Atlas IED Atmosphere Audio Processor Integration Guide

Table of Contents

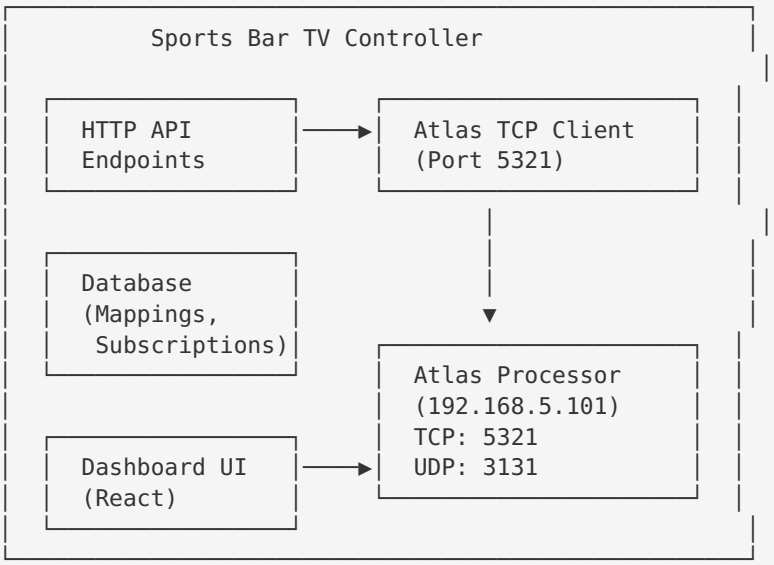
Overview

The Sports Bar TV Controller application includes comprehensive integration with AtlasIED Atmosphere AZM4/AZM8 audio processors. This integration provides:

- **Real-time Control**: Set zone volumes, mute/unmute, and switch sources
- **Parameter Mapping**: Map user-friendly names to Atlas parameter names
- **Batch Operations**: Execute multiple commands in a single request
- **Subscriptions**: Subscribe to parameter updates for real-time monitoring
- **Scene Management**: Trigger preset scenes and configurations
- **AI Integration**: Automatic gain control and audio monitoring

Architecture

Components



Database Schema

AtlasMapping - Maps app-friendly names to Atlas parameters
- `appKey`: User-friendly key (e.g., 'mainBarGain', 'zone1Volume')
- `atlasParam`: Atlas parameter name (e.g., 'ZoneGain_0')
- `paramType`: Type of parameter (zone, source, mix, group, etc.)
- `paramCategory`: Category (gain, mute, source, meter, etc.)
- `format`: Data format ('val', 'pct', or 'str')
- `minValue/maxValue`: Range for gain parameters (typically -80 to 0 dB)

AtlasSubscription - Tracks active parameter subscriptions
- `paramName`: Atlas parameter being monitored
- `format`: Subscription format
- `subscriptionType`: TCP or UDP
- `isActive`: Whether subscription is currently active

Setup and Configuration

1. Enable Third-Party Control on Atlas

1. Open the Atlas Atmosphere UI in your browser (http://192.168.5.101)
2. Log in with your credentials (default: admin/password)
3. Navigate to **Settings → Third Party Control**
4. Enable **Third Party Control**
5. Set TCP Port to **5321**
6. Set UDP Port to **3131**
7. Save configuration

2. Configure Audio Processor in Application

1. Navigate to the **Audio Control** page

2. Click **Add Processor**
3. Enter the following details:
   - **Name**: Your processor name (e.g., “Main Atlas”)
   - **Model**: AZM4 or AZM8
   - **IP Address**: 192.168.5.101
   - **TCP Port**: 5321
   - **Username**: admin (optional)
   - **Password**: Your password (optional)
4. Click **Save**

3. Environment Variables

Add to your `.env` file:

```
# Atlas Audio Processor Configuration
ATLAS_IP_ADDRESS=192.168.5.101
ATLAS_TCP_PORT=5321
ATLAS_UDP_PORT=3131
ATLAS_USERNAME=admin
ATLAS_PASSWORD=your_password_here
```

Parameter Mappings

Finding Atlas Parameter Names

Parameter names are dynamically assigned in the Atlas UI based on your configuration. To find them:

1. Open Atlas UI → **Settings → Third Party Control → Message Table**
2. Locate your zone/source in the **Names** column
3. Follow the row to the desired parameter column (Gain, Mute, Source, etc.)
4. Note the parameter name (e.g., `ZoneGain_0`, `SourceMute_2`)

Common Parameter Types

Zone Parameters

- `ZoneGain_N` - Zone gain/volume (-80 to 0 dB or 0-100%)
- `ZoneMute_N` - Zone mute state (0=unmuted, 1=muted)
- `ZoneSource_N` - Zone source selection (0-based index, -1=no source)
- `ZoneMeter_N` - Zone audio level meter (read-only)
- `ZoneName_N` - Zone name (read-only)
- `ZoneGrouped_N` - Whether zone is in a group (read-only)

Source Parameters

- `SourceGain_N` - Source gain (-80 to 0 dB or 0-100%)
- `SourceMute_N` - Source mute state (0=unmuted, 1=muted)
- `SourceMeter_N` - Source audio level meter (read-only)
- `SourceName_N` - Source name (read-only)

Group Parameters

- `GroupGain_N` - Group gain (-80 to 0 dB or 0-100%)
- `GroupMute_N` - Group mute state (0=unmuted, 1=muted)
- `GroupSource_N` - Group source selection

- `GroupActive_N` - Activate/deactivate group (0=inactive, 1=active)

Action Parameters

- `RecallScene` - Recall a preset scene (pass scene index as value)
- `PlayMessage` - Play a pre-recorded message (pass message index)
- `RecallGpoPreset` - Recall GPO preset

Creating Mappings

Via Dashboard UI

1. Navigate to **Atlas Config → Parameter Mappings**
2. Click **Add Mapping**
3. Fill in the form:
   - **App Key**: User-friendly name (e.g., `mainBarVolume`)
   - **Atlas Parameter**: Atlas name from Message Table (e.g., `ZoneGain_0`)
   - **Parameter Type**: zone, source, mix, group, etc.
   - **Category**: gain, mute, source, meter, etc.
   - **Format**: val (dB), pct (percentage), or str (string)
   - **Min/Max Values**: Range for gain parameters (-80 to 0)
   - **Description**: Optional description
4. Click **Create**

Via API

```
curl -X POST http://localhost:3000/api/atlas/mappings \
  -H "Content-Type: application/json" \
  -d '{
    "processorId": "processor-uuid",
    "appKey": "mainBarVolume",
    "atlasParam": "ZoneGain_0",
    "paramType": "zone",
    "paramCategory": "gain",
    "format": "pct",
    "minValue": -80,
    "maxValue": 0,
    "minPercent": 0,
    "maxPercent": 100,
    "description": "Main bar zone volume"
  }'
```

API Endpoints

Set Gain

Set the gain/volume for a zone or source.

Endpoint: `POST /api/atlas/set-gain`

Request Body:

```
{
  "processorId": "processor-uuid",
  "appKey": "mainBarVolume",
  "value": 75,
  "format": "pct"
}
```

Alternative (using Atlas parameter directly):

```
{
  "processorId": "processor-uuid",
  "atlasParam": "ZoneGain_0",
  "value": -10,
  "format": "val"
}
```

Response:

```
{
  "success": true,
  "data": {
    "processorId": "processor-uuid",
    "appKey": "mainBarVolume",
    "atlasParam": "ZoneGain_0",
    "value": 75,
    "format": "pct",
    "response": { ... }
  }
}
```

Set Mute

Mute or unmute a zone or source.

Endpoint: `POST /api/atlas/set-mute`

Request Body:

```
{
  "processorId": "processor-uuid",
  "appKey": "mainBarMute",
  "muted": true
}
```

Response:

```
{
  "success": true,
  "data": {
    "processorId": "processor-uuid",
    "appKey": "mainBarMute",
    "atlasParam": "ZoneMute_0",
    "muted": true,
    "response": { ... }
  }
}
```

Trigger Scene

Recall a preset scene.

Endpoint: `POST /api/atlas/trigger-scene`

Request Body:

```
{
  "processorId": "processor-uuid",
  "sceneIndex": 0
}
```

Or with app key:

```
{
  "processorId": "processor-uuid",
  "appKey": "gameTimeScene"
}
```

Get Status

Get current value of a parameter.

Endpoint: `GET /api/atlas/get-status?processorId=xxx&appKey=mainBarVolume`

Response:

```
{
  "success": true,
  "data": {
    "processorId": "processor-uuid",
    "appKey": "mainBarVolume",
    "atlasParam": "ZoneGain_0",
    "format": "pct",
    "value": 75,
    "response": { ... }
  }
}
```

Batch Operations

Execute multiple commands in a single request.

Endpoint: `POST /api/atlas/batch`

Request Body:

```
{
  "processorId": "processor-uuid",
  "commands": [
    {
      "method": "set",
      "appKey": "zone1Volume",
      "value": 75
    },
    {
      "method": "set",
      "appKey": "zone2Volume",
      "value": 80
    },
    {
      "method": "set",
      "appKey": "zone1Mute",
      "value": 0
    }
  ]
}
```

Response:

```
{
  "success": true,
  "data": {
    "processorId": "processor-uuid",
    "totalCommands": 3,
    "successfulCommands": 3,
    "failedCommands": 0,
    "results": [ ... ]
  }
}
```

Subscribe to Updates

Subscribe to parameter updates for real-time monitoring.

Endpoint: `POST /api/atlas/subscribe`

Request Body (Subscribe):

```
{
  "processorId": "processor-uuid",
  "appKey": "zone1Meter",
  "subscribe": true
}
```

Request Body (Unsubscribe):

```
{
  "processorId": "processor-uuid",
  "appKey": "zone1Meter",
  "subscribe": false
}
```

Get All Subscriptions: `GET /api/atlas/subscribe?processorId=xxx`

Parameter Mappings Management

Get All Mappings: `GET /api/atlas/mappings?processorId=xxx`

Create Mapping: `POST /api/atlas/mappings` (see Creating Mappings section)

Update Mapping: `PUT /api/atlas/mappings`

```
{
  "id": "mapping-uuid",
  "description": "Updated description",
  "minValue": -70
}
```

Delete Mapping: `DELETE /api/atlas/mappings?id=xxx`

Usage Examples

Example 1: Set Main Bar Volume to 75%

```
const response = await fetch('/api/atlas/set-gain', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    processorId: 'your-processor-id',
    appKey: 'mainBarVolume',
    value: 75,
    format: 'pct'
  })
});

const data = await response.json();
console.log(data);
```

Example 2: Mute All Zones

```
const response = await fetch('/api/atlas/batch', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    processorId: 'your-processor-id',
    commands: [
      { method: 'set', appKey: 'zone1Mute', value: 1 },
      { method: 'set', appKey: 'zone2Mute', value: 1 },
      { method: 'set', appKey: 'zone3Mute', value: 1 },
      { method: 'set', appKey: 'zone4Mute', value: 1 }
    ]
  })
});

const data = await response.json();
console.log(`Muted ${data.data.successfulCommands} zones`);
```

Example 3: Recall “Game Time” Scene

```
const response = await fetch('/api/atlas/trigger-scene', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    processorId: 'your-processor-id',
    sceneIndex: 0  // or use appKey: 'gameTimeScene'
  })
});

const data = await response.json();
console.log('Scene recalled:', data);
```

Example 4: Subscribe to Zone Meters

```
// Subscribe to all zone meters
const zones = ['zone1Meter', 'zone2Meter', 'zone3Meter', 'zone4Meter'];

for (const appKey of zones) {
  const response = await fetch('/api/atlas/subscribe', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      processorId: 'your-processor-id',
      appKey,
      subscribe: true
    })
  });

  const data = await response.json();
  console.log(`Subscribed to ${appKey}:`, data.success);
}
```

Troubleshooting

Connection Issues

Problem: Cannot connect to Atlas processor

Solutions:
1. Verify Atlas IP address is correct (192.168.5.101)
2. Ensure TCP port 5321 is open on the Atlas device
3. Check that Third-Party Control is enabled in Atlas settings
4. Verify network connectivity: `ping 192.168.5.101`
5. Check firewall rules on both devices

Test Connection:

```
telnet 192.168.5.101 5321
```

Parameter Not Found

Problem: Mapping not found or parameter doesn’t work

Solutions:
1. Verify parameter name in Atlas UI (Settings → Third Party Control → Message Table)
2. Ensure spelling and capitalization are exact (case-sensitive)
3. Check that the parameter exists in your Atlas configuration
4. Verify processor ID is correct in mapping

No Response from Commands

Problem: Commands sent but no response received

Solutions:
1. Check command timeout (default 5 seconds)
2. Verify Atlas is not busy processing other commands
3. Check Atlas logs for errors
4. Ensure parameter is not read-only (like meters and names)
5. Verify value is within valid range

Subscription Updates Not Received

Problem: Subscribed to parameter but not receiving updates

Solutions:
1. Verify subscription was successful (check response)
2. Ensure UDP port 3131 is open for meter subscriptions
3. Check that parameter value is actually changing
4. Verify subscription is marked as active in database
5. Check network packet loss

Best Practices

1. Use Parameter Mappings

Always use app-friendly names (appKeys) instead of direct Atlas parameters. This provides:
- Better code readability
- Easier maintenance
- Protection against Atlas configuration changes
- Centralized parameter management

2. Batch Operations for Multiple Commands

When sending multiple commands, use the batch endpoint instead of individual requests:
- Reduces network overhead
- Faster execution
- Atomic operations
- Better error handling

3. Subscribe Wisely

Only subscribe to parameters you actively monitor:
- Reduces network traffic
- Minimizes database updates
- Improves performance
- Consider unsubscribing when not needed

4. Handle Errors Gracefully

Always check response status and handle errors:

```
const response = await fetch('/api/atlas/set-gain', { ... });
const data = await response.json();

if (!data.success) {
  console.error('Failed to set gain:', data.error);
  // Handle error appropriately
}
```

5. Use Percentage Format for User-Facing Controls

Percentages (0-100) are more intuitive than dB values (-80 to 0) for end users:

```
// User-friendly
{ format: 'pct', value: 75 }

// Technical
{ format: 'val', value: -10 }
```

6. Implement Connection Retry Logic

Network issues can occur, so implement retry logic:

```
async function setGainWithRetry(params, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      const response = await fetch('/api/atlas/set-gain', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(params)
      });

      const data = await response.json();
      if (data.success) return data;

      if (i < maxRetries - 1) {
        await new Promise(resolve => setTimeout(resolve, 1000 * (i + 1)));
      }
    } catch (error) {
      if (i === maxRetries - 1) throw error;
    }
  }
  throw new Error('Max retries exceeded');
}
```

7. Document Your Mappings

Keep clear documentation of your parameter mappings:
- What each appKey controls
- Valid value ranges
- Dependencies between parameters
- When to use each parameter

8. Test in Development First

Always test new mappings and commands in a development environment:
- Verify parameter names are correct
- Check value ranges
- Test error handling
- Monitor for unexpected behavior

9. Monitor Connection Health

Regularly check Atlas connection status:
- Implement health check endpoints
- Log connection errors
- Alert on repeated failures
- Track reconnection attempts

10. Version Control Configuration

Store Atlas configuration in version control:
- Parameter mappings (export/import feature)
- Scene configurations
- Network settings
- Documentation

Advanced Topics

Custom Parameter Ranges

Some scenarios require custom value ranges:

```
// Normalize a 0-100 slider to -60 to 0 dB range
function normalizeToDb(percentage) {
  const minDb = -60;
  const maxDb = 0;
  return minDb + (percentage / 100) * (maxDb - minDb);
}

const userValue = 75; // 75% from UI slider
const dbValue = normalizeToDb(userValue); // -15 dB

await fetch('/api/atlas/set-gain', {
  method: 'POST',
  body: JSON.stringify({
    processorId: 'xxx',
    appKey: 'mainBarVolume',
    value: dbValue,
    format: 'val'
  })
});
```

Scene-Based Control

Create preset scenes for common scenarios:

```
// Define scenes
const scenes = {
  quietHours: 0,
  normalOperation: 1,
  gameTime: 2,
  lateNight: 3
};

// Trigger scene
async function activateScene(sceneName) {
  const sceneIndex = scenes[sceneName];
  const response = await fetch('/api/atlas/trigger-scene', {
    method: 'POST',
    body: JSON.stringify({
      processorId: 'xxx',
      sceneIndex
    })
  });
  return response.json();
}

// Usage
await activateScene('gameTime');
```

Integration with AI Assistant

The application includes AI integration for voice control:

```
// Example: "Hey, turn up the main bar volume"
// AI parses intent and executes:
await fetch('/api/atlas/set-gain', {
  method: 'POST',
  body: JSON.stringify({
    processorId: 'xxx',
    appKey: 'mainBarVolume',
    value: currentValue + 10, // Increase by 10%
    format: 'pct'
  })
});
```

Support

For issues or questions:

1. Check the Troubleshooting section
2. Review Atlas IED documentation (ATS006993-B-AZM4-AZM8-3rd-Party-Control.pdf)
3. Check application logs for detailed error messages
4. Contact your system administrator

References

- Atlas IED Atmosphere Documentation (https://www.atlasied.com)
- JSON-RPC 2.0 Specification (https://www.jsonrpc.org/specification)
- Sports Bar TV Controller GitHub Repository (https://github.com/dfultonthebar/Sports-Bar-TV-Controller)