AtlasIED Atmosphere Protocol Implementation Guide

Overview

This document describes the implementation of the AtlasIED Atmosphere AZM4/AZM8 third-party control protocol in the Sports Bar TV Controller application.

Critical Protocol Requirements

Based on the official **ATS006993-B-AZM4-AZM8-3rd-Party-Control.pdf** specification and comprehensive research, the following protocol requirements are CRITICAL for proper communication:

1. Message Terminators

CRITICAL: All messages sent to the Atlas processor MUST be terminated with `\n` (newline), **NOT** `\r\n` (carriage return + newline).

```
// CORRECT
const message = JSON.stringify(command) + '\n'

// WRONG
const message = JSON.stringify(command) + '\r\n'
```

2. Communication Ports

- **TCP Port 5321**: Used for commands (set, bmp, sub, unsub, get) and responses
- **UDP Port 3131**: Used for meter subscription updates (audio levels)

3. Response Format

CRITICAL: GET responses use the method `"getResp"` with `"params"`, **NOT** `"result"`.

```
// Response to GET command
{
  "jsonrpc": "2.0",
  "method": "getResp",
  "params": {
    "param": "ZoneName_0",
    "str": "Main Bar"
  }
}
```

NOT:

```
// WRONG - This is NOT the Atlas protocol
{
  "jsonrpc": "2.0",
  "result": {
    "param": "ZoneName_0",
    "str": "Main Bar"
  },
  "id": 1
}
```

4. Parameter Indexing

All parameters use **0-based indexing**:
- `ZoneSource_0` = Zone 1
- `ZoneName_0` = Zone 1 Name
- `SourceName_0` = Source 1 Name

5. Keep-Alive Mechanism

Send a “get” command for the “KeepAlive” parameter every 4-5 minutes to prevent timeout:

```
{
  "jsonrpc": "2.0",
  "method": "get",
  "params": {
    "param": "KeepAlive",
    "fmt": "str"
  },
  "id": 123
}
```

Expected response:

```
{
  "jsonrpc": "2.0",
  "method": "getResp",
  "params": {
    "param": "KeepAlive",
    "str": "OK"
  }
}
```

6. Subscription Management

CRITICAL: Subscriptions are LOST on disconnect. When reconnecting, you MUST resubscribe to all parameters.

The implementation tracks all subscriptions and automatically resubscribes on reconnection.

Message Types

SET Command

Sets a parameter to an absolute value:

```
{
  "jsonrpc": "2.0",
  "method": "set",
  "params": {
    "param": "ZoneGain_0",
    "val": -20
  }
}
```

Format options:
- `"val"` : Exact value (e.g., dB for gains: -80 to 0)
- `"pct"` : Percentage (0-100)
- `"str"` : String

BUMP Command

Increments/decrements a parameter relatively:

```
{
  "jsonrpc": "2.0",
  "method": "bmp",
  "params": {
    "param": "ZoneGain_0",
    "val": 2
  }
}
```

SUBSCRIBE Command

Requests updates for a parameter (updates sent via TCP or UDP):

```
{
  "jsonrpc": "2.0",
  "method": "sub",
  "params": {
    "param": "ZoneMeter_0",
    "fmt": "val"
  }
}
```

UNSUBSCRIBE Command

Stops updates for a parameter:

```
{
  "jsonrpc": "2.0",
  "method": "unsub",
  "params": {
    "param": "ZoneMeter_0",
    "fmt": "val"
  }
}
```

GET Command

Retrieves current value (one-time):

```
{
  "jsonrpc": "2.0",
  "method": "get",
  "params": {
    "param": "ZoneName_0",
    "fmt": "str"
  },
  "id": 1
}
```

UPDATE (Received)

For subscribed parameter changes:

```
{
  "jsonrpc": "2.0",
  "method": "update",
  "params": {
    "param": "ZoneGain_0",
    "pct": 50
  }
}
```

Parameter Reference

Common Parameters

Parameter	Min Val	Max Val	Format	Read-Only	Description
SourceGain	-80	0	val, pct	No	Source input gain
SourceMeter	-80	0	val, pct	Yes	Source audio level (UDP)
SourceMute	0	1	val	No	Source mute state
SourceName	N/A	N/A	str	Yes	Source name (user-configured)
ZoneGain	-80	0	val, pct	No	Zone output gain
ZoneMeter	-80	0	val, pct	Yes	Zone audio level (UDP)
ZoneMute	0	1	val	No	Zone mute state
ZoneName	N/A	N/A	str	Yes	Zone name (user-configured)
ZoneSource	-1	N	val	No	Zone source assignment (-1=none)
GroupActive	0	1	val	No	Group activation (combine zones)
RecallScene	N/A	N/A	val	No	Trigger scene recall
PlayMessage	N/A	N/A	val	No	Trigger message playback

Read-Only Parameters

The following parameters can ONLY be queried (get/sub/unsub), not set:
- `SourceName_X`
- `ZoneName_X`
- `MixName_X`
- `GroupName_X`
- `SourceMeter_X`
- `ZoneMeter_X`
- `ZoneGrouped_X`
- All `*Name` parameters

Implementation Details

Client Class: `AtlasTCPClient`

The main client class (`src/lib/atlasClient.ts`) implements:

1. **TCP Connection** on port 5321 for commands/responses
2. **UDP Socket** on port 3131 for meter updates
3. **Keep-Alive Timer** (every 4 minutes)
4. **Automatic Reconnection** with exponential backoff
5. **Subscription Tracking** for automatic resubscription
6. **Comprehensive Logging** via `atlasLogger`

Hardware Query Service

The hardware query service (`src/lib/atlas-hardware-query.ts`) queries the Atlas processor for:

1. **Source Names** (`SourceName_0` , `SourceName_1` , etc.)
2. **Zone Names** (`ZoneName_0` , `ZoneName_1` , etc.)
3. **Real-time Zone Status**:
   - Current source assignment
   - Volume level (percentage)
   - Mute state

Usage Examples

Connect and Query Zone Name

```typescript
import { AtlasTCPClient } from '@/lib/atlasClient'

const client = new AtlasTCPClient({
  ipAddress: '192.168.5.101',
  tcpPort: 5321,
  udpPort: 3131
})

await client.connect()

// Get zone name
const response = await client.getParameter('ZoneName_0', 'str')
if (response.success && response.data.value) {
  console.log(`Zone 1 name: ${response.data.value}`)
}

client.disconnect()
```

Set Zone Volume

```typescript
// Set Zone 1 volume to 75%
const response = await client.setZoneVolume(0, 75, true)
if (response.success) {
  console.log('Volume set successfully')
}
```

Set Zone Source

```typescript
// Set Zone 1 to Source 3
const response = await client.setZoneSource(0, 2) // 0-indexed
if (response.success) {
  console.log('Source set successfully')
}
```

Subscribe to Meter Updates

```typescript
// Subscribe to Zone 1 meter updates
await client.subscribe('ZoneMeter_0', 'val')

// Override handleParameterUpdate for custom handling
class CustomAtlasClient extends AtlasTCPClient {
  protected handleParameterUpdate(param: string, value: any, fullParams: any): void {
    console.log(`Update: ${param} = ${value}`)
  }
}
```

Common Issues and Solutions

Issue: Connection Timeout

Symptom: `Connection timeout` error when trying to connect.

Solution:
1. Verify the Atlas processor is powered on and network accessible
2. Confirm you're using TCP port 5321 (NOT 80, NOT 23)
3. Check firewall rules allow TCP connections to port 5321

Issue: No Response to GET Commands

Symptom: `Command timeout` when sending GET commands.

Solution:
1. Ensure messages are terminated with `\n` (not `\r\n`)
2. Verify response parser looks for `method: "getResp"` with `params`
3. Check the parameter name is correct (case-sensitive)

Issue: Zone/Source Names Not Displaying

Symptom: Generic "Zone 1", "Source 1" instead of actual names.

Solution:
1. Verify names are configured in the Atlas web interface
2. Check the parameter format is `"str"` not `"val"` for name queries
3. Look for the value in `response.params.str` (not `response.result`)
4. Ensure proper response parsing extracts the `value` field

Issue: Subscriptions Stop After Reconnect

Symptom: No meter updates after connection is re-established.

Solution:
1. The implementation automatically resubscribes on reconnection
2. Verify the `subscriptions` Set is being maintained
3. Check the `resubscribeAll()` method is called on reconnect

Testing Checklist

- [ ] TCP connection to port 5321 succeeds
- [ ] UDP socket binds to port 3131
- [ ] GET command returns proper `getResp` response
- [ ] Zone names are retrieved correctly
- [ ] Source names are retrieved correctly
- [ ] Volume control works (set and verify)
- [ ] Source switching works (set and verify)
- [ ] Mute control works (set and verify)
- [ ] Keep-alive sends every 4 minutes
- [ ] Reconnection works after network interruption
- [ ] Subscriptions are restored after reconnect
- [ ] Meter updates are received via UDP

References

1. **ATS006993-B-AZM4-AZM8-3rd-Party-Control.pdf** - Official Atlas protocol specification

2. **Grok Specification Document** - Comprehensive protocol guide (see `/Uploads/user_message_2025-10-21_02-53-04.txt`)
3. **Atlas Web Interface** - Third Party Control Message Table (Settings > Third Party Control > Message Table)

Key Differences from Previous Implementation

❌ Previous (Incorrect) Implementation

```
// WRONG: Using \r\n terminator
const message = JSON.stringify(command) + '\r\n'

// WRONG: Expecting "result" key
if (response.result) {
  const value = response.result.str
}

// WRONG: Using port 80
const client = new AtlasTCPClient({ ipAddress: '192.168.5.101', port: 80 })
```

✅ Current (Correct) Implementation

```
// CORRECT: Using \n terminator
const message = JSON.stringify(command) + '\n'

// CORRECT: Expecting "getResp" method with "params"
if (response.method === 'getResp' && response.params) {
  const value = response.params.str
}

// CORRECT: Using port 5321
const client = new AtlasTCPClient({ ipAddress: '192.168.5.101', tcpPort: 5321 })
```

Deployment Notes

1. **Environment**: The Atlas processor must be on the same network or routable to the application server
2. **Firewall**: Ensure TCP port 5321 is open for outbound connections
3. **Database**: Update the `audioProcessors` table with correct `tcpPort` (5321) and `port` (80 for HTTP)
4. **Credentials**: Store Atlas web interface credentials in the database for HTTP configuration discovery
5. **Monitoring**: Check `~/Sports-Bar-TV-Controller/log/atlas-communication.log` for detailed protocol logs

Support

For issues related to the Atlas protocol implementation:

1. Review the comprehensive logs in `log/atlas-communication.log`
2. Verify the Atlas processor configuration in the web interface
3. Test connectivity using the built-in test scripts

4. Consult the official Atlas documentation

Last Updated: October 21, 2025
Implementation Version: 2.0 (Correct Protocol)
Specification: ATS006993-B RevB 4/22