Catena Manual

Catena

Flussonic

Table of contents

1. Products

2. Catena

2.0.1 Catena

Catena is a comprehensive IPTV service management solution consisting of server software and client applications for Smart TV, mobile devices, and set-top boxes. The system is designed to deliver both paid and free television to end subscribers.

Catena client applications are available for the following platforms:

- **Samsung Tizen** — application for Samsung Smart TV
- **LG WebOS** — application for LG Smart TV
- **Android** — application for Android TV, mobile devices, and tablets

Content is not included in the Catena delivery — it is always provided by the video streaming service operator.

Target Audience

Catena is designed for **video streaming service operators** who want to launch their own IPTV service. This can be:

- **Telecom operators** — Internet service providers who want to offer their subscribers interactive television services
- **OTT operators** — companies that provide video content over the Internet
- **Cable operators** — traditional TV operators transitioning to IP technologies
- **Corporate clients** — organizations creating their own internal TV services for employees or customers
- **Media companies** — publishers and broadcasters who want to create a direct connection with their audience

Key Capabilities

Catena provides a complete set of tools for managing an IPTV service:

CONTENT MANAGEMENT

- **TV Channel Management** — create, configure, and organize broadcast channels. Each channel has a unique name for streaming, a display title for users, a logo, and a link to an EPG source
- **Channel Package Management** — create tariff packages from groups of channels. Packages are used to sell channel bundles to subscribers
- **Electronic Program Guide (EPG) Management** — connect and synchronize electronic program guide sources, display program schedules for each channel

SUBSCRIBER MANAGEMENT

- **Subscriber Management** — register subscribers, manage their data (name, phone), generate tokens for content playback
- **Subscription Management** — assign channel packages to subscribers, manage content access, support both paid and free packages

MONITORING AND ANALYTICS

- **Playback Sessions** — view current and historical viewing sessions: which channels subscribers are watching, from which devices, how much data has been transferred
- **Operations Log** — detailed history of all actions in the system (creating/deleting subscribers, changing subscriptions, etc.) with filtering and audit capabilities

ADMINISTRATION

- **Portal Settings** — configure service parameters: name, domain, branding (logo, description), API key management, free package configuration
- **Manager Management** — create administrator accounts with different access levels: infrastructure management, subscriber management, content management

Web Interface and API

Catena provides two ways to manage the system:

WEB INTERFACE (UI)

Through the web interface, administrators can:

- **Visually manage all entities** — create, edit, and delete channels, packages, subscribers through a convenient graphical interface
- **View real-time statistics** — see active viewing sessions, number of subscribers, channel popularity
- **Upload logos and images** — add visual elements for channels and portal
- **Manage program guide** — view and update EPG, link channels to program sources
- **Administer access** — create managers with different permissions, manage API keys
- **Track operation history** — view logs of all actions for audit and control

MANAGEMENT API

The API is designed for programmatic integration of Catena with the operator's external systems:

- **Subscriber Management Automation** — integration with billing systems for automatic creation/blocking of subscribers on payment/non-payment of services
- **CRM Integration** — synchronization of subscriber data, subscriptions, and operations with the operator's CRM system
- **Dynamic Content Management** — programmatic addition/removal of channels and packages, metadata updates
- **Analytics Retrieval** — export viewing statistics for external analytics and reporting systems
- **EPG Updates** — programmatic trigger for updating program guide from external sources
- **Portal Monitoring** — programmatic reading of portal settings and status

The API is built on REST standards using JSON for data exchange. Authentication is performed through API keys (`X-Auth-Token` header), which can be generated in the web interface. The API supports pagination for working with large amounts of data.

TYPICAL USE CASES

Small-scale operator:

- Primarily uses the web interface for manual management of a small number of subscribers (up to several thousand)
- Manually creates channels and packages
- Registers subscribers through the UI when contacting support

Medium and large-scale operator:

- Uses API for automatic synchronization with billing and CRM
- Automates creation/blocking of subscribers when payment status changes
- Uses the web interface for monitoring, analytics, and manual operations
- Configures automatic EPG updates through API

Documentation Structure

Catena documentation is organized into sections according to the operator's main tasks:

1. **Quick Start** — step-by-step guide for initial setup
2. **Content Management** — working with channels, packages, and EPG
3. **Subscriber Management** — registration and service of subscribers
4. **Monitoring and Analytics** — viewing statistics and sessions
5. **Administration** — portal configuration and access management
6. **API Reference** — complete reference for all API methods
7. **Integration** — examples of integration with external systems

---

If you are just starting to work with Catena, we recommend starting with the **Quick Start** section, which will guide you through the main stages of system setup.

2.1 Portal Management

A portal is an independent branded domain with its own set of subscribers, channels, and packages in the Catena system. Portals allow managing multiple IPTV services with different brands on a single infrastructure.

2.1.1 What is a Portal

A portal in Catena is an isolated space for a separate IPTV service with its own settings, subscribers, and content. Each portal represents an independent branded service with its own domain, logo, and visual design.

Key concept:

```
Catena Infrastructure
  ├── Portal "Netflix-like service" (myiptv.com)
  │   ├── Subscribers: 10,000
  │   ├── Channels: 200
  │   └── Branding: red logo, modern design
  ├── Portal "Regional provider" (region-tv.com)
  │   ├── Subscribers: 5,000
  │   ├── Channels: 150
  │   └── Branding: blue logo, classic style
  └── Portal "Sports service" (sport-tv.com)
      ├── Subscribers: 3,000
      ├── Channels: 50 (sports only)
      └── Branding: green logo, dynamic design
```

Main portal characteristics:

- **Independent domain** — each portal accessible via its own URL
- **Own branding** — logo, name, description, visual design
- **Isolated data** — subscribers of one portal not visible in another
- **Separate content** — own set of channels, packages, EPG
- **Branded applications** — ability to create mobile app for portal
- **Shared infrastructure** — all portals run on same servers

Why portals are needed:

1. **Multi-branding** — managing multiple IPTV brands
2. **White-label solutions** — providing service under client's brand
3. **Geographic separation** — different portals for different regions
4. **Audience segmentation** — premium and budget services on one platform
5. **Partner projects** — separate portals for B2B partners
6. **Testing environment** — separate portal for testing new features

2.1.2 Managing Multiple Portals

Multi-tenancy Concept

One infrastructure — multiple portals:

- All portals use the same streaming servers
- Content can be shared or unique to portal
- One manager can control multiple portals
- Billing system can serve all portals

Benefits:

- **Resource savings** — one server for all portals
- **Centralized management** — single admin panel
- **Shared content** — same channels for different brands
- **Flexible marketing** — different strategies for each portal
- **Scalability** — easy to add new portals

One Manager — Multiple Portals

Scenario: IPTV business owner manages three brands

```
Manager: admin@company.com
  ├── Portal 1: myiptv.com (owner, full access)
  ├── Portal 2: premium-tv.com (owner, full access)
  └── Portal 3: budget-tv.com (content admin, content management)
```

How access works:

- Manager is created separately for each portal
- One email can be used across different portals
- On login, system shows list of available portals
- Manager selects portal to work with
- API key is bound to specific portal

Manager permission types:

- **isAdmin** — infrastructure management (creating portals, servers)
- **canManage** — full portal management (owner)
- **canManageSubscribers** — subscriber and subscription management
- **canManageContent** — channels, packages, EPG management

2.1.3 Main Portal Parameters

Technical Parameters

Portal ID

- Automatically generated when creating portal
- Format: base64-encoded Snowflake ID with `+/=` replaced by `-_.`
- Example: `pKl9SW3AAAE.`
- Used in all API requests
- Links all entities (subscribers, channels, packages) to portal

Internal Name (Name)

- Technical portal name in system
- Visible only to administrators and portal owner
- Used for identification in logs and control panel
- Must be unique in system
- Examples: `catena-netflix`, `my-iptv-service`, `test-portal`

Domain

- Domain name under which portal is accessible
- Specified by portal owner as domain claim
- Real DNS binding performed by system administrator
- Used for branded mobile applications
- Example: `myiptv.com`, `tv.example.org`

Owner ID

- Identifier of manager — portal owner
- Owner has full access to all settings
- Can assign other managers
- Set when creating portal

API Key

- Unique key for Management API portal access
- Generated automatically when creating portal
- Used for authentication of all API requests
- Can be regenerated via `/portal/reset_api_key`
- Must be stored securely

Branding Parameters

Logo

- URL or base64-encoded image of portal logo
- Displayed in mobile apps and web interface
- Seen by end users (subscribers)
- Recommended format: PNG with transparency
- Recommended size: 512x512px or higher

Title

- Public portal name for end users
- Displayed in applications, website, notifications
- Examples: "My IPTV", "Premium TV Service", "Sport TV"

Description

- Brief service description for users
- Used in app stores, landing pages
- Can contain slogan or brief benefits description
- Example: "Best IPTV for the whole family. 200+ channels in HD quality"

Free Packages

Concept:

- List of packages available to all portal subscribers automatically
- No need to create subscription for each subscriber
- Used for basic content, demo channels, trial period

Applications:

- **Basic channels** — public, freely available channels
- **Trial access** — first month for all new subscribers
- **Promo content** — advertising and informational channels
- **Loyalty program** — bonus channels for all clients

Management:

```
# Add package to free list
POST /portal/free-packages/{packageId}

# Remove package from free list
DELETE /portal/free-packages/{packageId}
```

2.1.4 Getting Portal Information

Via Web Interface

1. **Log into Catena control panel**
2. **Select portal** (if you have access to multiple)
3. **Open "Portal Settings" section**
4. **View parameters:**
5. Basic information (name, domain)
6. Branding (logo, description)
7. API key
8. Free packages list
9. Access permissions

Via Management API

Get current portal information:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/portal \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "portalId": "pKl9SW3AAAE.",
  "name": "my-iptv-service",
  "domain": "myiptv.com",
  "freePackages": ["basicKl9SW3AAAE.", "trialKl9SW3AAAE."],
  "branding": {
    "logo": "https://myiptv.com/logo.png",
    "title": "My IPTV Service",
    "description": "Premium IPTV streaming for everyone"
  },
  "apiKey": "secret_api_key_1234567890",
  "ownerId": "mKl9SW3AAAE.",
  "createdAt": "2024-01-15T10:00:00Z",
  "updatedAt": "2024-10-16T14:30:00Z",
  "flags": {
    "canManage": true,
    "canManageSubscribers": true,
    "canManageContent": true
  }
}
```

Response fields:

- **portalId** — unique portal identifier
- **name** — internal technical name
- **domain** — portal domain name
- **freePackages** — array of free package IDs
- **branding** — branding parameters
- **apiKey** — API key for authentication
- **ownerId** — portal owner ID
- **createdAt/updatedAt** — creation and update dates
- **flags** — current manager's access permissions

2.1.5 Editing a Portal

Via Web Interface

1. **Open "Portal Settings" section**
2. **Click "Edit"**
3. **Change parameters:**
4. User-facing title (Title)
5. Service description (Description)
6. Logo URL (Logo)
7. **Save changes**

Important: Technical parameters (name, domain, portalId) usually not editable after creation.

Via Management API

Update portal parameters:

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/portal \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "my-iptv-service",
    "branding": {
      "logo": "https://myiptv.com/new-logo.png",
      "title": "My IPTV - New Name",
      "description": "Updated service description"
    }
  }'
```

Response:

Updated Portal object with new values.

What can be changed:

- Branding parameters (logo, title, description)
- Free packages list (via separate endpoints)

What cannot be changed:

- portalId — generated automatically
- name — set on creation
- domain — set on creation
- ownerId — changed separately by administrator

2.1.6 API Key Management

API Key Security

API key is a secret token for portal access. Handle it carefully:

- Store in secure place (environment variables, secret manager)
- Don't commit to Git repositories
- Don't share with third parties
- Regularly update (every 6-12 months)
- Update immediately if compromise suspected

API Key Regeneration

When to regenerate:

- API key accidentally pushed to public repository
- Suspected unauthorized access
- Employee with key access terminated
- Scheduled update per security policy
- Integration or billing system change

Via Web Interface:

1. **Open "Portal Settings"**
2. **Go to "Security" section**
3. **Click "Generate New API Key"**
4. **Confirm action**
5. **Copy new key** (old one stops working immediately)
6. **Update key in all integrations**

Via Management API:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/portal/reset_api_key \
  -H "X-Auth-Token: current-api-key"
```

Response:

```
{
  "portalId": "pKl9SW3AAAE.",
  "name": "my-iptv-service",
  "apiKey": "new_secret_api_key_0987654321",
  "branding": { ... },
  ...
}
```

Important:

- Old API key stops working immediately
- All current integrations with old key will start getting 401 error
- Update key everywhere: billing, monitoring, scripts
- Save new key in secure location

2.1.7 Managing Free Packages

Adding Free Package

Via Web Interface:

1. **Open "Portal Settings"**
2. **Go to "Free Packages" section**
3. **Click "Add Package"**
4. **Select package** from available list
5. **Confirm addition**

All portal subscribers immediately get access to this package's channels.

Via Management API:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/basicKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response: HTTP 201 Created

Removing Free Package

Via Web Interface:

1. **Open "Portal Settings"**
2. **Go to "Free Packages" section**
3. **Find package** in list
4. **Click "Remove"**
5. **Confirm removal**

Subscribers without explicit subscription to this package will lose access to its channels.

Via Management API:

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/basicKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response: HTTP 201 Created

Important:

- If subscriber has explicit subscription to package, they retain access
- Removing from free doesn't delete the package itself
- Changes take effect immediately

2.1.8 Typical Use Cases

Scenario 1: Multiple Brands on One Platform

Task: Company manages three IPTV brands

Structure:

```
Company "IPTV Group"
├── Brand "Premium TV" (premium-tv.com)
│   ├── Target audience: premium segment
│   ├── Content: 300 HD/4K channels
│   ├── Price: from $20/month
│   └── Branding: gold logo, elegant design
│
├── Brand "Family TV" (family-tv.com)
│   ├── Target audience: families with kids
```

```
│   ├── Content: 150 channels (movies, kids, general)
│   ├── Price: from $10/month
│   └── Branding: bright colors, friendly design
│
└── Brand "Sport TV" (sport-tv.com)
    ├── Target audience: sports fans
    ├── Content: 50 sports channels
    ├── Price: from $15/month
    └── Branding: dynamic, energetic style
```

Benefits:

- One streaming server for all brands
- Centralized content management
- Different marketing strategies
- Isolated subscriber bases
- Infrastructure cost savings

Implementation:

1. Create 3 portals in Catena
2. Configure branding for each
3. Distribute channels across portals
4. Create packages with different pricing
5. Integrate with unified billing system
6. Deploy branded mobile apps

Scenario 2: White-label Solution for Partners

Task: Provide IPTV platform to partners under their brand

Business model:

- You — infrastructure and content provider
- Partners — subscriber base and brand owners
- Each partner gets their own portal
- Partner pays per subscriber count or fixed fee

Example structure:

```
Your platform: catena-platform.com

├── Partner 1: regional-provider.com
│   └── 5,000 subscribers
│
├── Partner 2: city-tv.com
│   └── 3,000 subscribers
│
└── Partner 3: corporate-tv.net
    └── 1,000 subscribers (corporate TV)
```

What partner gets:

- Own portal with unique domain
- Full control over branding
- Access to your channel catalog
- Branded mobile application
- API for integration with their billing
- Technical support from your team

What you do:

- Create portal for partner
- Provide access to channels
- Maintain infrastructure
- Update EPG
- Ensure stable operation
- Bill the partner

Workflow for creating partner portal:

1. Partner registers in your system
2. You create portal with their domain
3. Partner configures branding (logo, colors, name)
4. You connect channels per tariff
5. Partner gets API key for integration
6. You create branded mobile app for partner
7. Partner starts attracting subscribers

Scenario 3: Geographic Separation

Task: Provide IPTV in different countries/regions

Why separate portals needed:

- Different content due to licensing restrictions
- Different interface languages
- Different currencies and payment methods
- Local channels for each region
- Compliance with local legislation

Example:

```
International IPTV service

├── Portal "IPTV Russia" (iptv.ru)
│   ├── Content: Russian + international channels
│   ├── Language: Russian
│   ├── Currency: rubles
│   └── 50,000 subscribers
│
├── Portal "IPTV Europe" (iptv.eu)
│   ├── Content: European channels
│   ├── Languages: English, German, French
│   ├── Currency: euros
│   └── 30,000 subscribers
│
└── Portal "IPTV USA" (iptv.com)
    ├── Content: American channels
    ├── Language: English
    ├── Currency: dollars
    └── 20,000 subscribers
```

Scenario 4: Testing Environment

Task: Safely test new features

Solution:

- Create separate portal `test.myiptv.com`
- Use for internal testing
- Test new channels, packages, features
- Don't affect production portals

Benefits:

- Complete isolation from production data
- Ability to experiment
- Integration testing
- Training new employees

2.1.9 Branded Mobile Applications

Branded App Concept

Each portal can have separate mobile application with unique brand.

What branded application includes:

- Portal logo as app icon
- Portal name in App Store / Google Play
- Portal color scheme in interface
- Unique Bundle ID / Package Name
- Connection to portal API via API key

Platforms:

- **iOS** — Swift/SwiftUI app for iPhone/iPad
- **Android** — Kotlin/Java app
- **Android TV** — Smart TV version
- **Apple TV** — Apple TV version

App Creation Process

Typical workflow:

1. **You provide portal parameters:**
2. Domain name (domain)
3. Logo (logo)
4. Title (title)
5. Color scheme
6. API endpoint
7. **Developer creates application:**
8. Brands interface according to design
9. Integrates with Catena API
10. Configures SMS authentication
11. Implements player for viewing
12. **Store publication:**
13. Registration in Apple Developer / Google Play Console
14. Preparing screenshots and description
15. Passing moderation
16. Publishing application
17. **Subscribers download:**
18. Find your app in store
19. Install on device
20. Login via SMS
21. Watch channels

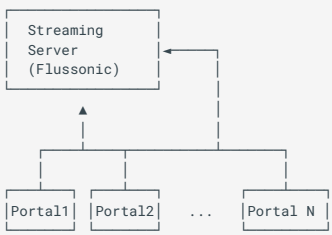
Important:

- iOS requires Apple Developer account ($99/year)
- Android requires Google Play Console ($25 one-time)
- App must comply with store rules
- App updates go through moderation

2.1.10 Shared Infrastructure for Portals

Shared Streaming Servers

All portals use same servers for content delivery:



Benefits:

- One source for channel → N portals
- Traffic and CPU savings
- Centralized stream management
- Single monitoring point

Access control:

- Streaming server checks subscriber's `playback_token`
- Token contains portal_id information
- Subscriber can only watch their portal's channels
- Technically possible to provide one channel to multiple portals

Shared Channels for Multiple Portals

Scenario: One channel source for different brands

Example:

```
Source: "Discovery Channel" (rtmp://source.tv/discovery)
                    |
      ┌─────────────┼─────────────┐
      |             |             |
Portal A      Portal B       Portal C
Channel:      Channel:       Channel:
"Discovery"   "Discovery HD" "Discov"
(in "Science" (in premium    (in basic
 package)      package)       package)
```

How it works:

1. Channel added to each portal separately
2. Each portal has its own `channelId`
3. But source URL is same
4. Streaming server caches stream
5. All portals receive stream from cache

Benefits:

- One source → multiple uses
- Licensing cost savings (depends on contract)
- Centralized EPG update
- Single quality monitoring point

2.1.11 Best Practices

Portal Naming

Internal name (name):

- Use understandable technical names
- Examples: `company-premium`, `partner-acme`, `test-portal`
- Avoid spaces and special characters
- Keep consistency: `brand-segment` or `client_name`

Public name (title):

- Use attractive marketing name
- Examples: "Premium TV", "Family Television", "Sport TV+"
- Consider target audience
- Check name availability (trademark)

Content Organization

Channel distribution strategies:

1. **Full duplication** — all portals have same content
2. Easier to manage
3. Suitable for white-label without segmentation
4. **Segmented content** — different content for different portals
5. Premium portal: exclusive channels
6. Basic portal: standard set
7. Thematic portal: sports/movies/news only
8. **Common base + unique content**
9. Basic channels available everywhere
10. Premium channels only in expensive portals
11. Local channels in regional portals

Security

Protecting API keys:

```
# BAD - key in code
api_key = "secret_key_123456"

# GOOD - key in environment variable
api_key = os.getenv('CATENA_API_KEY')

# BETTER - key in secret manager
api_key = secrets_manager.get('catena_api_key')
```

Manager access permissions:

- Grant minimum necessary permissions
- Content admin doesn't need subscriber access
- Support doesn't need API key access
- Regularly review manager list
- Remove access for terminated employees

Monitoring

What to track for each portal:

- Number of active subscribers
- Number of concurrent sessions
- Popular channels
- Login errors (failed SMS, invalid tokens)
- API requests count and latency
- Storage usage per portal

Tools:

- Grafana dashboards with portal breakdown
- Prometheus metrics with `portal_id` label
- Alerts on anomalies (sudden subscriber drop)
- Regular reports for portal owners

2.1.12 Troubleshooting

Subscribers Cannot Login

Possible causes:

- Incorrect domain specified in application
- API key expired after regeneration
- Portal temporarily unavailable
- SMS gateway not configured for portal

Solution:

1. Check domain in portal settings
2. Ensure API key is current
3. Check service status (API, SMS gateway)
4. Review authorization error logs
5. Test login from another app/browser

Channels Won't Play

Possible causes:

- Streaming server issues
- Channel not added to portal
- Subscriber not subscribed to package with channel
- Network issues on subscriber's side

Solution:

1. Check channel works on another portal
2. Ensure channel exists in this portal
3. Check subscriber's subscriptions
4. Review streaming server logs
5. Check subscriber's playback_token

API Returns 401 Unauthorized

Possible causes:

- Invalid API key
- API key was regenerated
- Key passed in incorrect format
- Key from different portal

Solution:

1. Check API key currency: `GET /portal`
2. Ensure key in header: `X-Auth-Token: your-key`
3. Verify using correct portal's key
4. Regenerate key if needed

Two Portals See Each Other's Subscribers

Problem: Data isolation between portals violated

This should not happen by system design. If it does:

1. Immediately contact technical support
2. Verify using correct API key
3. Check not mixing portals in code
4. Review API request logs

Causes (rare):

- System bug (requires fix)
- Incorrect integration (one key used for different portals)
- Client-side caching

2.1.13 See Also

- **Manager Management** — creating users to manage portals
- **Subscriber Management** — subscribers tied to specific portal
- **Package Management** — packages created within portal
- **Channel Management** — channels added to portals
- **Subscription Management** — portal free packages

2.2 Portal Manager Management

Managers are users who have access to the portal control panel in Catena. The manager system allows providing different access levels to various employees for managing content, subscribers, and portal settings.

2.2.1 What is a Manager

A manager in Catena is a user account with permissions to manage a portal. Unlike subscribers (who watch channels), managers administrate the system.

Key features:

- **Email and password authentication** — login to control panel
- **Permission system** — flexible permission configuration for each manager
- **Multiple portals** — one email can manage multiple portals
- **Roles and authorities** — from viewing statistics to full administration
- **Portal isolation** — manager sees only their portal's data

Typical team structure:

```
Portal "My IPTV Service"
├── Portal Owner (ownerId)
│   └── Assigned outside system, full access
├── Chief Administrator (isAdmin: true)
│   └── Infrastructure management
├── Content Manager (isContentAdmin: true)
│   └── Channel, package, EPG management
├── Subscriber Manager (isSubscriberAdmin: true)
│   └── Subscriber and subscription work
└── Support Operator (read-only)
    └── View data, no changes
```

2.2.2 Portal Owner vs Managers

Portal Owner

Important: Portal owner is a special role managed **outside this Management API**.

Owner characteristics:

- Set when creating portal at infrastructure level
- **Cannot be changed** via portal's Management API
- Has full and unlimited portal access
- Access rights **don't apply** to them (isAdmin, isContentAdmin, etc.)
- Can create, modify, and delete other managers
- Can assign any permissions to other managers

ownerId field in portal:

```
{
  "portalId": "pKl9SW3AAAE.",
  "ownerId": "mKl9SW3AAAE.",  // Owner ID
  "name": "my-portal"
}
```

Changing owner:

- Performed by Catena system administrator
- Not available via regular Management API
- Requires technical support contact
- Used when transferring portal to another person

Regular Managers

Managers created via API:

- Created by portal owner or other administrators
- Have limited permissions per settings
- Can be modified or deleted by owner
- Subject to permission system

Key difference:

Characteristic	Owner	Regular Manager
Creation	Outside Management API	Via Management API
Permission changes	Not applicable	Configured by owner
Deletion	Only by system admin	By portal owner
Access	Always full	According to permissions

2.2.3 Permission System

Access Levels

isAdmin — infrastructure administrator

- Technical portal settings management
- Creating and deleting other managers
- Changing critical parameters
- Server settings access
- **Doesn't automatically grant content or subscriber access**

isContentAdmin — content administrator

- Channel management (create, edit, delete)
- Channel package management
- EPG source management
- Channel-package link configuration
- **No subscriber access**

isSubscriberAdmin — subscriber administrator

- Subscriber management (create, edit, delete)
- Package subscription management
- Viewing playback sessions
- Viewing operations log
- **Cannot modify channels and packages**

Permission combination:

Manager can have multiple permissions simultaneously:

```
{
  "isAdmin": true,
  "isContentAdmin": true,
  "isSubscriberAdmin": true
}
```

This grants full access to all portal functions (except owner change).

2.2.4 Creating a Manager

Via Management API

Create new manager:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/managers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "content@company.com",
    "name": "Content Manager",
    "password": "SecurePassword123!",
    "isAdmin": false,
    "isContentAdmin": true,
    "isSubscriberAdmin": false
  }'
```

Response:

```
{
  "managerId": "mKl9SW3AAAB.",
  "portalId": "pKl9SW3AAAE.",
  "email": "content@company.com",
  "name": "Content Manager",
  "isAdmin": false,
  "isContentAdmin": true,
  "isSubscriberAdmin": false,
  "createdAt": "2024-10-16T15:00:00Z",
  "updatedAt": "2024-10-16T15:00:00Z"
}
```

Note: `password` field is not returned in response.

2.2.5 Login

Authentication Process

Managers login via email and password:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@company.com",
    "password": "password123"
  }'
```

Response:

```
{
  "portals": [
    {
      "portalId": "pKl9SW3AAAE.",
      "portalName": "My IPTV Service",
      "sessionId": "sessionKl9SW3AAAE."
    },
    {
      "portalId": "pKl9SW3AAAB.",
      "portalName": "Premium TV",
      "sessionId": "sessionKl9SW3AAAB."
    }
  ]
}
```

Multiple portals:

- If email used in multiple portals, list of all available is returned
- Manager selects portal to work with
- Each portal has its own `sessionId` for further work

2.2.6 Best Practices

Password Security

Password requirements:

- Minimum 8 characters
- Upper and lowercase letters
- Numbers and special characters
- Don't use common passwords
- Change every 90 days

Permission Management

Principle of least privilege:

```
DON'T grant more permissions than needed for work
```

Offboarding

Checklist when employee leaves:

1. ☑ Immediately delete manager account
2. ☑ Check if they were portal owner
3. ☑ Change portal API keys (if had access)
4. ☑ Check operations log for suspicious actions
5. ☑ Notify team about access changes

2.2.7 Troubleshooting

Cannot Login

Possible causes:

1. Incorrect email or password
2. Account deleted
3. Account blocked
4. Email specified for different portal

Cannot Change Portal Owner

This is correct behavior:

- Portal owner managed **outside Management API**
- Owner change is critical operation
- Requires Catena system administrator contact
- Cannot be performed independently

Owner change procedure:

1. Contact Catena technical support
2. Provide:
3. Portal ID
4. Current owner ID
5. New owner ID
6. Justification
7. Administrator performs change at system level
8. New owner receives full access

2.2.8 See Also

- **Portal Management** — portal owner and their role
- **Operations Log** — manager action audit
- **Subscriber Management** — what managers with isSubscriberAdmin can do
- **Channel Management** — what managers with isContentAdmin can do

3. Content Management

3.1 TV Channel Management

TV channels are the basic content unit in the Catena system. Each channel represents a separate video content stream that is delivered to subscribers through client applications.

3.1.1 What is a Channel in Catena

A channel in Catena is an entity that combines:

- **Technical parameters** — unique identifier and name for the streaming server
- **Visual presentation** — display title and logo for users
- **Program guide** — link to EPG (Electronic Program Guide) source
- **Pricing** — inclusion in channel packages for selling to subscribers

It's important to understand that **Catena does not handle direct video stream delivery** — that's the streaming server's job (e.g., Flussonic Media Server). Catena manages channel metadata and access rights.

3.1.2 Main Channel Parameters

Technical Parameters

Channel ID

- Automatically generated when creating a channel
- Format: base64-encoded Snowflake ID with `+/=` replaced by `-_.`
- Example: `aKl9SW3AAAE.`
- Used for programmatic access via API
- Not editable after creation

Streaming Name (Name)

- Unique technical channel name within the portal
- Used by the streaming server to identify the stream
- Requirements:
- Only Latin letters, digits, hyphen and underscore: `[a-zA-Z0-9_-]`
- Length from 2 to 20 characters
- Must be unique within your portal
- Examples: `sport1`, `news-hd`, `first_channel`

Display Parameters

User Title (Title)

- Localized channel name that viewers see
- Can contain any characters, including Cyrillic
- Examples: `First Channel`, `Sport HD`, `News 24`

Logo

- Channel image for display in client applications
- Format: PNG, transparent background recommended
- Uploaded as binary data (base64)
- Available through a separate endpoint: `GET /channels/{channelId}/logo`
- Optimal size: 300x300 pixels

EPG Integration

EPG Source Name

- Name of the electronic program guide source
- References a previously created EPG Source in the system
- One EPG source can be used for multiple channels

EPG Channel Name

- Channel identifier within the EPG source
- Used to match your channel with the program guide from the EPG file
- Must exactly match the channel name in the XML EPG

Link Example: If in your EPG file the channel is called `perviy-kanal`, then in the EPG Channel Name field you need to specify exactly `perviy-kanal`, even if in Catena your channel is called `first_channel`.

3.1.3 Creating a Channel

Via Web Interface

1. **Open the "Channels" section** in the Catena control panel
2. **Click the "Create Channel" button**
3. **Fill in required fields:**
4. **Name** — technical name for the streaming server (in Latin)
5. **Title** — name to display to users
6. **Fill in additional fields (optional):**
7. **Logo** — upload channel image (PNG)
8. **EPG Source Name** — select program guide source
9. **EPG Channel Name** — specify channel name in the EPG source
10. **Save the channel**

After creation, the channel will receive a unique ID and will be available for adding to packages.

Via Management API

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/channels \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "sport1",
    "title": "Sport HD",
    "epgSourceName": "main-epg",
    "epgChannelName": "sport-channel-1"
  }'
```

Response:

```
{
  "channelId": "aKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE.",
  "name": "sport1",
  "title": "Sport HD",
  "epgSourceName": "main-epg",
  "epgChannelName": "sport-channel-1",
  "packages": []
}
```

3.1.4 Viewing Channel List

Via Web Interface

The "Channels" section displays a table with all portal channels:

- **Logo** — channel logo thumbnail
- **Title** — display name (Title)
- **Technical Name** — streaming name (Name)
- **Packages** — list of packages that include the channel
- **EPG** — information about the connected program guide source
- **Actions** — edit and delete buttons

Via Management API

Get list of all channels:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/channels \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "channels": [
    {
      "channelId": "aKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "name": "sport1",
      "title": "Sport HD",
      "packages": ["basic", "premium"],
      "epgSourceName": "main-epg",
      "epgChannelName": "sport-channel-1"
    }
  ],
  "next": "cursor-for-next-page"
}
```

Pagination: To get the next page, use the `cursor` parameter:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/channels?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

3.1.5 Editing a Channel

Via Web Interface

1. **Open the channel list**
2. **Find the needed channel** and click the "Edit" button
3. **Change parameters:**
4. Title — can change the display name
5. Logo — upload new image
6. EPG Source Name / EPG Channel Name — change link to program guide

7. **Save changes**

Important: The `name` field (technical name) cannot be changed after channel creation. If you need to change the technical name, create a new channel and delete the old one.

Via Management API

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/channels/aKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "sport1",
    "title": "Sport Full HD",
    "epgSourceName": "new-epg-source",
    "epgChannelName": "sport-hd-channel"
  }'
```

3.1.6 Uploading and Getting Logo

Uploading Logo via API

When creating or updating a channel, the logo is passed in the `logo` field as a base64 string:

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/channels/aKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "sport1",
    "title": "Sport HD",
    "logo": "..."
  }'
```

Getting Logo

The logo is available through a separate endpoint:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/channels/aKl9SW3AAAE./logo \
  -H "X-Auth-Token: your-api-key" \
  --output channel-logo.png
```

This URL can be used in client applications to display the logo.

3.1.7 Deleting a Channel

Via Web Interface

1. **Open the channel list**
2. **Find the channel** to delete
3. **Click the "Delete" button**
4. **Confirm deletion**

Warning: When deleting a channel, it will be automatically removed from all packages it was included in.

Via Management API

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/channels/aKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

3.1.8 Channel-Package Relationship

A channel by itself is not accessible to subscribers. To provide access to a channel, it needs to be included in a channel package.

The `packages` field in the channel (read-only): When getting channel information, the `packages` field contains a list of package names that include the channel. This field is read-only and is automatically updated when adding/removing a channel to/from packages via the channel-package relationship management API.

Example:

```
{
  "channelId": "aKl9SW3AAAE.",
  "name": "sport1",
  "title": "Sport HD",
  "packages": ["basic", "premium", "sport-package"]
}
```

3.1.9 Channel-EPG Relationship

How EPG Integration Works

1. **Create an EPG source** in the EPG Sources section
2. **Specify the URL** of the XML file with the program guide
3. **Start synchronization** of EPG data
4. **In the channel settings, specify:**
5. `epgSourceName` — name of the created EPG source
6. `epgChannelName` — channel name as specified in the XML EPG

Channel Mapping

EPG XML structure example:

```
<tv>
  <channel id="perviy-kanal">
    <display-name>First Channel</display-name>
  </channel>
  <programme start="20241015120000" stop="20241015130000" channel="perviy-kanal">
    <title lang="en">News</title>
  </programme>
</tv>
```

In Catena specify:

- EPG Source Name: `main-epg` (name of the source you created)
- EPG Channel Name: `perviy-kanal` (value of the `id` or `display-name` attribute from XML)

After this, the program guide will be automatically available for this channel in client applications.

3.1.10 Typical Use Cases

Launching a New Channel

Task: Add a new sports channel to the service

Steps:

1. Create a channel in Catena with the name `sport-premium`
2. Upload the channel logo
3. Configure EPG link (if program guide is available)
4. Add the channel to one or more packages
5. Configure the corresponding stream on the streaming server with the name `sport-premium`

Bulk Adding Channels

Task: Add 50 channels from a new content provider

Solution via API:

1. Prepare CSV or JSON with channel data
2. Create a script for automatic channel creation via API
3. Upload logos for each channel
4. Configure EPG mapping
5. Group channels into thematic packages

Updating EPG for Channels

Task: Change EPG source for a group of channels

Steps:

1. Create a new EPG Source with current data
2. Update channels, specifying the new `epgSourceName`
3. Check the correctness of `epgChannelName` mapping
4. Start EPG update

Channel Rebranding

Task: Change channel name and logo

Steps:

1. Open channel editing
2. Update the `title` field with the new name
3. Upload a new logo
4. Save changes
5. Changes will automatically appear in client applications on the next data update

3.1.11 Best Practices

Channel Naming

- **Name (technical name):**
- Use short, clear names: `sport1`, `news`, `movies-hd`
- Avoid special characters except hyphen and underscore
- Add suffixes for HD/SD versions: `sport-hd`, `sport-sd`
- **Title (display name):**
- Use full, clear names: "Sport HD", "News 24"
- Can use any characters and emoji
- Indicate quality in the name: "4K", "HD", "SD" (if important)

Channel Organization

- **Group channels logically** by themes, using packages
- **Use a consistent style** for logos (size, background, format)
- **Keep EPG data current**
- **Document** technical channel mapping

Logo Management

- **Size:** optimal 300x300 pixels
- **Format:** PNG with transparent background
- **File size:** no more than 100 KB for fast loading
- **Consistent style:** use the same style for all logos

Streaming Server Integration

Remember that **the technical channel name in Catena must match the stream name on the streaming server**:

- Catena: `name: "sport1"`
- Flussonic: stream must be named `sport1`

This ensures proper operation of access tokens and viewing analytics.

3.1.12 Troubleshooting

Channel Not Displayed in Application

Possible causes:

- Channel is not included in any package
- Subscriber doesn't have a subscription to a package with this channel
- Corresponding broadcast is not configured on the streaming server

Solution:

1. Check that the channel is added to a package
2. Make sure the subscriber is subscribed to this package
3. Verify that the streaming server is delivering a stream with the corresponding name

EPG Not Displayed for Channel

Possible causes:

- Incorrect `epgChannelName` specified
- EPG Source not updated or contains errors
- No data for this channel in the EPG XML

Solution:

1. Open the XML EPG and find the correct channel identifier
2. Update `epgChannelName` in the channel settings
3. Start forced EPG update
4. Check EPG update logs for errors

"Name must be unique" Error

Cause: A channel with this technical name already exists in your portal

Solution:

- Use a different technical name
- Or delete the existing channel with that name (if no longer needed)

Logo Won't Upload

Possible causes:

- File is too large
- Unsupported image format
- Base64 encoding error

Solution:

- Use PNG format
- Compress the image to size < 100 KB
- Check base64 encoding correctness

3.1.13 See Also

- **Channel Package Management** — grouping channels for sale
- **EPG Management** — configuring electronic program guide
- **Subscriber Management** — providing access to channels
- **API Reference** — complete API documentation for channels

3.2 Channel Package Management

Channel packages are groups of TV channels bundled together for selling to subscribers. Packages allow you to create various pricing plans and monetize your IPTV service.

3.2.1 What is a Channel Package

A channel package in Catena is a named group of TV channels that can be assigned to subscribers. Packages allow you to:

- **Create pricing plans** — group channels by themes (sports, movies, news) or access levels (basic, premium)
- **Monetize the service** — sell access to channel packages to subscribers
- **Manage access** — provide different subscribers with access to different sets of channels
- **Simplify administration** — assign packages instead of managing access to each channel individually

Important concept: A channel by itself is not accessible to subscribers. Access to a channel is provided only through packages that the subscriber is subscribed to.

3.2.2 Main Package Parameters

Technical Parameters

Package ID

- Automatically generated when creating a package
- Format: base64-encoded Snowflake ID with `+/=` replaced by `-_.`
- Example: `aKl9SW3AAAE.`
- Used for programmatic access via API
- Not editable after creation

Package Name (Name)

- Unique technical package name within the portal
- Used in the system to identify the package
- Requirements:
- Only Latin letters, digits, hyphen and underscore: `[a-zA-Z0-9_-]`
- Length from 2 to 20 characters
- Must be unique within your portal
- Examples: `basic`, `premium`, `sport-pack`, `movies_hd`

Portal ID

- Identifier of the portal the package belongs to
- Automatically set upon creation

Display Parameters

Description

- Text description of the package for administrators
- Can contain information about package content, pricing, target audience
- Not a required field
- Examples: "Basic package of 30 channels", "Premium sports channels in HD quality"

Package Content

Channel List (Channels)

- Read-only field
- Contains names of all channels included in the package
- Automatically updated when adding/removing channels via relationship management API
- Example: `["sport1", "sport2", "news", "movies-hd"]`

3.2.3 Creating a Package

Via Web Interface

1. **Open the "Packages" section** in the Catena control panel
2. **Click the "Create Package" button**
3. **Fill in required fields:**
4. **Name** — technical package name (in Latin)
5. **Description** — package description (optional)
6. **Save the package**
7. **Add channels to the package** through the composition management section

After creation, the package will receive a unique ID and will be available for assignment to subscribers.

Via Management API

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "premium",
    "description": "Premium package with HD channels"
  }'
```

Response:

```
{
  "packageId": "pKl9SW3AAAE.",
  "portalId": "portal123",
  "name": "premium",
  "description": "Premium package with HD channels",
  "channels": []
}
```

3.2.4 Viewing Package List

Via Web Interface

The "Packages" section displays a table with all portal packages:

- **Name** — package name (Name)
- **Description** — text package description
- **Channel Count** — how many channels are in the package
- **Subscribers** — number of subscribers with this package
- **Actions** — edit and delete buttons

Via Management API

Get list of all packages:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/packages \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "packages": [
    {
      "packageId": "pKl9SW3AAAE.",
      "portalId": "portal123",
      "name": "premium",
      "description": "Premium package with HD channels",
      "channels": ["sport1", "sport2", "news-hd", "movies-4k"]
    },
    {
      "packageId": "bKl9SW3AAAE.",
      "portalId": "portal123",
      "name": "basic",
      "description": "Basic channel package",
      "channels": ["news", "general"]
    }
  ],
  "next": "cursor-for-next-page"
}
```

Pagination:

To get the next page, use the `cursor` parameter:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/packages?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

3.2.5 Getting Package Information

Via Management API

Get package by ID:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/packages/pKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "packageId": "pKl9SW3AAAE.",
  "portalId": "portal123",
  "name": "premium",
  "description": "Premium package with HD channels",
  "channels": ["sport1", "sport2", "news-hd", "movies-4k"]
}
```

3.2.6 Editing a Package

Via Web Interface

1. **Open the package list**
2. **Find the needed package** and click the "Edit" button
3. **Change parameters:**
4. Name — technical name (better not to change after creation)
5. Description — package description
6. **Save changes**

Note: Changing the channel composition of a package is done separately through channel-package relationship management.

Via Management API

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/packages/pKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "premium",
    "description": "Premium package with HD and 4K channels"
  }'
```

3.2.7 Deleting a Package

Via Web Interface

1. **Open the package list**
2. **Find the package** to delete
3. **Click the "Delete" button**
4. **Confirm deletion**

Warning: When deleting a package:

- All subscribers will lose access to channels from this package (if they don't have other packages with these channels)
- Relationships between the package and channels will be removed
- Relationships between the package and subscribers will be removed

Via Management API

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/packages/pKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

3.2.8 Managing Package Composition

Adding a Channel to Package

Via Management API:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/channels-packages \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "channelId": "cKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE.",
    "portalId": "portal123"
  }'
```

Response:

```
{
  "channelId": "cKl9SW3AAAE.",
  "packageId": "pKl9SW3AAAE.",
  "portalId": "portal123"
}
```

After adding:

- The channel will be displayed in the package's `channels` field
- The package will be displayed in the channel's `packages` field
- All subscribers with this package will receive access to the added channel

Removing a Channel from Package

Via Management API:

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/channels-packages \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "channelId": "cKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE.",
    "portalId": "portal123"
  }'
```

Important: Removing a channel from a package does not delete the channel itself from the system, only breaks the relationship between the channel and package.

Bulk Channel Management

To add multiple channels to a package, use sequential API calls:

```
# Add channel 1
curl -X POST https://your-catena-domain.com/tv-management/api/v1/channels-packages \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{"channelId": "channel1", "packageId": "premium", "portalId": "portal123"}'

# Add channel 2
curl -X POST https://your-catena-domain.com/tv-management/api/v1/channels-packages \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{"channelId": "channel2", "packageId": "premium", "portalId": "portal123"}'
```

3.2.9 Assigning Packages to Subscribers

For more details on assigning packages to subscribers, see the Subscription Management section.

Adding a Package to Subscriber

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "packageId": "pKl9SW3AAAE.",
    "subscriberId": "sKl9SW3AAAE.",
    "portalId": "portal123"
  }'
```

After assigning a package to a subscriber:

- The subscriber will receive access to all channels from this package
- The package will appear in the subscriber's package list
- Access will be provided in all client applications

Removing a Package from Subscriber

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "packageId": "pKl9SW3AAAE.",
    "subscriberId": "sKl9SW3AAAE.",
    "portalId": "portal123"
  }'
```

3.2.10 Free Packages

Free packages are packages that are automatically available to all portal subscribers without the need for explicit assignment.

How It Works

Free packages are configured at the portal level:

- Administrator adds a package to the portal's free packages list
- All subscribers automatically receive access to channels from free packages
- No explicit assignment of the package to each subscriber is required

Typical Usage:

- **Trial period** — provide a basic set of channels to all new subscribers
- **Free channels** — open channels available to everyone (news, public channels)
- **Demo content** — show service capabilities before purchasing a subscription

Adding Package to Free List

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/pKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

After adding to the free list:

- All existing and new subscribers will receive access to channels from this package
- The package will be displayed as free in the portal settings

Removing Package from Free List

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/pKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Important: Removing a package from free packages does not delete the package itself, only removes automatic access for all subscribers.

3.2.11 Typical Use Cases

Creating Basic Pricing Grid

Task: Create three subscription levels: Basic, Standard, Premium

Steps:

1. **Create three packages:**
2. `basic` — 30 basic channels
3. `standard` — 50 channels (basic + entertainment)
4. `premium` — 80 channels (all + sports and movies in HD)
5. **Fill packages with channels:**
6. Basic: news, general, music
7. Standard: basic + series, documentaries
8. Premium: standard + sports HD, movies 4K, exclusive
9. **Set up free package:**
10. Create a `trial` package with 5 open channels
11. Add it to the portal's free packages list
12. **Assign packages to subscribers** depending on their subscription

Thematic Packages

Task: Create thematic add-on packages

Package Examples:

- `sport-package` — all sports channels
- `kids-package` — children's channels
- `movies-package` — movie channels
- `news-package` — news channels

Advantages:

- Subscriber can purchase topics of interest in addition to basic subscription
- More flexible monetization
- Personalized offering

Regional Packages

Task: Provide different sets of channels for different regions

Solution:

1. Create regional packages:
2. `region-moscow` — Moscow regional channels
3. `region-spb` — St. Petersburg channels
4. `region-south` — southern Russia channels
5. Assign the appropriate regional package when registering a subscriber
6. Subscriber will receive basic package + regional

Temporary Promotions

Task: Conduct a promo campaign with extended access

Steps:

1. Create a temporary package `promo-may`
2. Add premium channels to it
3. Assign the package to all active subscribers
4. Remove the package from all subscribers at the end of the campaign

3.2.12 Best Practices

Planning Package Structure

Naming Recommendations:

- Use clear technical names: `basic`, `premium`, `sport-hd`
- Avoid using versions in the name: not `premium-v2`, but create a new package
- Use prefixes for grouping: `addon-sport`, `addon-kids`, `addon-movies`

Pricing Grid Structure:

- **Basic level** — minimum set to start using the service
- **Mid-level** — optimal price/channel count ratio
- **Premium level** — maximum set with all available channels
- **Add-on packages** — thematic add-ons to main packages

Change Management

When changing package composition:

- Inform subscribers about adding new channels
- Warn in advance about removing channels from the package
- Maintain documentation on each package composition
- Keep history of changes for analytics

When changing pricing:

- Don't change the technical package name when changing price
- Use the billing system to manage prices
- Ensure smooth transition for existing subscribers

Monitoring and Analytics

Track metrics:

- Number of subscribers on each package
- Channel popularity within packages
- Conversion from free package to paid
- Subscriber churn when changing package composition

Use data for optimization:

- Form packages based on channel popularity
- Test different channel combinations
- Adjust package composition based on analytics results

Billing Integration

Recommendations:

- Use technical package name (name) as identifier in billing system
- Synchronize package assignment/removal with payments via API
- Automate access blocking on non-payment
- Set up webhooks for subscription change notifications

3.2.13 Troubleshooting

Subscriber Doesn't See Channels from Package

Possible causes:

- Package is not assigned to subscriber
- Package has no channels (empty package)
- Data synchronization issues in client application

Solution:

1. Check subscriber's package list via API
2. Make sure the package contains channels
3. Verify that channels are properly configured on streaming server
4. Restart client application to update data

Channels Are Duplicated in Subscriber's List

Cause: Channel is included in multiple packages assigned to the subscriber

This is normal behavior:

- Subscriber can have multiple packages
- Channel can be included in multiple packages simultaneously
- Client application should deduplicate channel list

Solution: Make sure the client application correctly handles duplicates.

Error Adding Channel to Package

Possible causes:

- Incorrect `channelId` or `packageId`
- Channel is already in this package
- Channel or package belong to a different portal

Solution:

1. Check channel and package existence
2. Make sure `portalId` matches for channel, package and request
3. Check if this channel is already added to the package

Package Won't Delete

Possible causes:

- Package is assigned to subscribers
- Package is in the portal's free packages list
- Insufficient permissions for deletion

Solution:

1. First remove the package from all subscribers
2. Remove package from free list (if it's there)
3. Then delete the package itself

Free Package Not Working

Possible causes:

- Package is not added to portal's free packages list
- Subscriber is explicitly blocked or not activated
- Package is empty (contains no channels)

Solution:

1. Check the free packages list in portal settings
2. Make sure the subscriber is active
3. Verify the presence of channels in the package

3.2.14 See Also

- **Channel Management** — creating and configuring TV channels
- **Subscriber Management** — registration and subscriber management
- **Subscription Management** — assigning packages to subscribers
- **Portal Settings** — managing free packages
- **API Reference** — complete package API documentation

3.3 EPG Source Management

EPG sources (Electronic Program Guide) provide program information for each channel: program names, start and end times, descriptions, genres, and age ratings.

3.3.1 What is an EPG Source

An EPG source in Catena is a link to an external XML file with TV program schedules. The system periodically downloads this file and synchronizes program data for display in client applications.

Key capabilities:

- **Automatic synchronization** — regular download and update of program guide
- **Multiple source support** — different EPG sources for different channel groups
- **Download monitoring** — tracking status and results of EPG updates
- **Program viewing** — retrieving schedules via API for integration

Typical workflow:

1. Create an EPG source with XML file URL specified
2. Configure automatic update period
3. Link channels to EPG source (in channel settings)
4. System automatically downloads and updates program guide
5. Subscribers see current program guide in applications

3.3.2 Main EPG Source Parameters

Technical Parameters

Source ID (EPG Source ID)

- Automatically generated when creating a source
- Format: base64-encoded Snowflake ID with `+/=` replaced by `-_.`
- Example: `aKl9SW3AAAE.`
- Used for programmatic access via API
- Not editable after creation

Source Name (Name)

- Unique technical name of the EPG source
- Used for identification in the system and in channel settings
- Examples: `main-epg`, `sports-epg`, `xmltv-provider`

Portal ID

- Identifier of the portal the source belongs to
- Automatically set upon creation

Download Parameters

Source URL (URL)

- Full URL to XML file with program guide
- Supported protocols: HTTP, HTTPS
- Example: `https://epg.example.com/epg.xml`
- **Required field**

Update Period (Period)

- Interval in days for automatic EPG update
- Example: `7` — update every 7 days
- Minimum value: 1 day
- If not specified, default value is used
- **The system automatically downloads EPG** according to the specified period without manual intervention

Last Download Result

Download Information (Last Fetch Result)

The `last_fetch_result` field contains information about the last EPG update:

- **fetched_at** — time of last download (ISO 8601)
- **job_id** — download job identifier
- **status** — download status:
- `success` — successfully downloaded
- `error` — download error
- `timeout` — timeout exceeded
- **message** — text message about result
- **channels** — number of updated channels
- **foundPrograms** — number of programs found in XML
- **importedPrograms** — number of imported programs
- **deletedPrograms** — number of deleted old programs
- **fetchDuration** — XML download time in seconds
- **importDuration** — data import time in seconds

3.3.3 Creating an EPG Source

Via Web Interface

1. **Open the "EPG Sources" section** in the Catena control panel
2. **Click the "Create EPG Source" button**
3. **Fill in required fields:**
4. **Name** — technical source name (e.g., `main-epg`)
5. **URL** — full URL to EPG XML file
6. **Fill in optional fields:**
7. **Period** — update period in days (e.g., `7`)
8. **Save the source**
9. **Start initial download** via "Update Now" button

After creation, the source will receive a unique ID and will be available for linking to channels.

Via Management API

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/epg-sources \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "main-epg",
    "url": "https://epg.example.com/xmltv.xml",
    "period": 7
  }'
```

Response:

```
{
  "epgSourceId": "eKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE.",
  "name": "main-epg",
  "url": "https://epg.example.com/xmltv.xml",
  "period": 7,
  "last_fetch_result": null
}
```

3.3.4 Viewing EPG Source List

Via Web Interface

The "EPG Sources" section displays a table with all sources:

- **Name** — source name (Name)
- **URL** — XML file address
- **Period** — update interval in days
- **Last Update** — date and time of last download
- **Status** — result of last download (success/error/timeout)
- **Programs** — number of imported programs
- **Actions** — update, edit, and delete buttons

Via Management API

Get list of all EPG sources:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/epg-sources \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "epgSources": [
    {
      "epgSourceId": "eKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "name": "main-epg",
      "url": "https://epg.example.com/xmltv.xml",
      "period": 7,
      "last_fetch_result": {
        "epgSourceId": "eKl9SW3AAAE.",
        "fetched_at": "2024-10-16T10:30:00Z",
        "job_id": "job123",
        "status": "success",
        "message": "EPG source fetched successfully",
        "channels": 25,
        "foundPrograms": 5000,
        "importedPrograms": 4950,
        "deletedPrograms": 2100,
        "fetchDuration": 5,
        "importDuration": 12
      }
    }
  ]
}
```

3.3.5 Getting Source Information

Via Management API

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response: Similar to EPG source object from the list.

3.3.6 Editing an EPG Source

Via Web Interface

1. **Open the EPG sources list**
2. **Find the needed source** and click the "Edit" button
3. **Change parameters:**
4. Name — technical name
5. URL — XML file address
6. Period — update period
7. **Save changes**

Note: Changing URL or period does not trigger automatic update — use "Update Now" button for immediate download.

Via Management API

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "main-epg",
    "url": "https://epg.example.com/updated-xmltv.xml",
    "period": 3
  }'
```

3.3.7 Forced EPG Update

You can trigger an unscheduled EPG update at any time.

Via Web Interface

1. **Open the EPG sources list**
2. **Find the needed source**
3. **Click the "Update Now" button**
4. **Wait for completion** — process may take from several seconds to minutes

Via Management API

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./update \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "jobId": "job456"
}
```

Important: Update is performed asynchronously. Use `jobId` to track progress or check the `last_fetch_result` field after some time.

3.3.8 Deleting an EPG Source

Via Web Interface

1. **Open the EPG sources list**
2. **Find the source** to delete
3. **Click the "Delete" button**
4. **Confirm deletion**

Warning: When deleting an EPG source:

- All programs from this source will be deleted
- Channels linked to this source will lose EPG connection
- Subscribers will stop seeing program guide for these channels

Via Management API

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

3.3.9 Viewing Programs from EPG Source

You can get a list of programs for analysis or debugging.

Getting Programs via API

Get all programs from source:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./programs \
  -H "X-Auth-Token: your-api-key"
```

Filter by date:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./programs?date=2024-10-16" \
  -H "X-Auth-Token: your-api-key"
```

Filter by channel:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./programs?epgChannelName=channel1" \
  -H "X-Auth-Token: your-api-key"
```

Combined filtering:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./programs?date=2024-10-16&epgChannelName=channel1" \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "programs": [
    {
      "programId": "prKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "epgSourceId": "eKl9SW3AAAE.",
      "epgChannelName": "channel1",
      "date": "2024-10-16",
      "start": "2024-10-16T12:00:00Z",
      "end": "2024-10-16T13:00:00Z",
      "title": "News",
      "language": "en",
      "description": "Main news of the day",
      "genre": "News",
      "rating": "0+"
    }
  ],
  "next": "cursor-for-next-page"
}
```

Pagination:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-sources/eKl9SW3AAAE./programs?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

3.3.10 Viewing EPG Update History

Catena automatically saves the history of all EPG updates, allowing you to track download success, analyze issues, and monitor source performance.

Automatic EPG Updates

How automatic updates work:

- The system automatically triggers EPG updates according to each source's `period` parameter
- Updates are performed in the background without system downtime
- Each update is recorded in history with complete result information
- Upon successful update, new programs become immediately available to subscribers

Benefits of automatic updates:

- No manual intervention required to keep EPG current
- Program guide always stays up-to-date for subscribers
- Ability to track all update attempts and identify issues

Viewing History via Web Interface

The EPG source page displays:

- **Recent update history** — table with all download attempts
- **Date and time** of each update
- **Status** — success/error/timeout
- **Statistics** — number of found, imported, and deleted programs
- **Duration** — download and import time
- **Error messages** (if any)

Getting History via API

Get history of all EPG updates:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/epg-fetches \
  -H "X-Auth-Token: your-api-key"
```

Filter by specific source:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-fetches?epgSourceId=eKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Pagination:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-fetches?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "epgFetches": [
    {
      "epgFetchId": "fKl9SW3AAAE.",
      "epgSourceId": "eKl9SW3AAAE.",
      "created_at": "2024-10-16T10:25:00Z",
```

```
      "fetched_at": "2024-10-16T10:30:00Z",
      "job_id": "job123",
      "status": "success",
      "message": "EPG source fetched successfully",
      "channels": 25,
      "foundPrograms": 5000,
      "importedPrograms": 4950,
      "deletedPrograms": 2100,
      "fetchDuration": 5,
      "importDuration": 12
    },
    {
      "epgFetchId": "fKl9SW3AAAA.",
      "epgSourceId": "eKl9SW3AAAE.",
      "created_at": "2024-10-09T10:25:00Z",
      "fetched_at": "2024-10-09T10:29:00Z",
      "job_id": "job122",
      "status": "success",
      "message": "EPG source fetched successfully",
      "channels": 25,
      "foundPrograms": 4800,
      "importedPrograms": 4750,
      "deletedPrograms": 1950,
      "fetchDuration": 4,
      "importDuration": 10
    }
  ],
  "next": "cursor-for-next-page"
}
```


Update History Fields

Main history record fields:

- **epgFetchId** — unique identifier of update attempt
- **epgSourceId** — EPG source identifier
- **created_at** — time when update task was created (ISO 8601)
- **fetched_at** — time when download actually completed (ISO 8601)
- **job_id** — background job identifier
- **status** — update status:
- `success` — successfully downloaded and imported
- `error` — an error occurred
- `timeout` — timeout exceeded
- **message** — text description of result
- **channels** — number of updated channels
- **foundPrograms** — total programs found in XML file
- **importedPrograms** — programs successfully imported to database
- **deletedPrograms** — outdated programs deleted
- **fetchDuration** — XML file download time (seconds)
- **importDuration** — data processing and import time (seconds)

Analyzing Update History

Using history for monitoring:

1. **Tracking stability** — verify that updates occur regularly
2. **Identifying issues** — find records with error or timeout status
3. **Performance analysis** — track download and import times
4. **Data quality control** — compare program counts between updates

Usage examples:

```
# Check recent updates with errors
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-fetches" \
```

```
  -H "X-Auth-Token: your-api-key" | jq '.epgFetches[] | select(.status != "success")'

# Get average import time for a source
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-fetches?epgSourceId=eKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key" | jq '[.epgFetches[].importDuration] | add / length'
```

3.3.11 Linking Channels to EPG Source

An EPG source by itself does not provide program guide to channels. You need to explicitly specify in each channel's settings:

- **EPG Source Name** — name of EPG source
- **EPG Channel Name** — channel name in XML EPG

For more details see Channel EPG Integration section.

Example channel configuration:

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/channels/cKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "sport1",
    "title": "Sport HD",
    "epgSourceName": "main-epg",
    "epgChannelName": "sport-channel-1"
  }'
```

After this, the program guide from `main-epg` source for channel `sport-channel-1` will be available for channel `sport1`.

3.3.12 EPG XML Format

XMLTV Structure

Catena supports the standard XMLTV format. Basic structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<tv>
  <!-- Channel descriptions -->
  <channel id="channel1">
    <display-name>First Channel</display-name>
  </channel>

  <!-- Programs -->
  <programme start="20241016120000" stop="20241016130000" channel="channel1">
    <title lang="en">News</title>
    <desc lang="en">Main news of the day</desc>
    <category lang="en">News</category>
    <rating system="age">
      <value>0+</value>
    </rating>
  </programme>
</tv>
```

Supported Fields

Required program fields:

- `start` — start time (format: YYYYMMDDHHmmss)
- `stop` — end time
- `channel` — channel identifier (used for matching)
- `title` — program name

Optional fields:

- `desc` — program description
- `category` — genre (News, Sports, Movie, etc.)
- `rating` — age rating (0+, 6+, 12+, 16+, 18+)
- `lang` — program language

3.3.13 Typical Use Cases

Connecting Standard XMLTV Provider

Task: Connect EPG from third-party provider

Steps:

1. Get XMLTV URL from provider (e.g., `https://provider.com/epg.xml`)
2. Create EPG source in Catena with this URL
3. Set update period to 1 day
4. Start initial download
5. Check download result in `last_fetch_result`
6. Link channels to source, specifying correct `epgChannelName`
7. Verify program guide display in client applications

Using Multiple EPG Sources

Task: Different channel groups get EPG from different sources

Example:

- `epg-russia` — Russian channels
- `epg-europe` — European channels
- `epg-sports` — sports channels from specialized provider

Advantages:

- Independent update of different channel groups
- Ability to use specialized providers for specific topics
- Problem isolation — error in one source doesn't affect others

Monitoring EPG Updates

Task: Track EPG download success

Solution via API:

```
# Get update history for all sources
curl -X GET https://your-catena-domain.com/tv-management/api/v1/epg-fetches \
  -H "X-Auth-Token: your-api-key"

# Or get history for a specific source
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/epg-fetches?epgSourceId=eKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"

# Check status for each record
# If status != "success" — there's a problem
# If last update is old — EPG is outdated
```

Setting up alerts:

- Create EPG update history check script
- Run on schedule (e.g., every hour)
- Send notifications on errors or missing updates
- Use `/epg-fetches` endpoint to get complete history

3.3.14 Best Practices

Choosing EPG Provider

Selection criteria:

- **Data completeness** — presence of descriptions, genres, ratings
- **Timeliness** — how often EPG is updated
- **Coverage** — support for your needed channels
- **Reliability** — service uptime, response speed
- **Format** — compliance with XMLTV standard
- **Cost** — free vs paid sources

Configuring Update Period

Recommendations:

- **1 day** — for sources with daily updates
- **7 days** — for sources with program guide week ahead
- **Less than a day** — not recommended, creates unnecessary load

Consider:

- How often provider updates EPG
- XML file size (large files — update less often)
- Load on your server

Error Handling

Monitoring:

- Regularly check `last_fetch_result.status`
- Set up alerts when status = "error" or "timeout"
- Track `fetched_at` — warn if EPG hasn't updated for >2 days

On errors:

1. Check URL availability (open in browser)
2. Check XML format — structure validity
3. Check file size — possibly too large
4. Check server network settings (firewall, proxy)

Performance Optimization

Recommendations:

- Don't download EPG more often than necessary
- Use CDN for EPG XML distribution if it's your file
- Ensure XML file is compressed (gzip)
- Split large EPG into multiple sources by topics

3.3.15 Troubleshooting

EPG Not Loading

Possible causes:

- URL unavailable or returns error
- XML file has incorrect format
- Network issues on Catena server side
- Timeout — file too large

Solution:

1. Check `last_fetch_result.message` for error details
2. Open URL in browser — verify availability
3. Validate XML through online validator
4. Check file size — try to reduce
5. Check Catena server logs

Programs Not Displayed in Application

Possible causes:

- Channel not linked to EPG source
- Incorrect `epgChannelName` in channel settings
- EPG not updated or download had error
- Programs in EPG are outdated (past dates)

Solution:

1. Check channel settings — `epgSourceName` and `epgChannelName`
2. Compare `epgChannelName` with channel identifier in XML
3. Check `last_fetch_result` — was download successful
4. Check program presence via API `/epg-sources/{id}/programs`
5. Ensure EPG has programs for current/future date

Incomplete Program Data

Cause: EPG XML doesn't contain all fields (descriptions, genres, ratings)

Solution:

- Contact EPG provider to improve data
- Use another provider with more complete data
- Accept as is — basic information (name, time) will still be available

Timezone Mismatch

Problem: Program times display incorrectly

Solution:

1. Ensure EPG XML time is in UTC or with timezone specified
2. Check timezone settings on Catena server
3. Client applications should convert UTC to user's local time

Duplicate Programs

Cause: EPG updates but old programs not deleted

Solution:

- This is normal behavior during updates
- The `deletedPrograms` field in `last_fetch_result` shows how many deleted
- If duplicates remain, may be issue in program identification in EPG XML

3.3.16 See Also

- **Channel Management** — linking channels to EPG sources
- **Channel EPG Integration** — configuring connection
- **API Reference** — complete EPG sources API documentation

4. Subscriber Management

4.1 Subscriber Management

Subscribers are the end users of your IPTV service who get access to watch TV channels. The Catena system provides flexible subscriber management, their channel package subscriptions, and access control.

4.1.1 What is a Subscriber

A subscriber in Catena is a user account that has access to watch channels through connected packages.

Key capabilities:

- **SMS authentication** — primary login method for subscribers via code sent to phone
- **Subscription management** — connecting and disconnecting channel packages
- **Access control** — automatic channel access management based on subscriptions
- **Playback tokens** — unique tokens for authorization during viewing
- **Activity monitoring** — tracking viewing sessions and subscriber activity

Typical workflow:

1. Create subscriber account with phone number
2. Connect channel packages to subscriber
3. Subscriber receives SMS with login code for the app
4. After login, subscriber gets access to all channels from their packages
5. System automatically manages access rights based on active subscriptions

4.1.2 Main Subscriber Parameters

Technical Parameters

Subscriber ID

- Automatically generated when creating a subscriber
- Format: base64-encoded Snowflake ID with `+/=` replaced by `-_.`
- Example: `aKl9SW3AAAE.`
- Used for programmatic access via API
- Not editable after creation

Portal ID

- Identifier of the portal the subscriber belongs to
- Automatically set upon creation
- Subscriber can only access channels and packages from their portal

Personal Information

Subscriber Name

- Display name or user identifier
- Can be full name, nickname, or identifier from external system
- Used for display in management interface
- Examples: "John Doe", "user123", "Apartment 42"

Phone Number (Phone)

- Subscriber's phone number **without country code**
- Used for **SMS authentication** — the primary login method
- Digits only, no spaces or special characters
- Validation pattern: `^[0-9]*$`
- Examples: `2345678901`, `9161234567`

Country Code (Phone Country Code)

- Phone country code **without plus sign**
- Used together with `phone` field to form complete number
- Digits only
- Validation pattern: `^[0-9]*$`
- Examples: `1` (USA), `44` (UK), `7` (Russia)

Complete phone number is formed as: `+{phoneCountryCode}{phone}`

Example: `phoneCountryCode: "1"` + `phone: "2345678901"` = `+12345678901`

Access Parameters

Playback Token

- Unique token for authorization during video playback
- Generated automatically by the system
- Used by streaming server to verify access rights
- Transmitted to app after successful authentication
- Can be regenerated if needed

Package List (Packages)

- Array of channel package identifiers the subscriber is connected to
- Read-only field — displays current active subscriptions
- Updated automatically when packages are connected/disconnected
- Determines which channels the subscriber has access to
- Example: `["pKl9SW3AAAE.", "bKl9SW3AAAE."]`

4.1.3 Subscriber Authentication

SMS Login (Primary Method)

Catena uses SMS authentication as the primary login method for subscribers. This provides:

- **Ease of use** — no need to remember passwords
- **Security** — one-time codes tied to phone
- **Convenience** — quick registration and login
- **Fraud protection** — phone as authentication factor

SMS login process:

1. **Subscriber enters phone number** in the app
2. **System sends SMS** with one-time code to the specified number
3. **Subscriber enters code** from SMS in the app
4. **System verifies code** and issues access token
5. **Subscriber gets access** to watch channels from their packages

Important: Phone number is the unique identifier of the subscriber in the system. Ensure numbers are entered correctly when creating accounts.

Automatic Subscriber Creation

The system can automatically create subscriber accounts on first login attempt via SMS if this feature is enabled in portal settings. This allows for self-service user registration.

4.1.4 Creating a Subscriber

Via Web Interface

1. **Open the "Subscribers" section** in the Catena control panel
2. **Click the "Create Subscriber" button**
3. **Fill in required fields:**
4. **Name** — subscriber name or identifier
5. **Phone Country Code** — country code (e.g., `1` for USA)
6. **Phone** — phone number without country code
7. **Save the subscriber**
8. **Connect packages** via subscription management section

After creation, the subscriber will receive a unique ID and can login to the system via SMS.

Via Management API

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Doe",
    "phoneCountryCode": "1",
    "phone": "2345678901"
  }'
```

Response:

```
{
  "subscriberId": "sKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE.",
  "name": "John Doe",
  "phoneCountryCode": "1",
  "phone": "2345678901",
```

```
  "playback_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "packages": []
}
```

4.1.5 Viewing Subscriber List

Via Web Interface

The "Subscribers" section displays a table with all portal subscribers:

- **Name** — subscriber name or identifier
- **Phone** — complete phone number
- **Packages** — number of connected packages
- **Last Activity** — time of last login or viewing
- **Status** — active/blocked
- **Actions** — edit, package management, and delete buttons

Via Management API

Get list of all subscribers:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/subscribers \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "subscribers": [
    {
      "subscriberId": "sKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "name": "John Doe",
      "phoneCountryCode": "1",
      "phone": "2345678901",
      "playback_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "packages": ["pKl9SW3AAAE.", "bKl9SW3AAAE."]
    },
    {
      "subscriberId": "tKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "name": "Jane Smith",
      "phoneCountryCode": "1",
      "phone": "2345678902",
      "playback_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "packages": ["pKl9SW3AAAE."]
    }
  ],
  "next": "cursor-for-next-page"
}
```

Pagination:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/subscribers?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

4.1.6 Getting Subscriber Information

Via Management API

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/subscribers/sKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response: Similar to subscriber object from the list.

4.1.7 Editing a Subscriber

Via Web Interface

1. **Open the subscriber list**
2. **Find the needed subscriber** and click the "Edit" button
3. **Change parameters:**
4. Name — subscriber name
5. Phone Country Code — country code
6. Phone — phone number
7. **Save changes**

Note: When changing phone number, subscriber will need to re-authenticate via SMS with the new number.

Via Management API

```
curl -X PUT https://your-catena-domain.com/tv-management/api/v1/subscribers/sKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Michael Doe",
    "phoneCountryCode": "1",
    "phone": "2345678901"
  }'
```

4.1.8 Managing Package Subscriptions

Connecting Package to Subscriber

Via Web Interface:

1. **Open subscriber card**
2. **Go to "Packages" section**
3. **Click "Add Package"**
4. **Select package** from available list
5. **Confirm addition**

The subscriber immediately gets access to all channels from the added package.

Via Management API:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE."
  }'
```

Response:

```
{
  "subscriberId": "sKl9SW3AAAE.",
  "packageId": "pKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE."
}
```

Disconnecting Package from Subscriber

Via Web Interface:

1. **Open subscriber card**
2. **Go to "Packages" section**
3. **Find package** in active list
4. **Click "Remove"**
5. **Confirm disconnection**

The subscriber immediately loses access to all channels from the disconnected package.

Via Management API:

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE."
  }'
```

Bulk Subscription Management

For bulk connecting or disconnecting packages use loops or scripts. Example of adding a package to multiple subscribers:

```
#!/bin/bash
SUBSCRIBERS=("sKl9SW3AAAE." "tKl9SW3AAAE." "uKl9SW3AAAE.")
PACKAGE_ID="pKl9SW3AAAE."

for SUBSCRIBER_ID in "${SUBSCRIBERS[@]}"; do
  curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
    -H "X-Auth-Token: your-api-key" \
    -H "Content-Type: application/json" \
    -d "{
      \"subscriberId\": \"$SUBSCRIBER_ID\",
      \"packageId\": \"$PACKAGE_ID\"
    }"
done
```

4.1.9 Deleting a Subscriber

Via Web Interface

1. **Open the subscriber list**
2. **Find the subscriber** to delete
3. **Click "Delete" button**
4. **Confirm deletion**

Warning: When deleting a subscriber:

- Account will be completely deleted
- All package subscriptions will be cancelled
- Viewing history will be preserved for analytics
- Subscriber will lose access to watch channels
- Account recovery will be impossible

Via Management API

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/subscribers/sKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

4.1.10 Monitoring Subscriber Activity

Viewing Playback Sessions

Catena automatically registers all channel viewing sessions by subscribers. This allows tracking activity, channel popularity, and identifying issues.

Get sessions for specific subscriber:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Get only active sessions:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&active=true" \
  -H "X-Auth-Token: your-api-key"
```

Filter by time:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&opened_at_gte=1714233600&opened_at_lt=1714320000" \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "sessions": [
    {
      "sessionId": "sessKl9SW3AAAE.",
      "subscriberId": "sKl9SW3AAAE.",
      "channelId": "chKl9SW3AAAE.",
      "channelName": "sport1",
      "portalId": "pKl9SW3AAAE.",
      "openedAt": 1714233600,
      "closedAt": 1714237200,
      "active": false,
      "bytes": 5242880000,
      "ip": "192.168.1.100",
      "userAgent": "VLC/3.0.16"
    }
  ],
  "next": "cursor-for-next-page"
}
```

Operations Log

All changes in subscriber accounts (creation, deletion, subscription changes) are recorded in the operations log.

Get operations for a subscriber:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/operations?subscriberId=sKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Filter by operation type:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/operations?subscriberId=sKl9SW3AAAE.&type=createPackageSubscriber" \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "operations": [
    {
      "operationId": "opKl9SW3AAAE.",
      "type": "createSubscriber",
      "subscriberId": "sKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "createdAt": "2024-10-16T10:00:00Z",
      "payload": {
        "name": "John Doe",
        "phone": "+12345678901"
      }
    },
    {
      "operationId": "opKl9SW3AAAB.",
      "type": "createPackageSubscriber",
      "subscriberId": "sKl9SW3AAAE.",
      "packageId": "pKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
```

```
      "createdAt": "2024-10-16T10:05:00Z",
      "payload": {
        "packageId": "pKl9SW3AAAE."
      }
    }
  ],
  "next": "cursor-for-next-page"
}
```

4.1.11 Typical Use Cases

Creating New Subscriber with Basic Package

Task: Register a new subscriber and connect basic package

Steps:

1. Create subscriber account via API
2. Get `subscriberId` from response
3. Connect basic package via packages-subscribers API
4. Subscriber receives SMS to login to app
5. After login, subscriber sees channels from basic package

Example script:

```
#!/bin/bash

# 1. Create subscriber
RESPONSE=$(curl -s -X POST https://your-catena-domain.com/tv-management/api/v1/subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "New Subscriber",
    "phoneCountryCode": "1",
    "phone": "2345678901"
  }')

# 2. Extract subscriber ID
SUBSCRIBER_ID=$(echo $RESPONSE | jq -r '.subscriberId')

# 3. Connect basic package
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d "{
    \"subscriberId\": \"$SUBSCRIBER_ID\",
    \"packageId\": \"basic-package-id\"
  }"

echo "Subscriber created with ID: $SUBSCRIBER_ID"
```

Upgrading Subscriber to Premium Package

Task: Move subscriber from basic to premium package

Option 1: Add premium to basic

```
# Subscriber will get access to channels from both packages
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "premium-package-id"
  }'
```

Option 2: Replace basic with premium

```
# First disconnect basic
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "basic-package-id"
  }'
```

```
# Then connect premium
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "premium-package-id"
  }'
```

Integration with Billing System

Task: Automatically manage subscriptions based on payments

Concept:

1. Billing system tracks subscriber payments
2. On successful payment, billing calls Catena API to connect package
3. On subscription expiry, billing disconnects package via API
4. Catena automatically manages channel access

Example webhook from billing:

```
import requests

def on_payment_success(subscriber_phone, package_name):
    # 1. Find subscriber by phone
    subscribers = requests.get(
        f"https://catena.example.com/tv-management/api/v1/subscribers",
        headers={"X-Auth-Token": "your-api-key"}
    ).json()

    subscriber = next(
        s for s in subscribers['subscribers']
        if f"+{s['phoneCountryCode']}{s['phone']}" == subscriber_phone
    )

    # 2. Connect paid package
    requests.post(
        "https://catena.example.com/tv-management/api/v1/packages-subscribers",
        headers={"X-Auth-Token": "your-api-key"},
        json={
            "subscriberId": subscriber['subscriberId'],
            "packageId": get_package_id(package_name)
        }
    )

def on_subscription_expired(subscriber_phone, package_name):
    # Similar, but via DELETE
    pass
```

Bulk Subscriber Migration

Task: Migrate subscribers from old system to Catena

Steps:

1. Export subscriber data from old system (CSV/JSON)
2. Create bulk import script via API
3. Create accounts in Catena
4. Connect corresponding packages
5. Notify subscribers about transition to new system

Example import script:

```
import csv
import requests

API_URL = "https://catena.example.com/tv-management/api/v1"
API_KEY = "your-api-key"

def import_subscribers(csv_file):
    with open(csv_file, 'r') as f:
        reader = csv.DictReader(f)
```

```
        for row in reader:
            # Create subscriber
            response = requests.post(
                f"{API_URL}/subscribers",
                headers={"X-Auth-Token": API_KEY},
                json={
                    "name": row['name'],
                    "phoneCountryCode": row['country_code'],
                    "phone": row['phone']
                }
            )

            subscriber_id = response.json()['subscriberId']

            # Connect packages
            for package_id in row['packages'].split(','):
                requests.post(
                    f"{API_URL}/packages-subscribers",
                    headers={"X-Auth-Token": API_KEY},
                    json={
                        "subscriberId": subscriber_id,
                        "packageId": package_id.strip()
                    }
                )

            print(f"Imported: {row['name']} ({subscriber_id})")

import_subscribers('subscribers.csv')
```

4.1.12 Best Practices

Managing Phone Numbers

Recommendations:

- **Input validation** — verify number format before sending to API
- **Uniqueness** — one phone number = one subscriber
- **International format** — store country code and number separately
- **Number change** — require confirmation via SMS to new number
- **Deactivation** — promptly update data when operator disconnects number

Security

Protecting playback tokens:

- Don't transmit `playback_token` to third parties
- Use HTTPS for all API requests
- Regularly update tokens if compromise suspected
- Log access attempts with invalid tokens

Access control:

- Limit number of simultaneous sessions per subscriber
- Track suspicious activity (different IPs, different devices)
- Block subscribers upon fraud detection

Subscription Management

Recommendations:

- **Smooth transition** — notify about subscription changes in advance
- **Automation** — integrate with billing for automatic management
- **Free packages** — use portal free packages for demo content
- **Trial periods** — temporarily connect premium packages for trial
- **Change history** — use operations log for audit

Subscriber Communication

When to send notifications:

- Upon account creation
- Upon subscription changes
- Upon paid period expiry
- Upon phone number change
- Upon access blocking

Communication channels:

- SMS — for login codes and critical notifications
- Email — for informational newsletters (if available in your system)
- Push notifications — via mobile app
- In-app messages — upon app login

4.1.13 Troubleshooting

Subscriber Cannot Login via SMS

Possible causes:

- Phone number incorrectly specified during registration
- SMS not delivered (carrier issues)
- Code from SMS expired
- Phone number blocked in SMS gateway

Solution:

1. Check phone number in subscriber account
2. Ensure number format is correct (+country_code + number)
3. Check SMS gateway logs for message delivery
4. Try sending SMS again
5. If SMS doesn't arrive — check SMS gateway balance and settings

Subscriber Doesn't See Channels

Possible causes:

- Subscriber has no connected packages
- Packages contain no channels
- Playback token expired or invalid
- Technical issues with streaming server

Solution:

1. Check package list in subscriber's `packages` field
2. Ensure packages contain channels
3. Verify channels are active and working
4. Regenerate `playback_token` if needed
5. Check streaming server logs

Errors When Connecting Package

Possible causes:

- Package already connected to subscriber
- Incorrect `packageId` or `subscriberId` specified
- Package and subscriber belong to different portals
- Package doesn't exist or deleted

Solution:

1. Check subscriber's current packages via GET `/subscribers/{id}`
2. Ensure package and subscriber IDs are correct
3. Verify package exists via GET `/packages/{id}`
4. Ensure `portalId` matches for package and subscriber

Duplicate Phone Numbers

Problem: Attempt to create subscriber with existing number

Solution:

- API should return error when creating duplicate
- Check for subscriber with such number before creation
- Use UPDATE instead of CREATE for existing subscribers
- Implement phone number search in your interface

4.1.14 See Also

- **Channel Package Management** — creating and configuring packages for subscribers
- **Channel Management** — setting up TV channels
- **Operations Log** — tracking system changes
- **Play Sessions** — monitoring viewing activity

4.2 Subscription Management

Subscriptions are the relationships between subscribers and channel packages that determine which channels each subscriber has access to. The subscription system is the key mechanism for monetizing IPTV services in Catena.

4.2.1 What is a Subscription

A subscription in Catena is an active link between a subscriber and a channel package. When a subscriber has a subscription to a package, they automatically get access to all channels included in that package.

Key concept:

```
Subscriber → Subscription → Package → Channels → Viewing
```

1. **Subscriber** subscribes to one or more **packages**
2. Each **package** contains a set of **channels**
3. Subscriber gets access to all **channels** from all their packages
4. System checks subscription presence when viewing

Key capabilities:

- **Flexible access control** — connecting and disconnecting packages in real-time
- **Multiple subscriptions** — subscriber can be subscribed to multiple packages simultaneously
- **Free packages** — automatic access to basic content for all subscribers
- **Logging** — complete history of all subscription changes
- **API-first approach** — easy integration with billing systems

Typical workflow:

1. Billing system receives payment from user
2. Billing calls Catena API to create subscription
3. Catena immediately grants access to package channels
4. Subscriber starts watching channels
5. At period end, billing disconnects subscription
6. Access to paid channels is automatically blocked

4.2.2 Subscription Lifecycle

Creating a Subscription

When subscription is created:

- Upon package payment through billing system
- Upon manual connection by administrator
- Upon promo code or bonus activation
- Upon trial period provision

What happens when created:

1. Record is created in database about subscriber-package link
2. Subscriber immediately gets access to all package channels
3. Record is added to operations log (type `createPackageSubscriber`)
4. On next player request, available channels list is updated

Active Subscription

During active subscription:

- Subscriber can watch all package channels without restrictions
- System logs all viewing sessions
- Subscriber's `packages` field contains active package IDs
- Streaming server verifies access rights on each stream request

Cancelling a Subscription

When subscription is cancelled:

- Upon paid period expiration
- Upon subscription cancellation by user
- Upon subscriber blocking by administrator
- Upon package deletion from system

What happens when cancelled:

1. Subscriber-package link record is deleted
2. Subscriber immediately loses access to package channels
3. Record is added to operations log (type `deletePackageSubscriber`)
4. Active viewing sessions of package channels are interrupted

4.2.3 Creating a Subscription

Via Web Interface

1. **Open subscriber card** in "Subscribers" section
2. **Go to "Subscriptions" or "Packages" tab**
3. **Click "Add Subscription"**
4. **Select package** from dropdown list of available packages
5. **Confirm addition**

Subscriber immediately gets access to all channels of the selected package.

Via Management API

Create subscriber subscription to package:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE."
  }'
```

Request parameters:

- **subscriberId** (required) — ID of subscriber to connect package to
- **packageId** (required) — ID of package to connect

Response:

```
{
  "subscriberId": "sKl9SW3AAAE.",
  "packageId": "pKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE."
}
```

Important points:

- Subscriber and package must belong to same portal
- If subscription already exists, API will return error
- Changes take effect immediately
- Operation is recorded in log

4.2.4 Deleting a Subscription

Via Web Interface

1. **Open subscriber card**
2. **Go to "Subscriptions" tab**
3. **Find package** in active subscriptions list
4. **Click "Delete" or "Disconnect"**
5. **Confirm disconnection**

Subscriber immediately loses access to channels of this package.

Via Management API

Delete subscriber subscription to package:

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/packages-subscribers \
  -H "X-Auth-Token: your-api-key" \
  -H "Content-Type: application/json" \
  -d '{
    "subscriberId": "sKl9SW3AAAE.",
    "packageId": "pKl9SW3AAAE."
  }'
```

Request parameters:

- **subscriberId** (required) — subscriber ID
- **packageId** (required) — package ID to disconnect

Response:

HTTP 201 - subscription deleted

Important points:

- If subscription doesn't exist, API will return error
- Active viewing sessions will be interrupted
- Changes take effect immediately
- Operation is recorded in log

4.2.5 Viewing Subscriptions

Specific Subscriber Subscriptions

Get subscriber's package list:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/subscribers/sKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "subscriberId": "sKl9SW3AAAE.",
  "portalId": "pKl9SW3AAAE.",
  "name": "John Doe",
  "phoneCountryCode": "1",
  "phone": "2345678901",
  "playback_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "packages": ["pKl9SW3AAAE.", "sportKl9SW3AAAE."]
}
```

The `packages` field contains array of IDs of all packages the subscriber is subscribed to.

Specific Package Subscribers

Unfortunately, there's no direct API to get list of package subscribers. Use operations log or get all subscribers and filter by `packages`:

```
# Get all subscribers
curl -X GET https://your-catena-domain.com/tv-management/api/v1/subscribers \
  -H "X-Auth-Token: your-api-key" \
  | jq '.subscribers[] | select(.packages[] | contains("pKl9SW3AAAE."))'
```

Subscription History via Operations Log

Get all subscription operations for specific subscriber:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/operations?subscriberId=sKl9SW3AAAE.&type=createPackageSubscriber&type=deletePackageSubscriber"
\
  -H "X-Auth-Token: your-api-key"
```

Get operations for specific package:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/operations?packageId=pKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "operations": [
    {
      "operationId": "opKl9SW3AAAE.",
      "type": "createPackageSubscriber",
      "subscriberId": "sKl9SW3AAAE.",
      "packageId": "pKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "createdAt": "2024-10-16T10:05:00Z",
      "payload": {
        "packageId": "pKl9SW3AAAE.",
        "subscriberId": "sKl9SW3AAAE."
      }
    },
    {
      "operationId": "opKl9SW3AAAB.",
      "type": "deletePackageSubscriber",
      "subscriberId": "sKl9SW3AAAE.",
      "packageId": "pKl9SW3AAAE.",
      "portalId": "pKl9SW3AAAE.",
      "createdAt": "2024-10-20T15:30:00Z",
      "payload": {
        "packageId": "pKl9SW3AAAE.",
        "subscriberId": "sKl9SW3AAAE."
      }
    }
  ],
  "next": "cursor-for-next-page"
}
```

Operation types:

- `createPackageSubscriber` — subscription creation
- `deletePackageSubscriber` — subscription deletion
- `autoCreateSubscriber` — automatic subscriber creation (may include basic package subscription)

4.2.6 Portal Free Packages

Catena supports the concept of "free packages" — packages that are automatically available to all portal subscribers without explicit subscription creation.

Free Packages Concept

How it works:

- Portal settings define list of free packages
- All portal subscribers automatically get access to channels from these packages
- No need to create individual subscriptions for each subscriber
- Perfect for basic content, demo channels, promotional channels

Use cases:

- **Basic content** — public channels available to all
- **Trial period** — demo content for new users
- **Promo channels** — advertising and informational channels
- **Public interest channels** — mandatory distribution channels

Managing Free Packages

View portal free packages:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/portal \
  -H "X-Auth-Token: your-api-key"
```

Response:

```
{
  "portalId": "pKl9SW3AAAE.",
  "name": "my-iptv-portal",
  "domain": "iptv.example.com",
  "freePackages": ["basicKl9SW3AAAE.", "demoKl9SW3AAAE."],
  "branding": {
    "title": "My IPTV Service",
    "description": "Premium IPTV streaming"
  }
}
```

Add package to free list:

```
curl -X POST https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/basicKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Remove package from free list:

```
curl -X DELETE https://your-catena-domain.com/tv-management/api/v1/portal/free-packages/basicKl9SW3AAAE. \
  -H "X-Auth-Token: your-api-key"
```

Important:

- Free package changes apply to all subscribers instantly
- On addition — all subscribers get access to package channels
- On removal — only those without explicit subscription lose access

4.2.7 Billing System Integration

Integration Architecture

Typical scheme:

```
[Billing System] ←→ [Catena API] ←→ [Streaming Server]
       ↓                  ↓                ↓
   Payments        Subscriptions    Channel Access
```

Billing responsibilities:

- Receiving payments from users
- Managing tariffs and subscription periods
- Tracking subscription expirations
- Calling Catena API to connect/disconnect packages

Catena responsibilities:

- Managing channel access
- Verifying rights during viewing
- Logging subscriber activity
- Providing viewing statistics

Integration Examples

EXAMPLE 1: WEBHOOK ON PAYMENT

Billing sends webhook to your service on successful payment:

```
from flask import Flask, request
import requests

app = Flask(__name__)

CATENA_API_URL = "https://catena.example.com/tv-management/api/v1"
CATENA_API_KEY = "your-api-key"

@app.route('/billing-webhook', methods=['POST'])
def billing_webhook():
    data = request.json

    if data['event'] == 'payment.success':
        # Payment received - activate subscription
        subscriber_id = get_subscriber_id(data['user_phone'])
        package_id = get_package_id(data['tariff_name'])

        # Create subscription in Catena
        response = requests.post(
            f"{CATENA_API_URL}/packages-subscribers",
            headers={"X-Auth-Token": CATENA_API_KEY},
            json={
                "subscriberId": subscriber_id,
                "packageId": package_id
            }
        )

        if response.status_code == 200:
            return {"status": "ok", "message": "Subscription activated"}
        else:
            return {"status": "error", "message": response.text}, 500

    elif data['event'] == 'subscription.expired':
        # Subscription expired - deactivate
        subscriber_id = get_subscriber_id(data['user_phone'])
        package_id = get_package_id(data['tariff_name'])
```

```
        # Delete subscription in Catena
        response = requests.delete(
            f"{CATENA_API_URL}/packages-subscribers",
            headers={"X-Auth-Token": CATENA_API_KEY},
            json={
                "subscriberId": subscriber_id,
                "packageId": package_id
            }
        )

        return {"status": "ok", "message": "Subscription deactivated"}

    return {"status": "ok"}

def get_subscriber_id(phone):
    """Get Catena subscriber ID by phone number"""
    response = requests.get(
        f"{CATENA_API_URL}/subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY}
    )
    subscribers = response.json()['subscribers']

    for sub in subscribers:
        full_phone = f"+{sub['phoneCountryCode']}{sub['phone']}"
        if full_phone == phone:
            return sub['subscriberId']

    # If subscriber not found - create
    return create_subscriber(phone)

def get_package_id(tariff_name):
    """Map tariff name to package ID"""
    tariff_mapping = {
        "basic": "basicKl9SW3AAAE.",
        "premium": "premiumKl9SW3AAAE.",
        "sport": "sportKl9SW3AAAE."
    }
    return tariff_mapping.get(tariff_name)

if __name__ == '__main__':
    app.run(port=5000)
```

EXAMPLE 2: PERIODIC SYNCHRONIZATION

Regular checking and synchronization of subscriptions:

```
import requests
from datetime import datetime, timedelta

CATENA_API_URL = "https://catena.example.com/tv-management/api/v1"
CATENA_API_KEY = "your-api-key"
BILLING_DB = "postgresql://billing_db"

def sync_subscriptions():
    """Synchronize subscriptions between billing and Catena"""

    # 1. Get active subscriptions from billing
    active_billing_subscriptions = get_active_subscriptions_from_billing()

    # 2. Get all Catena subscribers
    response = requests.get(
        f"{CATENA_API_URL}/subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY}
    )
    catena_subscribers = response.json()['subscribers']

    # 3. Compare and synchronize
    for billing_sub in active_billing_subscriptions:
        phone = billing_sub['phone']
        package_id = get_package_id(billing_sub['tariff'])

        # Find subscriber in Catena
        catena_sub = find_subscriber_by_phone(catena_subscribers, phone)

        if catena_sub:
            # Check if needed subscription exists
            if package_id not in catena_sub['packages']:
                # No subscription - create
                create_subscription(catena_sub['subscriberId'], package_id)
                print(f"Activated: {phone} -> {package_id}")
        else:
            # No subscriber - create with subscription
            create_subscriber_with_package(phone, package_id)
            print(f"Created subscriber: {phone}")

    # 4. Disconnect expired subscriptions
    for catena_sub in catena_subscribers:
        phone = f"+{catena_sub['phoneCountryCode']}{catena_sub['phone']}"

        for package_id in catena_sub['packages']:
            if not has_active_billing_subscription(phone, package_id):
```

```
                # No subscription in billing - remove from Catena
                delete_subscription(catena_sub['subscriberId'], package_id)
                print(f"Deactivated: {phone} -> {package_id}")

def create_subscription(subscriber_id, package_id):
    requests.post(
        f"{CATENA_API_URL}/packages-subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY},
        json={
            "subscriberId": subscriber_id,
            "packageId": package_id
        }
    )

def delete_subscription(subscriber_id, package_id):
    requests.delete(
        f"{CATENA_API_URL}/packages-subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY},
        json={
            "subscriberId": subscriber_id,
            "packageId": package_id
        }
    )

# Run this function on schedule (e.g., every hour)
if __name__ == '__main__':
    sync_subscriptions()
```

EXAMPLE 3: TRIAL PERIOD

Automatic trial period for new subscribers:

```
import requests
from datetime import datetime, timedelta

def activate_trial_subscription(phone, trial_days=7):
    """Activate trial subscription for N days"""

    # 1. Create or get subscriber
    subscriber_id = get_or_create_subscriber(phone)

    # 2. Connect trial package
    trial_package_id = "trialKl9SW3AAAE."

    response = requests.post(
        f"{CATENA_API_URL}/packages-subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY},
        json={
            "subscriberId": subscriber_id,
            "packageId": trial_package_id
        }
    )

    if response.status_code == 200:
        # 3. Schedule automatic cancellation
        schedule_subscription_cancellation(
            subscriber_id,
            trial_package_id,
            datetime.now() + timedelta(days=trial_days)
        )

        return {
            "success": True,
            "message": f"Trial activated for {trial_days} days",
            "expires_at": (datetime.now() + timedelta(days=trial_days)).isoformat()
        }

    return {"success": False, "error": response.text}

def schedule_subscription_cancellation(subscriber_id, package_id, cancel_date):
    """Schedule subscription cancellation"""
    # Save to task database or use scheduler
    # E.g., Celery, APScheduler, or cron job
    pass
```

4.2.8 Typical Use Cases

Auto-Renewal Subscription

Task: Implement monthly subscription with automatic renewal

Solution:

1. Billing charges payment each month
2. On successful charge, billing checks subscription presence in Catena
3. If subscription exists — do nothing (it's already active)
4. If no subscription — create via API
5. On failed charge — delete subscription via API

```
def process_monthly_renewal(user_id, package_name):
    """Process monthly renewal"""

    # Charge attempt
    payment_success = billing_charge(user_id, get_package_price(package_name))

    subscriber_id = get_subscriber_id_by_user(user_id)
    package_id = get_package_id(package_name)

    if payment_success:
        # Payment successful - ensure subscription is active
        ensure_subscription_active(subscriber_id, package_id)
    else:
        # Payment failed - disconnect subscription
        deactivate_subscription(subscriber_id, package_id)
        send_notification(user_id, "payment_failed")
```

Family Subscription

Task: One payment — access for multiple subscribers (family account)

Solution:

1. Create family tariff in billing
2. On payment, connect package to all family subscribers
3. Store link between subscribers in billing

```
def activate_family_subscription(family_id, package_name):
    """Activate family subscription"""

    package_id = get_package_id(package_name)

    # Get all family members from billing
    family_members = get_family_members(family_id)

    for member in family_members:
        subscriber_id = get_subscriber_id(member['phone'])

        # Connect package to each
        requests.post(
            f"{CATENA_API_URL}/packages-subscribers",
            headers={"X-Auth-Token": CATENA_API_KEY},
            json={
                "subscriberId": subscriber_id,
                "packageId": package_id
            }
        )

    return {"activated": len(family_members)}
```

Temporary Promotion

Task: Give access to premium channels for the weekend

Solution:

1. Friday evening — connect promo package to all active subscribers
2. Monday morning — disconnect promo package

```
#!/bin/bash
# friday-promo.sh - run via cron on Friday at 6 PM

PROMO_PACKAGE_ID="weekendKl9SW3AAAE."

# Get all subscribers
```

```
SUBSCRIBERS=$(curl -s -X GET "$CATENA_API_URL/subscribers" \
  -H "X-Auth-Token: $CATENA_API_KEY" \
  | jq -r '.subscribers[].subscriberId')

# Connect promo package to each
for SUBSCRIBER_ID in $SUBSCRIBERS; do
  curl -X POST "$CATENA_API_URL/packages-subscribers" \
    -H "X-Auth-Token: $CATENA_API_KEY" \
    -H "Content-Type: application/json" \
    -d "{
      \"subscriberId\": \"$SUBSCRIBER_ID\",
      \"packageId\": \"$PROMO_PACKAGE_ID\"
    }"
done

echo "Promo activated for $(echo "$SUBSCRIBERS" | wc -l) subscribers"
```

```
#!/bin/bash
# monday-cleanup.sh - run via cron on Monday at 6 AM

PROMO_PACKAGE_ID="weekendKl9SW3AAAE."

SUBSCRIBERS=$(curl -s -X GET "$CATENA_API_URL/subscribers" \
  -H "X-Auth-Token: $CATENA_API_KEY" \
  | jq -r '.subscribers[] | select(.packages[] | contains("'$PROMO_PACKAGE_ID'")) | .subscriberId')

for SUBSCRIBER_ID in $SUBSCRIBERS; do
  curl -X DELETE "$CATENA_API_URL/packages-subscribers" \
    -H "X-Auth-Token: $CATENA_API_KEY" \
    -H "Content-Type: application/json" \
    -d "{
      \"subscriberId\": \"$SUBSCRIBER_ID\",
      \"packageId\": \"$PROMO_PACKAGE_ID\"
    }"
done

echo "Promo deactivated for $(echo "$SUBSCRIBERS" | wc -l) subscribers"
```

Plan Downgrade

Task: Subscriber moves from premium to basic plan

Solution:

```
def downgrade_subscription(subscriber_id, from_package, to_package):
    """Downgrade subscriber plan"""

    from_package_id = get_package_id(from_package)
    to_package_id = get_package_id(to_package)

    # 1. Disconnect premium package
    requests.delete(
        f"{CATENA_API_URL}/packages-subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY},
        json={
            "subscriberId": subscriber_id,
            "packageId": from_package_id
        }
    )

    # 2. Connect basic package
    requests.post(
        f"{CATENA_API_URL}/packages-subscribers",
        headers={"X-Auth-Token": CATENA_API_KEY},
        json={
            "subscriberId": subscriber_id,
            "packageId": to_package_id
        }
    )

    # 3. Record in billing
    billing_record_downgrade(subscriber_id, from_package, to_package)

    return {"success": True, "new_package": to_package}
```

4.2.9 Best Practices

Package Design

Package structure recommendations:

- **Basic package** — minimal channel set for all
- **Thematic packages** — sports, movies, kids, news
- **Premium packages** — exclusive content, HD/4K channels
- **Combo packages** — multiple themes in one (savings for subscriber)

Avoid:

- Too many small packages — complicates choice
- Channel duplication between packages — billing confusion
- Package overlaps without logic — one channel in 5 different packages

Error Handling

When integrating with billing:

```
def safe_create_subscription(subscriber_id, package_id, retry_count=3):
    """Create subscription with retries"""

    for attempt in range(retry_count):
        try:
            response = requests.post(
                f"{CATENA_API_URL}/packages-subscribers",
                headers={"X-Auth-Token": CATENA_API_KEY},
                json={
                    "subscriberId": subscriber_id,
                    "packageId": package_id
                },
                timeout=10
            )

            if response.status_code == 200:
                return {"success": True}
            elif response.status_code == 409:
                # Subscription already exists - this is OK
                return {"success": True, "already_exists": True}
            else:
                # Other error
                error_msg = response.json().get('message', 'Unknown error')
                log_error(f"Failed to create subscription: {error_msg}")

        except requests.exceptions.Timeout:
            log_warning(f"Timeout on attempt {attempt + 1}")
            if attempt < retry_count - 1:
                time.sleep(2 ** attempt)  # Exponential backoff
            continue
        except Exception as e:
            log_error(f"Unexpected error: {str(e)}")
            break

    # All attempts failed - save for manual processing
    save_failed_operation("create_subscription", subscriber_id, package_id)
    return {"success": False, "error": "Failed after retries"}
```

State Synchronization

Regular data reconciliation:

```
def audit_subscriptions():
    """Check subscription consistency between systems"""

    discrepancies = []

    # Get data from both systems
    billing_subscriptions = get_billing_subscriptions()
    catena_subscriptions = get_catena_subscriptions()

    # Find discrepancies
    for billing_sub in billing_subscriptions:
        if not exists_in_catena(billing_sub, catena_subscriptions):
            discrepancies.append({
```

```
                "type": "missing_in_catena",
                "subscriber": billing_sub['phone'],
                "package": billing_sub['package']
            })

    for catena_sub in catena_subscriptions:
        if not exists_in_billing(catena_sub, billing_subscriptions):
            discrepancies.append({
                "type": "missing_in_billing",
                "subscriber": catena_sub['phone'],
                "package": catena_sub['package']
            })

    if discrepancies:
        # Send notification to administrator
        send_audit_report(discrepancies)

        # Optionally: auto-fix
        auto_fix_discrepancies(discrepancies)

    return discrepancies
```

Logging and Monitoring

What to log:

- All subscription creations and deletions
- Errors when calling API
- Catena API response time
- Discrepancies between billing and Catena

Metrics to track:

```
import prometheus_client as prom

# Prometheus metrics
subscription_creations = prom.Counter(
    'catena_subscription_creations_total',
    'Total number of subscription creations',
    ['package_name', 'status']
)

subscription_deletions = prom.Counter(
    'catena_subscription_deletions_total',
    'Total number of subscription deletions',
    ['package_name', 'status']
)

api_latency = prom.Histogram(
    'catena_api_latency_seconds',
    'Latency of Catena API calls',
    ['endpoint', 'method']
)

def monitored_create_subscription(subscriber_id, package_id):
    """Create subscription with monitoring"""

    package_name = get_package_name(package_id)

    with api_latency.labels('/packages-subscribers', 'POST').time():
        try:
            response = requests.post(...)

            if response.status_code == 200:
                subscription_creations.labels(package_name, 'success').inc()
                return {"success": True}
            else:
                subscription_creations.labels(package_name, 'error').inc()
                return {"success": False}

        except Exception as e:
            subscription_creations.labels(package_name, 'exception').inc()
            raise
```

Subscriber Notifications

When to send notifications:

1. **On subscription activation** — "Welcome! Now available channels: ..."
2. **3 days before expiration** — "Your subscription expires in 3 days"
3. **On renewal** — "Subscription renewed until ..."
4. **On cancellation** — "Subscription cancelled. To renew..."
5. **On payment error** — "Failed to charge. Please check..."

```
def notify_subscription_activated(subscriber_id, package_name):
    """Notification about subscription activation"""

    subscriber = get_subscriber(subscriber_id)
    phone = f"+{subscriber['phoneCountryCode']}{subscriber['phone']}"

    # Get package channel list
    package = get_package(get_package_id(package_name))
    channels = ", ".join(package['channels'][:5])  # First 5 channels

    message = f"""
      Subscription activated!

    Package: {package_name}
    Available channels: {channels} and more

    Enjoy watching!
    """

    send_sms(phone, message)
```

4.2.10 Troubleshooting

Subscription Not Creating

Possible causes:

- Invalid `subscriberId` or `packageId`
- Subscriber and package from different portals
- Subscription already exists
- API authorization issues

Solution:

1. Check subscriber existence: `GET /subscribers/{id}`
2. Check package existence: `GET /packages/{id}`
3. Ensure `portalId` matches
4. Check subscriber's current subscriptions
5. Verify API key validity

Subscriber Doesn't See Channels After Subscription Creation

Possible causes:

- Package contains no channels
- App didn't update channel list
- Streaming server issues

Solution:

1. Check package contents: `GET /packages/{id}`
2. Ensure package has channels
3. Ask subscriber to restart app
4. Check subscriber's `playback_token`
5. Check streaming server logs

Subscription Not Deleting

Possible causes:

- Subscription doesn't exist (already deleted)
- Invalid request parameters
- It's a portal free package (can't delete)

Solution:

1. Check subscriber's current subscriptions
2. Ensure it's not a portal free package
3. Verify `subscriberId` and `packageId` correctness
4. Check operations log for this subscriber

Discrepancies Between Billing and Catena

Problem: Subscription active in billing but not in Catena (or vice versa)

Solution:

1. Implement regular synchronization (every 15-60 minutes)
2. Use operations log to identify issues
3. On discrepancy, billing has priority (source of truth)
4. Log all changes for analysis

```
def fix_sync_issue(subscriber_id):
    """Fix desynchronization for subscriber"""

    # 1. Get "truth" from billing
    billing_packages = get_billing_packages(subscriber_id)

    # 2. Get current state in Catena
    subscriber = get_catena_subscriber(subscriber_id)
    catena_packages = subscriber['packages']

    # 3. Synchronize
    for package_id in billing_packages:
        if package_id not in catena_packages:
            # Should be but isn't - add
            create_subscription(subscriber_id, package_id)
            log_info(f"Fixed: added {package_id} to {subscriber_id}")

    for package_id in catena_packages:
        if package_id not in billing_packages:
            # Exists but shouldn't - remove
            delete_subscription(subscriber_id, package_id)
            log_info(f"Fixed: removed {package_id} from {subscriber_id}")
```

4.2.11 See Also

- **Subscriber Management** — creating and configuring subscriber accounts
- **Channel Package Management** — creating and configuring packages
- **Channel Management** — adding channels to packages

- **Operations Log** — tracking all subscription changes
- **Portal Configuration** — configuring free packages

5. Monitoring

5.1 Play Session Monitoring

Play sessions are records of subscribers watching channels. Catena automatically registers each stream opening and saves detailed session information for monitoring, analytics, and debugging.

5.1.1 What is a Play Session

A play session is a period of time when a subscriber watches a specific channel. Each session contains information about who, what, when, and from where they watched.

Session lifecycle:

```
Stream opening → Active viewing → Stream closing
   (openedAt)     (active: true)    (closedAt)
```

What is recorded:

- **Who is watching:** subscriberId, token
- **What is being watched:** channelId, channelName, programId
- **When:** openedAt, closedAt, updatedAt (timestamps)
- **From where:** IP address, userAgent (player/device)
- **How much:** bytes (data transferred), session duration
- **Status:** active (session open or closed)

Applications:

- **Real-time monitoring** — who is watching channels now
- **Problem debugging** — why subscriber can't watch channel
- **Viewing analytics** — popular channels, viewing time
- **Billing** — traffic consumption calculation
- **Security** — anomaly detection (one token from different IPs)
- **Statistics** — reports for content owners

5.1.2 Play Session Structure

Main Fields

Identifiers:

- **sessionId** — unique session ID
- Format: base64-encoded Snowflake ID
- Example: `sessKl9SW3AAAE.`
- Generated when opening stream
- **subscriberId** — ID of subscriber watching channel
- Link to user account
- Example: `sKl9SW3AAAE.`
- **channelId** — ID of channel being watched
- Example: `chKl9SW3AAAE.`
- **channelName** — technical channel name
- More convenient for debugging than ID
- Example: `sport1`, `news-hd`
- **programId** — program ID (if watching from archive)
- Null for live viewing
- Example: `prKl9SW3AAAE.`
- **portalId** — portal ID
- Data isolation between portals
- Example: `pKl9SW3AAAE.`

Timestamps:

- **openedAt** — Unix timestamp of session opening
- When subscriber started watching
- Example: `1714233600` (April 28, 2024, 10:00:00 UTC)
- **closedAt** — Unix timestamp of session closing
- When stream was stopped
- Null for active sessions
- Example: `1714237200`
- **updatedAt** — Unix timestamp of last update
- Updated periodically during viewing
- Used to determine "dead" connections

Network information:

- **ip** — subscriber's IP address
- Example: `192.168.1.100`, `2001:db8::1`
- Used for geolocation and anomaly detection
- **userAgent** — player User-Agent string
- Identifies device and application
- Examples:
  - `VLC/3.0.16`
  - `Mozilla/5.0 (Linux; Android 11) AppleWebKit/537.36`
  - `Catena/1.0 (Android 11; Samsung SM-G991B)`

Statistics:

- **active** — session activity flag
- `true` — session open, viewing in progress
- `false` — session closed, viewing finished
- **bytes** — data transferred in bytes
- Updated during viewing
- Example: `5242880000` (5 GB)
- Used for traffic billing
- **token** — subscriber's playback token
- Used by streaming server for authorization
- Example: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`

Device:

- **deviceId** — device identifier
- If app sends device ID
- Example: `device_android_samsung_s21`
- Helps track number of devices per subscriber

5.1.3 Getting Session List

Basic Request

Get list of all sessions:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/play-sessions \
  -H "X-Auth-Token: your-api-key"
```

Response:


```
{
  "sessions": [
    {
      "sessionId": "sessKl9SW3AAAE.",
      "subscriberId": "sKl9SW3AAAE.",
      "channelId": "chKl9SW3AAAE.",
      "channelName": "sport1",
      "programId": null,
      "portalId": "pKl9SW3AAAE.",
      "openedAt": 1714233600,
      "closedAt": null,
      "updatedAt": 1714237200,
      "active": true,
      "bytes": 1073741824,
      "ip": "192.168.1.100",
      "userAgent": "Catena/1.0 (Android 11)",
```

```
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "deviceId": "device_123"
    },
    {
      "sessionId": "sessKl9SW3AAAB.",
      "subscriberId": "sKl9SW3AAAB.",
      "channelId": "chKl9SW3AAAB.",
      "channelName": "news-hd",
      "programId": null,
      "portalId": "pKl9SW3AAAE.",
      "openedAt": 1714230000,
      "closedAt": 1714233600,
      "updatedAt": 1714233600,
      "active": false,
      "bytes": 524288000,
      "ip": "10.0.0.50",
      "userAgent": "VLC/3.0.16",
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "deviceId": null
    }
  ],
  "next": "cursor-for-next-page"
}
```


Pagination

For large data volumes, cursor-based pagination is used:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?cursor=cursor-for-next-page" \
  -H "X-Auth-Token: your-api-key"
```

Recommendations:

- Process data page by page
- Use filters to reduce volume
- For periodic monitoring, query only active sessions

5.1.4 Session Filtering

By Subscriber

Get all sessions for specific subscriber:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Applications:

- View specific user history
- Debug subscriber issues
- Analyze viewing patterns

Multiple subscribers:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&subscriberId=sKl9SW3AAAB." \
  -H "X-Auth-Token: your-api-key"
```

By Channel

Get all sessions for specific channel:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?channelId=chKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key"
```

Applications:

- Determine channel popularity
- Analyze channel load peaks
- Debug specific channel issues

Multiple channels:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?channelId=chKl9SW3AAAE.&channelId=chKl9SW3AAAB." \
  -H "X-Auth-Token: your-api-key"
```

By Activity Status

Only active sessions (who is watching now):

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?active=true" \
  -H "X-Auth-Token: your-api-key"
```

Only completed sessions (history):

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?active=false" \
  -H "X-Auth-Token: your-api-key"
```

Applications:

- **active=true** — real-time monitoring
- **active=false** — history analysis, report building

By Time

Sessions opened after specific time:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?opened_at_gte=1714233600" \
  -H "X-Auth-Token: your-api-key"
```

Sessions opened before specific time:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?opened_at_lt=1714320000" \
  -H "X-Auth-Token: your-api-key"
```

Sessions in time interval:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?opened_at_gte=1714233600&opened_at_lt=1714320000" \
  -H "X-Auth-Token: your-api-key"
```

Applications:

- Analyze views for specific period
- Build time-based graphs
- Identify peak hours

Converting dates to Unix timestamp:

```
# Current date/time
date +%s
# Result: 1714233600

# Specific date (GNU date)
date -d "2024-04-28 10:00:00" +%s

# macOS
date -j -f "%Y-%m-%d %H:%M:%S" "2024-04-28 10:00:00" +%s
```

Combined Filters

Active sessions for specific subscriber:

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&active=true" \
  -H "X-Auth-Token: your-api-key"
```

Channel sessions for last 24 hours:

```
# Current time minus 24 hours
TIMESTAMP_24H_AGO=$(date -d '24 hours ago' +%s)

curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?channelId=chKl9SW3AAAE.&opened_at_gte=$TIMESTAMP_24H_AGO" \
  -H "X-Auth-Token: your-api-key"
```

5.1.5 Typical Use Cases

Scenario 1: Real-time Monitoring

Task: Display dashboard with current viewers

Solution:

```
#!/bin/bash
# realtime-dashboard.sh

API_URL="https://catena.example.com/tv-management/api/v1"
API_KEY="your-api-key"

while true; do
  # Get active sessions
  RESPONSE=$(curl -s -X GET "$API_URL/play-sessions?active=true" \
    -H "X-Auth-Token: $API_KEY")

  # Total viewers
  TOTAL_VIEWERS=$(echo $RESPONSE | jq '.sessions | length')

  # Top-5 popular channels
  TOP_CHANNELS=$(echo $RESPONSE | jq -r '.sessions | group_by(.channelName) |
    map({channel: .[0].channelName, viewers: length}) |
    sort_by(.viewers) | reverse | .[0:5]')

  clear
  echo "=== Current Viewers ==="
  echo "Total: $TOTAL_VIEWERS"
  echo ""
  echo "Top channels:"
  echo "$TOP_CHANNELS" | jq -r '.[] | "\(.channel): \(.viewers) viewers"'

  sleep 10
done
```

Python version with Prometheus metrics:

```
import requests
import time
from prometheus_client import Gauge, start_http_server

# Metrics
active_sessions = Gauge('catena_active_sessions', 'Number of active sessions')
channel_viewers = Gauge('catena_channel_viewers', 'Viewers per channel', ['channel'])

API_URL = "https://catena.example.com/tv-management/api/v1"
API_KEY = "your-api-key"

def update_metrics():
    response = requests.get(
        f"{API_URL}/play-sessions?active=true",
        headers={"X-Auth-Token": API_KEY}
    )

    sessions = response.json()['sessions']

    # Update total count
    active_sessions.set(len(sessions))

    # Count by channels
    channels = {}
    for session in sessions:
        channel = session['channelName']
        channels[channel] = channels.get(channel, 0) + 1

    # Update channel metrics
```

```
    for channel, count in channels.items():
        channel_viewers.labels(channel=channel).set(count)

if __name__ == '__main__':
    # Start HTTP server for Prometheus
    start_http_server(8000)

    while True:
        update_metrics()
        time.sleep(30)
```

Scenario 2: Subscriber Issue Debugging

Task: Subscriber complains they can't watch channel

Debugging steps:

1. **Check subscriber's active sessions:**

```
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&active=true" \
  -H "X-Auth-Token: your-api-key"
```

Analysis: - If no sessions — authorization or network issue - If session exists — check `updatedAt` (recently updated?) - Check IP and userAgent — match subscriber's device?

1. **Check recent session history:**

```
# Last 1 hour
TIMESTAMP_1H_AGO=$(date -d '1 hour ago' +%s)

curl -X GET "https://your-catena-domain.com/tv-management/api/v1/play-sessions?subscriberId=sKl9SW3AAAE.&opened_at_gte=$TIMESTAMP_1H_AGO" \
  -H "X-Auth-Token: your-api-key"
```

What to look for: - Frequent reconnections (many short sessions) - Low data transfer (streaming issues) - Different IP addresses (subscriber switching networks)

1. **Check if subscriber can watch specific channel:**

```
# Check subscriptions
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/subscribers/sKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key" \
  | jq '.packages'

# Check which packages have this channel
curl -X GET "https://your-catena-domain.com/tv-management/api/v1/channels/chKl9SW3AAAE." \
  -H "X-Auth-Token: your-api-key" \
  | jq '.packages'
```

5.1.6 Best Practices

Periodic Data Collection

Recommendations:

- **Active sessions:** poll every 30-60 seconds for monitoring
- **History:** collect once daily for analysis
- **Archiving:** move old data (>30 days) to cold storage

Example cron jobs:

```
# Every minute — monitor active sessions
*/1 * * * * /usr/local/bin/monitor-active-sessions.sh

# Every hour — collect statistics
0 * * * * /usr/local/bin/collect-hourly-stats.sh

# Daily at 01:00 — generate reports
0 1 * * * /usr/local/bin/generate-daily-report.sh
```

Query Optimization

Use filters to reduce data volume:

```
# BAD — get all sessions
curl -X GET "$API_URL/play-sessions"

# GOOD — only active
curl -X GET "$API_URL/play-sessions?active=true"

# BETTER — active from last hour
TIMESTAMP_1H=$(date -d '1 hour ago' +%s)
curl -X GET "$API_URL/play-sessions?active=true&opened_at_gte=$TIMESTAMP_1H"
```

5.1.7 Troubleshooting

Sessions Not Created

Problem: Subscribers watching but sessions don't appear in API

Possible causes:

1. Streaming server not integrated with Management API
2. Incorrect webhook configuration on streaming server
3. Network issues between servers

Solution:

1. Check streaming server (Flussonic) configuration
2. Ensure webhook configured for Management API
3. Check streaming server logs for errors

Sessions Not Closing

Problem: Sessions remain active after viewing stopped

Causes:

- Subscriber closed app without proper stream stop
- Network connection lost
- Streaming server didn't send closing webhook

Solution:

- Sessions have timeout (usually 5-10 minutes of inactivity)
- Check `updatedAt` field — if not updated recently, session is "dead"
- Configure automatic cleanup of "stuck" sessions

5.1.8 See Also

- **Subscriber Management** — creating and configuring accounts
- **Channel Management** — setting up TV channels
- **Operations Log** — system action audit
- **Portal Management** — data isolation between portals

5.2 Operations Log

Operations log is a complete audit log of all actions with subscribers, packages, and subscriptions in Catena. Every change is recorded with a timestamp, enabling history tracking and **billing calculations**.

5.2.1 What is an Operation

An operation is a record of a specific action performed in the system. Each operation contains information about what was done, when, and with which objects.

Why operations log is needed:

- **Revenue calculation** — counting created and cancelled subscriptions for billing
- **Action audit** — who, what, and when did in the system
- **Problem debugging** — change history to identify causes
- **Business analytics** — growth metrics, churn rate, popular packages
- **Security** — tracking suspicious actions
- **Compliance** — evidence for regulators and auditors

What is recorded:

```
Subscription creation → Operation createPackageSubscriber
    ↓
Billing period start
    ↓
Subscription cancellation → Operation deletePackageSubscriber
    ↓
Cost calculation = (cancel date - create date) × price
```

5.2.2 Operation Types

Subscriber Operations

autoCreateSubscriber — automatic subscriber creation

- Happens on first SMS login (if auto-registration enabled)
- System creates account "on the fly"
- May automatically connect free packages

createSubscriber — manual subscriber creation

- Administrator or billing created account
- Via web interface or Management API
- Usually followed by package connection

deleteSubscriber — subscriber deletion

- Complete account removal
- Automatically cancels all subscriptions
- Irreversible action

disableSubscriber — subscriber blocking

- Temporary access blocking
- Subscriptions preserved but viewing access blocked
- Used for non-payment, rule violations

enableSubscriber — subscriber unblocking

- Access restoration after blocking
- Subscriptions remain active

Subscription Operations

createPackageSubscriber — subscription creation

- Package connection to subscriber
- **Key billing operation** — paid period start
- Records subscription start date

deletePackageSubscriber — subscription deletion

- Package disconnection from subscriber
- **Key billing operation** — paid period end
- Used for cost calculation

Package Operations

createPackage — package creation

- New tariff plan created
- Audit for tracking product line changes

deletePackage — package deletion

- Package removed from system
- All subscriptions to it must be cancelled beforehand

5.2.3 Operation Structure

Main Fields

operationId — unique operation identifier

- Format: base64-encoded Snowflake ID
- Example: `oKl9SW3AAAE.`
- Generated automatically when creating record

type — operation type

- One of predefined types (see above)
- Used for filtering and grouping
- Example: `createPackageSubscriber`

portalId — portal identifier

- Which portal the operation relates to
- Used for data isolation between portals
- Example: `pKl9SW3AAAE.`

subscriberId — subscriber identifier (optional)

- Present in subscriber-related operations
- Null for package operations
- Example: `sKl9SW3AAAE.`

packageId — package identifier (optional)

- Present in package-related operations
- Null for subscriber create/delete operations
- Example: `pkKl9SW3AAAE.`

createdAt — operation creation time

- Format: ISO 8601 timestamp
- Example: `2024-10-16T10:00:00Z`
- **Key field for billing** — exact event date

updatedAt — last update time

- Usually matches createdAt
- May differ if operation was modified
- Example: `2024-10-16T10:00:00Z`

payload — additional operation data

- JSON object with operation details
- Content depends on operation type
- Examples:
- `{"subscriberId": "sKl9SW3AAAE.", "name": "John Doe"}`
- `{"packageId": "pkKl9SW3AAAE.", "packageName": "premium"}`

5.2.4 Getting Operations List

Basic Request

Get all operations:

```
curl -X GET https://your-catena-domain.com/tv-management/api/v1/operations \
  -H "X-Auth-Token: your-api-key"
```

Response:


```
{
  "operations": [
    {
      "operationId": "opKl9SW3AAAE.",
      "type": "createSubscriber",
      "portalId": "pKl9SW3AAAE.",
      "subscriberId": "sKl9SW3AAAE.",
      "packageId": null,
      "createdAt": "2024-10-16T10:00:00Z",
      "updatedAt": "2024-10-16T10:00:00Z",
      "payload": {
        "name": "John Doe",
        "phone": "+12345678901"
      }
    },
    {
      "operationId": "opKl9SW3AAAB.",
      "type": "createPackageSubscriber",
      "portalId": "pKl9SW3AAAE.",
      "subscriberId": "sKl9SW3AAAE.",
      "packageId": "pkKl9SW3AAAE.",
      "createdAt": "2024-10-16T10:05:00Z",
      "updatedAt": "2024-10-16T10:05:00Z",
      "payload": {
```

```
        "packageName": "premium"
      }
    }
  ],
  "next": "cursor-for-next-page"
}
```

5.2.5 Billing Calculations

Revenue Calculation for Period

Task: Calculate revenue for October 2024

Python example:

```
import requests
from datetime import datetime
from collections import defaultdict

API_URL = "https://catena.example.com/tv-management/api/v1"
API_KEY = "your-api-key"

# Package prices (store in your billing DB)
PACKAGE_PRICES = {
    "basicKl9SW3AAAE.": 10.0,      # $10/month
    "premiumKl9SW3AAAE.": 20.0,    # $20/month
    "sportKl9SW3AAAE.": 15.0       # $15/month
}

def calculate_monthly_revenue(year, month):
    """Calculate revenue for month based on operations"""

    start_date = f"{year}-{month:02d}-01"
    if month == 12:
        end_date = f"{year + 1}-01-01"
    else:
        end_date = f"{year}-{month + 1:02d}-01"

    # Get subscription creations
    subscriptions = get_operations(
        type="createPackageSubscriber",
        created_at_gte=start_date,
        created_at_lt=end_date
    )

    # Calculate revenue
    total_revenue = 0
    for op in subscriptions:
        package_id = op['packageId']
        price = PACKAGE_PRICES.get(package_id, 0)
        total_revenue += price

    return total_revenue

revenue = calculate_monthly_revenue(2024, 10)
print(f"Revenue for October 2024: ${revenue:.2f}")
```

5.2.6 Best Practices

Data Retention

Recommendations:

- **In Catena:** Store operations minimum 90 days
- **In billing DB:** Store forever for tax reporting
- **Archiving:** Export old operations to cold storage (S3, glacier)

Duplicate Prevention

Use idempotent processing to avoid duplicate billing

5.2.7 See Also

- **Subscriber Management** — subscriber creation records operations
- **Subscription Management** — subscription create/delete

- **Package Management** — package operations
- **Play Sessions** — additional data for traffic billing

6. Client Applications

6.1 Catena Android App User Guide

Welcome to the Catena app for watching television on Android!

6.1.1 Table of Contents

6.1.2 Installation

System Requirements

- **Operating System:** Android 5.0 (Lollipop) or higher
- **Minimum RAM:** 2 GB
- **Free Space:** 50 MB
- **Internet Connection:** Wi-Fi or mobile network (recommended 5 Mbps+)

Install from Google Play Store

1. **Open Google Play Store** on your Android device
2. Search for **"Catena"** or **"com.flussonic.catena"**
3. Find **Catena** app by Flussonic developer
4. Tap **"Install"** button
5. Wait for installation to complete
6. Tap **"Open"** to launch the app

Direct link: Catena on Google Play Store

Install from APK File

If the app is not available in Google Play Store in your region:

1. Download APK file from official website or get it from your service administrator
2. In Android settings, enable **"Install from unknown sources"**
3. Path: Settings → Security → Unknown sources
4. Open downloaded APK file
5. Tap **"Install"**
6. Launch the app after installation

Important: Only install APK from trusted sources!

6.1.3 Logging In

First Launch

On first launch, you need to log in using your phone number.

STEP 1: ENTER PHONE NUMBER

1. Launch Catena app
2. Select **country code** from dropdown (e.g., "+1" for USA)
3. Enter **phone number** without country code
4. Example: for number +1 234 567-8901 enter `2345678901`
5. Tap **"Send Code"** button (or press Enter on keyboard)

STEP 2: ENTER SMS CODE

1. You will receive SMS with **4-digit code**
2. Enter this code in "SMS code" field
3. Code is verified automatically when entering 4th digit
4. Or tap **"Confirm"** button

SUCCESSFUL LOGIN

After successful login: - Your access token is saved automatically - Next time you launch, login happens automatically - You will arrive at main screen with channel list

Logging Out

To log out of the app: 1. On main screen, tap **"Logout"** button in top-right corner 2. You will return to login screen

6.1.4 Watching Channels

Channel Selection

On main screen you will see **list of all available TV channels**.

For each channel displayed: - **Channel name** - **Current program** (what's on now) - **Next program** (what's coming up)

START WATCHING

1. Find channel you're interested in the list
2. **Tap on channel** - playback screen will open
3. Video will start playing automatically

Refresh Channel List

To update channel and program information: - **Pull screen down** (swipe down) on main screen - Channel list and programs will refresh

6.1.5 Video Playback

Playback Controls

In portrait orientation (vertical): - **Tap on screen** - playback controls will appear - Standard buttons available: - Pause/Play - Rewind - Fast forward

In landscape orientation (horizontal): - Automatically switches to **fullscreen mode** - System panels hide for maximum comfort - Tap on screen to show controls - Controls automatically hide after 3 seconds

Channel Switching

METHOD 1: SWIPE UP/DOWN (IN PORTRAIT MODE)

1. **Swipe up** (swipe finger up on screen) → switch to next channel
2. **Swipe down** (swipe finger down on screen) → switch to previous channel

When starting swipe you will see: - Top: previous channel name - Bottom: next channel name

METHOD 2: "NEXT CHANNEL" BUTTON

- At bottom of playback screen there's a **"Switch to [channel name]"** button
- Tap it to switch to next channel

Screen Rotation

- **Portrait mode**: Video player occupies top part of screen, program schedule shown below
- **Landscape mode**: Fullscreen video playback, all interface elements hide

---

6.1.6 Viewing Archive

What is Archive?

Archive allows viewing programs that already aired. You can watch shows that were broadcast earlier.

How to Open Program Guide (EPG)

On playback screen **in portrait mode**: - Scroll down - there you'll see **program list** for current channel - List shows programs: - Already aired (past) - Currently airing (highlighted) - Scheduled for later (future)

Watch Program from Archive

1. On playback screen scroll down to **program list**
2. Find show you want to watch
3. **Tap on program** from list
4. Program will start playing from beginning

Program Information

For each program displayed: - **Start time** and **end time** - **Program title** - **Description** (if available) - Current program is **highlighted** visually

Automatic Switching

- When one program ends, **next program** automatically starts
- You can watch channel continuously, programs will switch automatically
- On automatic switch, next program starts from beginning

Current Playback Time

Top of screen shows: - **Exact playback time** - which moment of recording you're watching - This helps navigate when viewing archive

---

6.1.7 Useful Tips

For Smartphones

1. **Rotate phone horizontally** for fullscreen viewing
2. **Pull down** on main screen to refresh channel list
3. **Swipe up/down** on player for quick channel switching

Auto Login

- After first successful login, token is saved
- Next time you launch app, login happens **automatically**
- You don't need to enter phone number and code each time

Playback Quality

- App automatically adjusts quality to your internet speed
- For best quality use Wi-Fi connection

Gesture Controls

- **Single tap on screen** - show/hide controls
- **Swipe up** - next channel
- **Swipe down** - previous channel

6.1.8 Troubleshooting

SMS Code Not Received

1. Check entered phone number is correct
2. Make sure correct country code is selected
3. Wait 1-2 minutes - sometimes SMS arrives with delay
4. Try requesting code again

Error "sms code doesn't match"

- Check entered code from SMS is correct
- Code valid for limited time - request new code if much time passed

Video Not Playing

1. Check internet connection
2. Try switching to another channel
3. Close and reopen app
4. Ensure you have access to selected channel

Channel List Empty

1. Check internet connection
2. Pull screen down to refresh
3. Log out and log in again

4. Contact administrator - perhaps no channels configured for your account

Video Stutters or Buffers

1. Check internet connection speed
2. Connect to Wi-Fi instead of mobile internet
3. Close other apps using internet
4. Try viewing at different time of day

6.1.9 Support

If you have issues with the app not described in this guide: - Contact your TV service administrator - Provide exact problem description - Report your device model and Android version

Enjoy watching!