffic
3. **Application Running**: Confirm application is running with `iex -S mix`
4. **Browser Console**: Check for SSL certificate errors (self-signed cert warnings)

Routes Not Persisting

1. **Check Mnesia Storage**: Verify `mnesia_storage_type: :disc_copies` in config
2. **Mnesia Directory**: Ensure Mnesia directory is writable
3. **Check Logs**: Look for Mnesia errors in application logs

Auto-Refresh Not Working

1. **WebSocket Connection**: Check browser console for WebSocket errors
2. **Network**: Verify stable network connection
3. **Page Reload**: Try refreshing the page (F5)

Related Documentation

- **STP Guide** - Detailed routing configuration
- **HLR Guide** - Subscriber management

- **API Guide** - REST API for programmatic access
- **Configuration Reference** - All configuration parameters

Summary

The OmniSS7 Web UI provides intuitive, real-time management of routing tables and subscriber tracking:

◈ **Real-time Updates** - Auto-refresh keeps data current ◈ **Persistent Storage** - Mnesia ensures routes survive restarts ◈ **Role-Based UI** - Pages adapt to operational mode (STP/HLR/SMSc) ◈ **Interactive Management** - Add, edit, delete routes without restart ◈ **Status Monitoring** - Live connection and peer status

For advanced operations or automation, see the API Guide.