Serviio API documentation

API doc version 1.4

Serviio version 1.0

Table of Contents

1 Preface

1.1 Scope

Serviio server includes a simple RESTful interface that enables third-party components to access functionality of the server component. This document is the official reference for that functionality.

1.2 Intended Audience

This document should be used by third-party developers who wish to integrate their products with Serviio server component.

1.3 Revision history

Date	Version	Description
15/01/2011	1.0	Initial version for Serviio 0.5
28/01/2011	1.1	Added advertiseService operation; added properties retrieveOriginalTitle (Metadata), showParentCategoryTitle (Presentation)
22/05/2011	1.2	Updated successful return code example; added Ping resource; added new reference data lookups; added online resource management support (Library); updated library status payload; added JSON support note
18/11/2011	1.3	Updated request result with parameters; updated library payloads; updated transcoding payloads
01/03/2012	1.4	Added CDS API; updated Application resource; added License management; added Remote Access settings management; added Access Groups management

2 Serviio REST interface

Serviio REST interface is powered by Restlet. It exposes server functionality to the clients via a combination of HTTP and XML or JSON protocols.

Currently there are two major APIs, the Configuration API (see chapter 3) and Content Directory Service API (CDS, see chapter 4). They both run on separate ports for better maintenance and security tuning.

It is recommended to only access Configuration API on the local network. CDS API is designed to be secure and can be accessed from outside of the LAN (assuming successful port forwarding on the internet gateway).

2.1 XML or JSON

Serviio API supports both, XML and JSON languages for the REST protocol via content negotiation mechanism. XML is the default format (and will be used as a reference in the rest of this document) but the client should still specify **Accept** HTTP header with value **application/xml**. JSON format will be used with the header's value of **application/json**. The same values should be used for **Content-Type** HTTP header when submitting data in the request body (POST and PUT methods).

A JSON response example:

```
{
    "libraryUpdatesCheckerRunning": false,
    "libraryAdditionsCheckerRunning": false,
    "numberOfAddedFiles": 0
}
```

2.2 Handling returned values and error codes

Serviio always returns 200 OK HTTP status code and for PUT/POST methods **result** element in the response body, which includes **httpCode** (e.g. 400) and (optionally) **errorCode** (application specific code). Error result can also specify one or more 'parameters', which further specify what went wrong.

An error response with application error code 502 and http code 400 (Bad Request):

```
HTTP/1.1 200 OK

<result>
      <errorCode>502</errorCode>
      <httpCode>400</httpCode>
      <parameter>{param_value1}</parameter>
      <parameter>{param_value2}</parameter>
      ...
</result>
```

A successful response:

```
HTTP/1.1 200 OK
```

```
<result>
	<errorCode>0</errorCode>
	<httpCode>200</httpCode>
</result>
```

3 Configuration API

Configuration API is used to control a running instance of the Serviio server. It is designed for usage on a local network where security is not a concern. It is split into several resources, each representing a functional group of data.

The RESTful interface is (by default) bound to all IP addresses on the server machine and port **23423**, for example localhost:23423.

3.1 Checking if the API is available

There is a simple 'ping' resource which can be used for the clients to ensure the server API is currently running.

HTTP request:

```
GET /rest/ping
```

HTTP response:

```
<result>
      <errorCode>0</errorCode>
      <httpCode>200</httpCode>
</result>
```

3.2 Retrieving application information

This is useful to retrieve information related to the whole application, e.g. latest version that has been released, current version, edition and license information.

HTTP request:

```
GET /rest/application
```

HTTP response:

```
<application>
      <version>0.5</version>
      <updateVersionAvailable>0.5.1</updateVersionAvailable>
      <edition>PRO</edition>
      <license>
             <id>11111111-1111-1111-1111-111111111114</id>
             <type>UNLIMITED</type>
             <name>John Smith</name>
             <email>john.smith@provider.com</email>
             <expiresInMinutes>166764</expiresInMinutes>
    </license>
</application>
```

where:

- **version** is Serviio application version

- **updateVersionAvailable** includes an update version, if available; optional
- **edition** specifies current Serviio version (FREE, PRO)
- **license** (optional) keeps information about the current license and includes
    - **id** – license ID; optional
    - **type** – type of license (NORMAL, UNLIMITED, BETA, EVALUATION)
    - **name** – name of the customer; optional
    - **email** – email address of the customer; optional
    - **expiresInMinutes** – number of minutes left before the license expires; optional

3.3 Retrieving status of Serviio service

Retrieves status of the server (started or stopped).

HTTP request:

```
GET /rest/service-status
```

HTTP response:

```
<serviceStatus>
      <serviceStarted>true</serviceStarted>
</serviceStatus>
```

where:

- **serviceStarted** is true is Serviio server has successfully started, profiles are loaded, database initialized and the web server is ready to serve other API calls

3.4 Retrieving status of library updates

Specifies whether Serviio looks for new files to index and/or updates to currently indexed files.

HTTP request:

```
GET /rest/library-status
```

HTTP response:

```
<libraryStatus>
      <libraryUpdatesCheckerRunning>false</libraryUpdatesCheckerRunning>
      <libraryAdditionsCheckerRunning>false</libraryAdditionsCheckerRunning>
      <lastAddedFileName>filename.avi</lastAddedFileName>
      <numberOfAddedFiles>3</numberOfAddedFiles>
</libraryStatus>
```

where:

- **libraryUpdatesCheckerRunning** states whether thread checking for currently shared files' updates is running
- **libraryAdditionsCheckerRunning** states whether thread checking for new/removed files is running
- **lastAddedFileName** (optional) includes name of the last found file from the current or last run search cycle
- **numberOfAddedFiles** includes number of new files that have been added during the current or last run search cycle

3.5 Retrieving reference data

Returns data that can be used in lists, combo boxes, etc.

HTTP request:

```
GET /rest/refdata/{name_of_property}
```

where {name_of_property} currently can be:

- cpu-cores
- profiles
- metadataLanguages
- browsingCategoriesLanguages
- descriptiveMetadataExtractors
- categoryVisibilityTypes
- onlineRepositoryTypes
- onlineContentQualities
- accessGroups
- remoteDeliveryQualities

HTTP response:

```
<refdata>
	<values>
		<item>
			<name>6</name>
			<value>DirecTV HD-DVR</value>
		</item>
		<item>
			<name>1</name>
			<value>Generic DLNA profile</value>
		</item>
		…
	</values>
</refdata>
```

where:

- item's **name** is a property identifier
- **value** is a string representation of the value.

For multilingual support, the client should not use **value** strings, but retrieve a particular language version by the key (**name**) from its own repository.

3.6 Invoking bespoke actions

This serves to invoke bespoke actions. Posting the request with a XML body will invoke the action on the server and return a result.

HTTP request:

```
POST /rest/action
```

HTTP request body:

```
<action>
	<name>{action_name}</name>
	<parameter>{param_value1}</parameter>
	<parameter>{param_value2}</parameter>
	...
</action>
```

where {action_name} can be:

- **forceVideoFilesMetadataUpdate** – forces update of metadata for all video files (for enabled folders)
- **forceLibraryRefresh** – forces immediate library refresh (for enabled folders)
- **forceOnlineResourceRefresh** – forces refresh of requested online repository; requires 1 parameter (id of online repository to be refreshed)
- **startServer** – starts UPnP server
- **stopServer** – stops UPnP server
- **exitServiio** – exits Serviio server process
- **advertiseService** – sends SSDP alive messages
- **checkStreamUrl** – verifies live stream URL; requires 2 parameters(stream type (AUDIO,VIDEO), stream URL)

Most of the above actions do not need any parameters (unless specified otherwise) and therefore the parameters element can be omitted.

HTTP response:

```
HTTP/1.1 200 OK

<result>
	<errorCode>{error_code}</errorCode>
	<httpCode>{200|4xx|5xx}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful
- 602 – error during advertising the service
- 603 – live stream is not valid or currently available (see checkStreamUrl)
- 700 – invalid parameter

3.7 Managing Status data (Status tab)

This resource manages Status data of the server. It is used to retrieve existing values and update server with new values. It includes information about current UPnP server status, list of registered renderers and bound IP address.

3.7.1 Retrieving Status data

HTTP request:

```
GET /rest/status
```

HTTP response:

```
<status>
	<serverStatus>STARTED</serverStatus>
	<renderers>
		<renderer>
			<uuid>fd189287-8b6b-4bc3-a4a0-7e70dc3a9c1a</uuid>
			<ipAddress>192.168.0.15</ipAddress>
			<name>Unrecognized device</name>
			<profileId>1</profileId>
			<status>UNKNOWN</status>
			<enabled>true</enabled>
			<accessGroupId>1</accessGroupId>
		</renderer>
		...
	</renderers>
	<boundIPAddress>192.168.0.10</boundIPAddress>
</status>
```

where:

- **serverStatus** defines state of the UpnP server and can be:
  - INITIALIZING
  - STARTED
  - STOPPED
- **renderer** includes data about existing renderer devices. There can be 0 or more renderers.
  - Its **status** can be:
    - ACTIVE (device is currently connected)
    - INACTIVE (device is currently not connected)

        - UNKNOWN (connection status is unknown)
    - **profileId** includes ID of one of the supported profiles, list of which can be accessed as described in 3.5 (profiles)
    - **enabled** field specifies whether the renderer has access to the server
    - **accessGroupId** includes id of the access group assigned to this renderer, can include values as described in 3.5 (accessGroups). *When the Pro license expires this will always return value of '1'.*
- **boundIPAddress** (optional) includes IP address the UpnP server binds to

3.7.2 Updating Status data

HTTP request:

```
PUT /rest/status
```

HTTP request body:

```
<status>
      <renderers>
             <renderer>
                    <uuid>fd189287-8b6b-4bc3-a4a0-7e70dc3a9c1a</uuid>
                    <ipAddress>192.168.0.15</ipAddress>
                    <name>Renderer name</name>
                    <profileId>1</profileId>
                    <enabled>false</enabled>
                    <accessGroupId>1</accessGroupId>
             </renderer>
             ...
      </renderers>
      <boundIPAddress>192.168.0.10</boundIPAddress>
</status>
```

where:

- **renderers** must include a snapshot of renderers as they should be stored in the database. In a case a new renderer is added manually, a new UUID must be generated.
- **boundIPAddress** (optional) must include a valid IP address

*Please note that **accessGroupId** on the renderer element is only taken into account when using the Pro edition. In any other case it'll default to '1' (unrestricted access). It is advisable to not let the user change the access group in this case, to avoid confusion.*

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful

- 500 – invalid IP address
- 502 – the IP address is not unique (one of the provided renderers' addresses)

3.8 Managing Library (Library tab)

This resource manages shared folders and the ways files are added/removed and updated in the library. It allows adding/updating/removing folders (incl. their attributes) and setting preferences of how the library should be managed.

3.8.1 Retrieving Library data

HTTP request:

```
GET /rest/repository
```

HTTP response:

```
<repository>
	<sharedFolders>
		<sharedFolder>
			<id>40</id>
			<folderPath>D:\pictures</folderPath>
			<supportedFileTypes>
				<fileType>IMAGE</fileType>
			</supportedFileTypes>
			<descriptiveMetadataSupported>false</descriptiveMetadataSupported>
			<scanForUpdates>true</scanForUpdates>
			<accessGroupIds>
				<id>1</id>
				<id>2</id>
			</accessGroupIds>
		</sharedFolder>
		...
	</sharedFolders>
	<searchHiddenFiles>false</searchHiddenFiles>
	<searchForUpdates>true</searchForUpdates>
	<automaticLibraryUpdate>true</automaticLibraryUpdate>
	<automaticLibraryUpdateInterval>5</automaticLibraryUpdateInterval>
	<onlineRepositories>
		<onlineRepository>
			<id>1</id>
			<repositoryType>FEED</repositoryType>
			<contentUrl>http://someUrl</contentUrl>
			<fileType>IMAGE</fileType>
			<thumbnailUrl/>
			<repositoryName>feed name</repositoryName>
			<enabled>true</enabled>
			<accessGroupIds>
				<id>1</id>
				<id>2</id>
			</accessGroupIds>
		</onlineRepository>
		...
	</onlineRepositories>
	<maxNumberOfItemsForOnlineFeeds>10</maxNumberOfItemsForOnlineFeeds>
	<onlineFeedExpiryInterval>24</onlineFeedExpiryInterval>
	<onlineContentPreferredQuality>HIGH</onlineContentPreferredQuality>
</repository>
```

where:

- **sharedFolders** includes a list of sharedFolder elements

- **sharedFolder** represents a folder on the given path and
    - **supportedFileTypes** can include zero or more of **fileType** elements with values of
        - IMAGE
        - AUDIO
        - VIDEO
    - **descriptiveMetadataSupported** states whether descriptive metadata will be extracted for files in that folder
    - **scanForUpdates** states whether the folder should be scanned for new/updated/removed files (after the initial scan)
    - **accessGroupIds** includes a list of id elements representing access groups (as defined in 3.5 (accessGroups)) that have access to this repository. There will always be at least one id (with value '1'). *When the Pro license expires this list will always include only single value ('1').*
- **searchHiddenFiles** defines if hidden files will be added to the library
- **searchForUpdates** states whether Serviio should look for updates of currently shared files
- **automaticLibraryUpdate** controls turning on/off automatic library updates (for folders that enable it)
- **automaticLibraryUpdateInterval** includes time in minutes that must elapse before a new scan starts
- **onlineRepositories** includes a list of onlineRepository elements
- **onlineRepository** represents an online resource on the given URL and
    - **repositoryType** can include values of (as defined in 3.5 (onlineRepositoryTypes)):
        - FEED (for Atom/RSS based feeds)
        - LIVE_STREAM (for live streams)
        - WEB_RESOURCE (for other web resources)
    - **fileType** can include values of
        - IMAGE
        - AUDIO
        - VIDEO
    - **thumbnailUrl** (optional) includes address of a thumbnail image to be used when browsing for the resource (only for certain repositoryTypes)
    - **repositoryName** (optional) includes user-defined display name of the online resource
    - **enabled** specifies whether the resource will be parsed and available for browsing
    - **accessGroupIds** has the same meaning like accessGroupIds on **sharedFolder** above
- **maxNumberOfItemsForOnlineFeeds** includes number of feed items to retrieve from the online resource or -1 for all the items

- **onlineFeedExpiryInterval** gives number or hours the feed data should be kept in cache
- **onlineContentPreferredQuality** can include one of (as defined in 3.5 (onlineContentQualities)):
  - LOW
  - MEDIUM
  - HIGH

3.8.2 Updating Library data

HTTP request:

```
PUT /rest/repository
```

HTTP request body:

```
<repository>
	<sharedFolders>
		<sharedFolder>
			<id>40</id>
			<folderPath>D:\pictures</folderPath>
			<supportedFileTypes>
				<fileType>IMAGE</fileType>
			</supportedFileTypes>
			<descriptiveMetadataSupported>false</descriptiveMetadataSupported>
			<scanForUpdates>true</scanForUpdates>
			<accessGroupIds>
				<id>1</id>
				<id>2</id>
			</accessGroupIds>
		</sharedFolder>
		<sharedFolder>
			<folderPath>D:\new folder</folderPath>
			<supportedFileTypes>
				<fileType>AUDIO</fileType>
				<fileType>IMAGE</fileType>
			</supportedFileTypes>
			<descriptiveMetadataSupported>false</descriptiveMetadataSupported>
			<scanForUpdates>true</scanForUpdates>
			<accessGroupIds>
				<id>1</id>
			</accessGroupIds>
		</sharedFolder>
		...
	</sharedFolders>
	<searchHiddenFiles>false</searchHiddenFiles>
	<searchForUpdates>true</searchForUpdates>
	<automaticLibraryUpdate>true</automaticLibraryUpdate>
	<automaticLibraryUpdateInterval>5</automaticLibraryUpdateInterval>
	<onlineRepositories>
		<onlineRepository>
			<id>1</id>
			<repositoryType>FEED</repositoryType>
			<contentUrl>http://someUpdatedUrl</contentUrl>
			<fileType>IMAGE</fileType>
			<thumbnailUrl/>
			<enabled>true</enabled>
			<accessGroupIds>
				<id>1</id>
			</accessGroupIds>
		</onlineRepository>
		<onlineRepository>
			<repositoryType>LIVE_STREAM</repositoryType>
			<contentUrl>mmsh://someNewUrl</contentUrl>
```

```
                        <fileType>AUDIO</fileType>
                        <thumbnailUrl>http://thunmbnail.url</thumbnailUrl>
                        <repositoryName>test audio stream</repositoryName>
                        <enabled>false</enabled>
                        <accessGroupIds>
                                <id>1</id>
                                <id>2</id>
                        </accessGroupIds>
                </onlineRepository>
                ...
        </onlineRepositories>
        <maxNumberOfItemsForOnlineFeeds>10</maxNumberOfItemsForOnlineFeeds>
        <onlineFeedExpiryInterval>24</onlineFeedExpiryInterval>
        <onlineContentPreferredQuality>LOW</onlineContentPreferredQuality>
</repository>
```

where:

- **sharedFolders** must include a snapshot of folders as they should be stored in the database (incl. additions and removals and updates). In a case a new folder is added, its **id** attribute must not be provided.
- **onlineRepositories** must include a snapshot of online resources as they should be stored in the database (incl. additions and removals and updates). In a case a new resource is added, its **id** attribute must not be provided.
- values of the other elements are described in 3.8.1.
- **accessGroupIds** elements (on both, sharedFolders and onlineRepository) must include at least one value ('1'), if not '1' will be stored anyway.

*Please note that **accessGroupIds** element is only taken into account when using the Pro edition. In any other case it'll default to a list with single value '1' (unrestricted access). It is advisable to not let the user change the access groups in this case, to avoid confusion.*

HTTP response:

```
HTTP/1.1 200 OK

<result>
        <errorCode>{error_code}</errorCode>
        <httpCode>{200/500/400}</httpCode>
        <parameter>{param_value}</parameter>
</result>
```

where {error_code} can be:

- 0 – successful
- 503 – invalid online resource URL

and {param_value} can be:

- invalid online resource URL value

3.9 Managing Metadata (Metadata tab)

This resource is used to control metadata extraction for different file type. It enables to pick metadata extractors and configure them.

3.9.1 Retrieving Metadata data

HTTP request:

```
GET /rest/metadata
```

HTTP response:

```
<metadata>
      <audioLocalArtExtractorEnabled>true</audioLocalArtExtractorEnabled>
      <videoLocalArtExtractorEnabled>true</videoLocalArtExtractorEnabled>
      <videoOnlineArtExtractorEnabled>true</videoOnlineArtExtractorEnabled>
      <videoGenerateLocalThumbnailEnabled>true</videoGenerateLocalThumbnailEnabled>
      <metadataLanguage>fr</metadataLanguage>
      <retrieveOriginalTitle>true</retrieveOriginalTitle>
      <descriptiveMetadataExtractor>ONLINE_VIDEO_SOURCES</descriptiveMetadataExtractor>
</metadata>
```

where:

- **audioLocalArtExtractorEnabled** states whether to look for local audio covers
- **videoLocalArtExtractorEnabled** states whether to look for local video posters
- **videoGenerateLocalThumbnailEnabled** states whether to generate thumbnails from the video files
- **metadataLanguage** includes 2-digit ISO 639-1 code of preferred language of retrieved descriptive metadata as specified in 3.5 (metadataLanguages)
- **descriptiveMetadataExtractor** defines type of descriptive metadata extractor and can include one of values specified in 3.5 (descriptiveMetadataExtractors)
- **videoOnlineArtExtractorEnabled** specifies whether the video poster should be downloaded (in a case of onlime metadata extractor is selected)

3.9.2 Updating Metadata data

HTTP request:

```
PUT /rest/metadata
```

HTTP request body:

```
<metadata>
      <audioLocalArtExtractorEnabled>true</audioLocalArtExtractorEnabled>
      <videoLocalArtExtractorEnabled>true</videoLocalArtExtractorEnabled>
      <videoOnlineArtExtractorEnabled>true</videoOnlineArtExtractorEnabled>
      <videoGenerateLocalThumbnailEnabled>true</videoGenerateLocalThumbnailEnabled>
      <metadataLanguage>fr</metadataLanguage>
      <retrieveOriginalTitle>false</retrieveOriginalTitle>
      <descriptionMetadataExtractor>NONE</descriptionMetadataExtractor>
</metadata>
```

where values of the elements are described in 3.9.1.

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful

3.10 Managing Transcoding (Transcoding tab)

This resource is used to configure transcoding parameters.

3.10.1 Retrieving Transcoding data

HTTP request:

```
GET /rest/transcoding
```

HTTP response:

```
<transcoding>
      <audioDownmixing>true</audioDownmixing>
      <threadsNumber>1</threadsNumber>
      <transcodingFolderLocation>d:\</transcodingFolderLocation>
      <bestVideoQuality>true</bestVideoQuality>
      <transcodingEnabled>true</transcodingEnabled>
</transcoding>
```

where:

- **transcodingEnabled** states whether transcoding is enabled in Serviio
- **transcodingFolderLocation** includes file path of a folder for storing temporary transcoded files
- **threadsNumber** includes number of CPU cores to use for transcoding jobs
- **audioDownmixing** states whether audio should be downmixed to stereo(true) or kept with original channels layout (false)
- **bestVideoQuality** indicates whether to use the best quality when transcoding video streams. It might affect CPU usage.

3.10.2 Updating Transcoding data

HTTP request:

```
PUT /rest/transcoding
```

HTTP request body:

```
<transcoding>
	<audioDownmixing>true</audioDownmixing>
	<threadsNumber>2</threadsNumber>
	<transcodingFolderLocation>d:\</transcodingFolderLocation>
	<bestVideoQuality>true</bestVideoQuality>
	<transcodingEnabled>true</transcodingEnabled>
</transcoding>
```

where:

- maximum value for **threadsNumber** can be retrieved as described in 3.5 (cpu-cores)
- the rest of the elements is described in 3.10.1
- in a case where **transcodingEnabled** is *false* all the other elements are optional as they won't be persisted

HTTP response:

```
HTTP/1.1 200 OK

<result>
	<errorCode>{error_code}</errorCode>
	<httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful
- 501 – transcoding folder doesn't exist or cannot be written to

3.11 Managing Presentation (Presentation tab)

This resource includes values that describe how Serviio presents media files and folders on the device. It enables to manage browsing categories menu and to select preferred language of the menu.

3.11.1 Retrieving Presentation data

HTTP request:

```
GET /rest/presentation
```

HTTP response:

```
<presentation>
	<categories>
		<browsingCategory>
			<id>A</id>
			<title>Audio</title>
			<visibility>DISPLAYED</visibility>
			<subCategories>
				<browsingCategory>
					<id>A_A</id>
```

```
                                    <title>Artists</title>
                                    <visibility>DISPLAYED</visibility>
                                    <subCategories/>
                              </browsingCategory>
                              ...
                        </subCategories>
                  </browsingCategory>
                  ...
      </categories>
      <language>en</language>
      <showParentCategoryTitle>true</showParentCategoryTitle>
</presentation>
```

where:

- **categories** includes one or more **browsingCategory** elements which represents the root categories of the menu and their:
  - **visibility** can be one of (as defined in 3.5 (categoryVisibilityTypes)):
    - DISPLAYED
    - CONTENT_DISPLAYED
    - DISABLED
  - **subCategories** include zero or more **browsingCategory** elements representing it's sub-categories
- **language** includes 2-digit ISO 639-1 language code of preferred menu language, as defined in 3.5 (browsingCategoriesLanguages)
- **showParentCategoryTitle** is a switch deciding whether the name of a parent category will be included in item names (in the case of a 'CONTENT_DISPLAYED' category's items)

3.11.2 Updating Presentation data

HTTP request:

```
PUT /rest/transcoding
```

HTTP request body:

```
<presentation>
      <categories>
            <browsingCategory>
                  <id>A</id>
                  <visibility>DISPLAYED</visibility>
                  <subCategories>
                        <browsingCategory>
                              <id>A_A</id>
                              <visibility>DISPLAYED</visibility>
                              <subCategories/>
                        </browsingCategory>
                        ...
                  </subCategories>
            </browsingCategory>
            ...
      </categories>
      <language>en</language>
      <showParentCategoryTitle>true</showParentCategoryTitle>
</presentation>
```

where:

- **categories** must include the exact snapshot of existing categories including their id, visibility and (if applicable) subcategories
- **language** includes ISO 639-1 code of preferred menu language
- **showParentCategoryTitle** defines item names for CONTENT_DISPLAYED categories (see 3.11.1)

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful

3.12 Managing Remote Access settings

This resource is used to manage settings of remote access.

3.12.1 Retrieving Remote Access settings data

HTTP request:

```
GET /rest/remote-access
```

HTTP response:

```
<remoteAccess>
      <remoteUserPassword>password</remoteUserPassword>
      <preferredRemoteDeliveryQuality>ORIGINAL</preferredRemoteDeliveryQuality>
</remoteAccess>
```

where:

- **remoteUserPassword** is password the user uses for accessing Serviio remotely, e.g. for the CDS service (see 4.3)
- **preferredRemoteDeliveryQuality** is the user's preferred delivery quality for the remote access (can be one of remoteDeliveryQualities as described in 3.5)

3.12.2 Updating Remote Access settings

This functionality is only available for users with Pro edition (otherwise error 554 is returned).

HTTP request:

```
PUT /rest/remote-access
```

HTTP request body:

```
<remoteAccess>
      <remoteUserPassword>password</remoteUserPassword>
      <preferredRemoteDeliveryQuality>LOW</preferredRemoteDeliveryQuality>
</remoteAccess>
```

where:

- **remoteUserPassword** is password the user uses for accessing Serviio remotely, must not be null or empty
- the other elements are mandatory and have the values as described in 3.12.1

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful
- 504 – provided **remoteUserPassword** is empty
- 554 – invalid edition of Serviio, functionality not available for this edition

3.13 Managing console settings (Console settings tab)

This resource is at the moment tightly coupled with Serviio console and might be deprecated in future.

3.13.1 Retrieving Console settings data

HTTP request:

```
GET /rest/console-settings
```

HTTP response:

```
<consoleSettings>
      <language>en</language>
      <securityPin>1234</securityPin>
      <checkForUpdates>true</checkForUpdates>
</consoleSettings>
```

where:

- **language** is optional element and includes 2-digit ISO 639-1 language code the console should use for localization. If the element is not provided, the default language should be used.
- **securityPin** is optional element and is used to store security PIN that may be used in the API's client application
- **checkForUpdates** is a flag storing the user's preference of being notified when there is a new version of Serviio available (i.e. when element *updateVersionAvailable* is present in application information as per 3.2)

3.13.2 Updating Console settings data

HTTP request:

```
PUT /rest/console-settings
```

HTTP request body:

```
<consoleSettings>
      <language>de</language>
      <securityPin>1234</securityPin>
      <checkForUpdates>false</checkForUpdates>
</consoleSettings>
```

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful

3.14 Uploading customer license file

Serviio comes with a bundled evaluation license that expires after a time. In order for the customer to keep the Pro version features enabled he/she has to provide a purchased license file.

To upload the file a request has to be made with *Content-Type* header of **plain/text**.

HTTP request:

```
PUT /rest/license-upload
```

HTTP request body (example):

```
#Sat Mar 31 13:05:41 BST 2012
```

```
property_version=1.0
property_edition=PRO
property_type=UNLIMITED
startDate=1333195541877
signature=302d02150083c3bfc00f926450a52406968502141e05a628ea36bdf3c36145798c271a60abfbbd3d
property_id=11111111-1111-1111-1111-111111111114
creationDate=1333195541879
property_name=John Smith
version=2
property_email=john.smith@provider.com
```

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400/415}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful
- 555 – license is invalid

4 Content Directory Service API

Content Directory Service (CDS) represents an access to Serviio's media library. It allows to browse for content in a similar way the UPnP component of Serviio does, as well as retrieving the media items. This API is secure and requires authentication, thus is suitable for accessing from both, local and remote networks.

The API is bound (by default) to port **23424** on all the IP addresses of the Serviio machine.

The functionality provided by this API is only enabled in Serviio Pro edition.

4.1 Checking if the API is available

There is a simple 'ping' resource which can be used for the clients to ensure the server API is currently running. Further details are available in 3.1.

HTTP request:

```
GET /cds/ping
```

4.2 Retrieving application information

This is useful to retrieve information related to the whole application, e.g. latest version that has been released. Further details are available in 3.2.

HTTP request:

```
GET /cds/application
```

4.3 Logging in the CDS

Before accessing any other CDS resource the client needs to pass authentication process. That is based on passing custom headers to a login endpoint which will respond with an **authentication token**. This token will then be used during accessing the other CDS resources.

Note that the user has to have a non-empty password set up first.

HTTP request:

```
POST /cds/login
```

4.3.1 Overview of the Authentication Process

The CDS REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. The following steps describe the basic process for authentication.

1. You create a string based on specific information in the request. For more information, see 4.3.2.

2. You calculate a signature using the user's password, the string from step 1, and an HMAC-SHA1 algorithm. Informally, we call this process signing the request, and we call the output of the HMAC algorithm the signature because it simulates the security properties of a real signature. For instructions on creating the signature, see 4.3.3.
3. You include the signature in the login request and send the request to the API. For information about where to put the signature in the request, see 4.3.4.
4. We check your signature. When the request is received, we fetch the user password that you claim to have and use it in the same way you did to compute a signature for the message. We then compare the signature we calculated to the signature you presented in the request. If the two signatures match, we accept and process the request. Otherwise, we reject the request and respond with an error message.

4.3.2 The string to sign

In the first task in the preceding process, you form a string. The string is simply the UTF-8 encoded value of the Date header in the request (e.g., Thu, 19 Nov 2009 19:37:58 GMT). Your request must include either the **Date** header, the **X-Serviio-Date** header, or both (if both are present, we ignore the Date header when authenticating the request). You might decide to include the X-Serviio-Date header if your HTTP client doesn't let you set the Date header.

The format you use for the header value must be one of the full date formats specified in RFC 2616, section 3.3.1, for example, Wed, 05 Apr 2006 21:12:00 GMT.

4.3.3 Calculating the Signature

Calculating the value to include in the request is a simple procedure.

1. Calculate an RFC 2104-compliant HMAC-SHA1 hash, using the string (see 4.3.2) and the user's password as the key.
2. Convert the resulting value to base64.
3. The result is the signature you include in the request.

The following table shows a string, a fake Secret Access Key, and what the resulting base64 encoded signature would be.

String	Thu, 14 Aug 2008 17:08:48 GMT
Password	password
Base64 encoded signature	HKS3OvMF5qkM1BulBhAukntIGZU=

4.3.4 The Authorization Header

To pass the signature to Serviio, you include it as part of the standard HTTP **Authorization** header using the following format:

```
Authorization: Serviio <signature>
```

Note that there is a space after the Serviio.

An example HTTP request:

```
POST /cds/login HTTP/1.1
Host: localhost:23424
Date: Thu, 14 Aug 2008 17:08:48 GMT
X-Serviio-Date: Thu, 14 Aug 2008 17:08:48 GMT
Authorization: Serviio HKS3OvMF5qkM1BulBhAukntIGZU=
```

4.3.5 Authentication responses

Successful HTTP response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" ?>
<result>
    <errorCode>0</errorCode>
    <httpCode>200</httpCode>
    <parameter>5e2872a72d7d4f6ab9f5457594a72c9d</parameter>
</result>
```

where **parameter** includes the **authentication token** to be used throughout this logged-in session.

Error HTTP response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" ?>
<result>
    <errorCode>{error_code}</errorCode>
    <httpCode>401</httpCode>
</result>
```

where {error_code} can be:

- 550 – missing Date or X-Serviio-Date header
- 551 – missing Authorization header
- 552 – Authorization header has invalid value (possibly wrong password)
- 554 – invalid edition of Serviio, functionality not available for this edition
- 556 – the user has not set up their password yet or it is empty
- 557 – the server has been stopped

4.4 Logging out of CDS

Use this resource to log out of the server and dispose of the authentication token.

HTTP request:

```
POST /cds/logout?authToken=<token>
```

HTTP response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{200/500/400}</httpCode>
</result>
```

where {error_code} can be:

- 0 – successful
- 553 – missing or invalid authentication token
- 554 – invalid edition of Serviio, functionality not available for this edition
- 557 – the server has been stopped

4.5 Browsing Content Directory

Content Directory navigation is enabled by the browse method. It enables retrieving media containers and items from the Serviio library, similarly to how UPnP clients do. Items include relative URLs for thumbnails, subtitles and the content itself. The URL is relative to the CDS API host.

There may be **multiple content URLs** for an item, one for each delivery quality (as per 3.5 (remoteDeliveryQualities)). There should, however, always be at least one for the *ORIGINAL* quality. User can define a preferred delivery quality and that is referenced to with **preferred** attribute of the content URL element (there will always be one present in the list of content URLs).

HTTP request:

```
GET
/cds/browse/<ProfileId>/<ObjectId>/<BrowseMethod>/<Filter>/<StartIndex>/<Requested
Count>?authToken=<token>
```

where:

- *ProfileId* – Id of renderer profile the client wants to use
- *ObjectId* - Identification of object (container or item) to retrieve. Use **0** for the root container ('home').
- *BrowseMethod* - can be either **BrowseMetadata** (to get details of a particular object) or **BrowseDirectChildren** (returns the contents of a container).
- *Filter* – defines what kinds of objects should be taken into account when making the request. Can be **items**, **containers** or **all**.
- *StartIndex* – defines start index of all the results to return when BrowseDirectChildren method is used. Starts with 0. Use **0** for BrowseMetadata method.
- *RequestedCount* – defines number of results to return when BrowseDirectChildren method is used. To retrieve all objects in the container use **0**. Use **1** for BrowseMetadata method.

HTTP response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="1" totalMatched="25">
    <objects>
        <object id="<ObjectId>" type="<type>" fileType="<fileType>" parentId="<ParentId>">
            ….
        </object>
        <object id="<ObjectId>" type="<type>" fileType="<fileType>" parentId="<ParentId>">
            ….
        </object>
    </objects>
</contentDirectory>
```

where:

- **returnedSize** includes number of objects returned by the request
- **totalMatched** includes total number of objects matched by the query (and not restricted by RequestedCount)
- **objects** includes a list of found **object** elements where:
  - **ObjectId** is a string value representing the unique identifier of the object
  - **ParentId** is a string value representing the ObjectId of a parent container
  - **type** is the object type and can be on of:
    - ITEM for media items
    - CONTAINER for object containers
  - **fileType** is type of the item and can be one of:
    - VIDEO
    - AUDIO
    - IMAGE

HTTP error response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{500/401/404}</httpCode>
</result>
```

where {error_code} can be:

- 553 – missing or invalid authentication token
- 554 – invalid edition of Serviio, functionality not available for this edition
- 557 – the server has been stopped
- 700 – if the request is not well formed

4.5.1 Browse example requests

HTTP request for root containers:

```
GET /cds/browse/flv_player/0/BrowseDirectChildren/containers/0/0
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="3" totalMatched="3">
    <objects>
        <object id="A" type="CONTAINER" childCount="11" parentId="0">
            <title>Audio</title>
        </object>
        <object id="I" type="CONTAINER" childCount="6" parentId="0">
            <title>Image</title>
        </object>
        <object id="V" type="CONTAINER" childCount="11" parentId="0">
            <title>Video</title>
        </object>
    </objects>
</contentDirectory>
```

HTTP request for description of container with id V

```
GET /cds/browse/flv_player/V/BrowseMetadata/containers/0/1
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="1" totalMatched="1">
    <objects>
        <object id="V" type="CONTAINER" childCount="11" parentId="0">
            <title>Video</title>
        </object>
    </objects>
</contentDirectory>
```

HTTP request for empty result set (there are no items in the container, only (possibly) other sub folders):

```
GET /cds/browse/flv_player/V/BrowseDirectChildren/items/0/10
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="0" totalMatched="0">
      <objects />
</contentDirectory>
```

HTTP request for 3 VIDEO items (from total 18) from container V_M, skipping the first 2:

```
GET /cds/browse/flv_player/V_M/BrowseDirectChildren/items/2/3
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="3" totalMatched="18">
    <objects>
        <object id="V_M^V_2045" type="ITEM" fileType="VIDEO" parentId="V_M">
            <title>Elite Squad: The Enemy Within</title>
            <description>After a bloody invasion ...</description>
            <genre>Action</genre>
            <date>2011-11-11</date>
            <duration>1</duration>
            <thumbnailUrl>/cds/resource/4296/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality="ORIGINAL"
resolution="362x200">/cds/resource/2108/MEDIA_ITEM/FLV*0/ORIGINAL</contentUrl>
                <contentUrl quality="MEDIUM" resolution="362x200"
preferred=”true”>/cds/resource/2108/MEDIA_ITEM/FLV*0/MEDIUM</contentUrl>
                <contentUrl quality="LOW"
resolution="362x200">/cds/resource/2108/MEDIA_ITEM/FLV*0/LOW</contentUrl>
            </contentUrls>
            <live>false</live>
            <onlineIdentifiers>
                <identifier type="IMDB">tt1022603</identifier>
                <identifier type="TMDB">19913</identifier>
            </onlineIdentifiers>
            <contentType>MOVIE</contentType>
        </object>
        <object id="V_M^V_2029" type="ITEM" fileType="VIDEO" parentId="V_M">
            <title>Female Agents 1</title>
            <description>May 1944, a group of French ...</description>
            <genre>Action</genre>
            <date>2008-02-08</date>
            <duration>3398</duration>
            <thumbnailUrl>/cds/resource/4326/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality=”ORIGINAL” resolution="362x200"
preferred=”true”>/cds/resource/2029/MEDIA_ITEM/FLV*0</contentUrl>
            </contentUrls>
            <subtitlesUrl>/cds/resource/2029/SUBTITLE.srt</subtitlesUrl>
            <live>false</live>
            <onlineIdentifiers>
                <identifier type="IMDB">tt1025501</identifier>
                <identifier type="TVDB">113</identifier>
            </onlineIdentifiers>
            <contentType>MOVIE</contentType>
        </object>
        <object id="V_M^V_2030" type="ITEM" fileType="VIDEO" parentId="V_M">
            <title>Female Agents 2</title>
            <description>May 1944, a group of French ...</description>
            <genre>Action</genre>
            <date>2008-02-08</date>
            <duration>3329</duration>
            <thumbnailUrl>/cds/resource/4327/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality=”ORIGINAL” resolution="362x200"
preferred=”true”>/cds/resource/2030/MEDIA_ITEM/FLV*0</contentUrl>
            </contentUrls>
            <subtitlesUrl>/cds/resource/2030/SUBTITLE.srt</subtitlesUrl>
            <live>false</live>
            <contentType>MOVIE</contentType>
        </object>
    </objects>
</contentDirectory>
```

where:

- **onlineIdentifiers** includes list of the content's identifiers in different online databases, can be imdb.com (IMDB), themoviedb.com (TMDB), thetvdb.com (TVDB)
- **contentType** represents type of the video content and can be MOVIE, EPSIODE or UNKNOWN (e.g. home videos, any video that has not been recognized as either a movie or

episode)

HTTP request for 2 AUDIO items (from total 5) from container A_R, skipping the first 1:

```
GET /cds/browse/flv_player/A_R/BrowseDirectChildren/items/1/2
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="2" totalMatched="5">
    <objects>
        <object id="A_R^S_2015" type="ITEM" fileType="AUDIO" parentId="A_R">
            <title>Alabina</title>
            <genre>Latin</genre>
            <date>2010-01-25</date>
            <artist>Alabina</artist>
            <album>Alabina Best of</album>
            <originalTrackNumber>1</originalTrackNumber>
            <duration>349</duration>
            <thumbnailUrl>/cds/resource/3533/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality="ORIGINAL"
preferred="true">/cds/resource/2100/MEDIA_ITEM/MP3*0/ORIGINAL</contentUrl>
            </contentUrls>
            <live>true</live>
        </object>
        <object id="A_R^S_2014" type="ITEM" fileType="AUDIO" parentId="A_R">
            <title>Perfect</title>
            <genre>Rock</genre>
            <date>2010-12-31</date>
            <artist>Alanis Morissette</artist>
            <album>Jagged Little Pill</album>
            <originalTrackNumber>3</originalTrackNumber>
            <duration>188</duration>
            <thumbnailUrl>/cds/resource/3532/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality="ORIGINAL"
preferred="true">/cds/resource/2101/MEDIA_ITEM/MP3*0/ORIGINAL</contentUrl>
            </contentUrls>
            <live>false</live>
        </object>
    </objects>
</contentDirectory>
```

HTTP request for all 2 IMAGE items from container I_Y^YEAR_2009:

```
GET /cds/browse/flv_player/I_Y^YEAR_2009/BrowseDirectChildren/items/0/0
```

HTTP response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<contentDirectory returnedSize="2" totalMatched="2">
    <objects>
        <object id="I_Y^YEAR_2009^I_2009" type="ITEM" fileType="IMAGE"
parentId="I_Y^YEAR_2009">
            <title>PICT0002</title>
            <date>2009-03-05</date>
            <thumbnailUrl>/cds/resource/3527/COVER_IMAGE</thumbnailUrl>
            <contentUrls>
                <contentUrl quality="ORIGINAL"
resolution="2560x1920">/cds/resource/2086/MEDIA_ITEM/JPEG_LRG*0/ORIGINAL</contentUrl>
                <contentUrl quality="MEDIUM"
resolution="1500x1000">/cds/resource/2086/MEDIA_ITEM/JPEG_MED*0/MEDIUM</contentUrl>
                <contentUrl quality="LOW"
resolution="500x350">/cds/resource/2086/MEDIA_ITEM/JPEG_SM*0/LOW</contentUrl>
            </contentUrls>
        </object>
        <object id="I_Y^YEAR_2009^I_2008" type="ITEM" fileType="IMAGE"
parentId="I_Y^YEAR_2009">
            <title>IMG_8981</title>
            <date>2009-08-28</date>
            <originalResolution>3456x2304</originalResolution>
            <thumbnailUrl>/cds/resource/3526/COVER_IMAGE</thumbnailUrl>
            <contentUrl quality=”ORIGINAL” resolution=”100x100”
preferred=”true”>/cds/resource/2008/MEDIA_ITEM/JPEG_LRG*0/ORIGINAL</contentUrl>
        </object>
    </objects>
</contentDirectory>
```

4.6 Retrieving media content

Content URL is returned as a part of browse method results. It can be either the media file content itself, thumbnail or subtitle file. Each of these needs authentication token added to the URL.

HTTP request:

```
/GET /cds/resource/<pathToResource>?authToken=<token>
```

HTTP Response:

```
HTTP/1.1 200 OK

<bytes of the content>
```

HTTP error response:

```
HTTP/1.1 200 OK

<result>
      <errorCode>{error_code}</errorCode>
      <httpCode>{500/401/404}</httpCode>
</result>
```

where {error_code} can be:

- 553 – missing or invalid authentication token
- 554 – invalid edition of Serviio, functionality not available for this edition
- 557 – the server has been stopped