Table of Contents

About This Guide 5
About Haivision 6
Audience 6
Reliability of Information 6
Obtaining Documentation 6
Related Documents 7
Service Support 7
Document Conventions 7

Version 6.5 Update 9

Chapter 1: Introduction
URIs for REST Resources 11
URI Structure 11
OAuth 12
Implementing OAuth 12
REST API Responses 15
Example Success Response 16
Example Error Response 17
Sorting Response Content 17
Diagnostics 18
XML Entities 19

Chapter 2: API Reference
Summary of Furnace API Resources 22
Demo Resources 23
Demo XML Entities 23
Demo API Endpoints 23
Asset Resources 26
Asset XML Entities 26
Asset API Endpoints 27
Client Resources 32
Client XML Entities 32
Client API Endpoints 33

Command Resources 37
Command XML Entities 37
Command API Endpoints 42
Program Resources 43
Program XML Entities 43
Program API Endpoints 43
Recording Resources 45
Recording XML Entities 45
Recording API Endpoints 49
Recorder Resources 58
Recorder XML Entities 58
Recorder API Endpoints 59
Station Resources 63
Station XML Entities 63
Station API Endpoints 65
Volume Resources 72
Volumes XML Entities 72
Volumes API Endpoints 72

Chapter 3: Error Codes

Chapter 4: Example Implementation
PHP 77

Chapter 5: Simple Testing From the Command Line

Appendix A: Warranty Information
Software End User License Agreement 81
READ BEFORE USING 81

CHAPTER 1: Introduction

Haivision's Furnace API is a Representational State Transfer (REST) API. REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web. REST provides a set of rules (constraints) to which an architecture should conform. This is in contrast to an "unconstrained architecture" in which services are free to define their own idiosyncratic interfaces.

The most important aspect of REST is a uniform interface between components, allowing them to communicate in a standard way. Requests use the standard HTTP methods. GET, PUT and DELETE requests can do only what is expected.

The effect is that your services are accessible through standard tools, and it is safe for other services and utilities to use yours in ways you did not predict.

NOTE To help keep applications stable with future versions of the API, please allow for, and ignore, unknown XML elements. When expanding the API, it may be necessary for Haivision to add elements to existing XML elements (without changing existing elements).

TIP All communication with the REST API is done through the portal server. Your base URI for requests should match the root of your portal server. In the examples, replace "https://example.haivision.com/" with the address of your portal server.

REST Informational Links

Following are some useful external references to learn more about REST:

- Architectural Styles and the Design of Network-based Software Architectures (dissertation by Roy Fielding)
- Representational State Transfer (Wikipedia entry)
- REST in Plain English
- Explaining REST
- How to Create a REST Protocol
- REST Anti-Patterns

URIs for REST Resources

NOTE "foobar" is a place holder name intended to represent whatever is being discussed.

URI Structure

Given a list of foobars, the following URI scheme provides access to the list of foobar entities and to a particular foobar:

URI	Method	Notes
/foobars	GET	This returns a collection of the foobar entities. By default items in the list are a minimal representation of a foobar entity. Note that we use the plural for the directory name.
/foobars	POST	This creates a foobar entity and returns a link to entity in the form /foobars/foobar-{id}.
/foobars/foobar-{id}	GET	This returns the full content of the foobar identified by the given id. Note that we use the singular for the entity name.
/foobars/foobar-{id}	PUT	Update the contents of a foobar entity.
/foobars/foobar-{id}	DELETE	Delete the foobar entity.

Sub-elements of a foobar entity are made available as sub-resources of /foobars/foobar-{id}, e.g.:

```
/foobars/foobar-{id}/bazs/baz-{id}/bloops/bloop-{id}
```

NOTE The ID in each URI comes from the collection preceding it. When a resource contains multiple IDs, the notation does not imply that the IDs are identical. Refer to the collection to get the ID.

OAuth

The Furnace API uses the OAuth (Open Authorization) standard for authorization when a third party application requests access. As defined in the OAuth 1.0 Protocol Abstract:

> "OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections."

OAuth is a standardized authentication mechanism that works by signing the HTTPS request using a shared secret. Furnace uses a "two-legged" implementation to control which applications can make use of the API on the Furnace. Two-legged OAuth does not provide user authentication, it only validates an application's identity.

Implementing OAuth

Implementing OAuth for the Furnace API is relatively simple and straightforward. However, it requires that both the server and client side behave the same way. Therefore, it is important to take care to avoid even minor mistakes, which can lead to authentication errors. The instructions which follow provide an overview of the signature process. Please refer the Interactive OAuth Guide listed under "OAuth Informational Links", which walks you through an interactive example of OAuth signature construction. Another great resource for understanding OAuth in action is the REST Client for Firefox, also listed below.

In both cases, leave the Accessor Secret, Token and Token Secret fields blank. The REST client allows you to manually enter the nonce (number used once, in this case a random string) and timestamp so you can verify that your signature is accurate. The instructions below outline how to sign a request, and store it in an HTTP header. The Furnace API also supports the signature information as part of the query string, or part of the post data, but the header is preferred.

NOTE Usage of Oauth with the Furnace API requires that calls be sent via HTTPS protocol.

OAuth Informational Links

- Official Site (http://oauth.net/ Official Site)
- Oauth RFC (http://tools.ietf.org/html/rfc5849 Oauth RFC (official spec))
- Authoritative Guide to OAuth (http://hueniverse.com/oauth/guide/)

- Interactive OAuth Guide (http://hueniverse.com/2008/10/beginners-guide-to-oauth-part-iv-signing-requests/ Interactive OAuth Guide (Use "Create Your Own" to see the full details))
- Simple OAuth Sample (http://developer.yahoo.com/blogs/ydn/posts/2010/04/a_twolegged_oauth_serverclient_example/ Simple OAuth Sample)
- OAuth Libraries (http://oauth.net/code/ OAuth Libraries)
- RESTClient for Firefox that supports OAuth (https://addons.mozilla.org/en-US/firefox/addon/restclient/) (Only fill in consumer key and consumer secret to authenticate)

Preparing for OAuth

The preliminary steps are done using the VF Admin module. First, from the Configuration page, you set the API Version to 2.0 to enable authentication. Please refer to the Administration Guide (Chapter 4: "VF Admin") for this procedure.

Then, from the Credential Manager page, you create the credential (i.e., a key or secret pair). Please refer to the Administration Guide (Chapter 3: "Initial System Setup") for this procedure.

When you have retrieved this API credential, you can proceed to the next step.

Generating the Request Base String

The next step is to generate OAuth headers.

1. Generate OAuth parameters.
   a. Generate a random nonce and store it as `oauth_nonce`.

NOTE This value should not be reused and should not be sequential.

   b. Generate a timestamp and store it as `oauth_timestamp`.
   c. Set `oauth_consumer_key` to the Consumer Key retrieved from the VF Admin Credential Manager (see "Preparing for OAuth" above).
   d. Set `oauth_signature_method` to "HMAC-SHA1". (No other methods are currently supported.)
2. Gather all parameters:
   - OAuth parameters
   - GET parameters
   - POST parameters
3. Encode the parameters using UTF-8 standards/functions.

4. Encode the parameters using URL standards/functions.
5. Normalize parameters (sort parameters alphabetically per http://tools.ietf.org/html/rfc5849#section-3.4.1.3.2).
6. Concatenate parameters together with an ampersand "&" between each, similar to HTTP GET requests.

Signing a Request with OAuth:

The next step is to generate the oauth_signature and place it in the Authorization header.

1. Generate the signature base string:

   The base string is a combination of:

   - Encoded HTTP request method (GET, POST, PUT, DELETE, etc.) (http://tools.ietf.org/html/rfc5849#section-3.6)
   - Ampersand
   - Base URI (e.g.: https://example.haivision.com/apis/assets) (Leave off the query portion)
   - Ampersand
   - Request base string (Construction noted above)

2. Encrypt the base string using HMAC-SHA1:

   - The result should be base64 encoded.
   - Store the result as oauth_signature.

3. Place OAuth values (oauth_signature, oauth_consumer_key, oauth_nonce, etc.) in the Authorization header, e.g.:

```
Authorization: OAuth oauth_consumer_key="",oauth_token=
    "",oauth_nonce="",oauth_timestamp="0",oauth_signature_method=
    "HMAC-SHA1",oauth_version="1.0",oauth_signature=
    "mrhdcH0WE%2BD%2FPrETh%2Bn1Gw4PSXc%3D"
```

REST API Responses

Responses to a request consist of two elements: the HTTP status code and the response content. An application can act initially upon the HTTP status code (sensing success or failure) and then act specifically upon the data of the response content.

Response content is usually returned as application/xml data, with the root level of `<response>`.

Within the `<response>`, the content is context-specific. Individual API functions specify the type of response content later in this documentation.

If there is a problem processing or executing the request, the response content may contain an `<error>` element with a more application-specific error code.

For more details on the HTTP and `<error>` responses, see "Error Codes" on page 75.

Example Success Response

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
    <recording>
        <id>99fe1d68-9ed6-4813-883b-1044f77a6b63</id>
        <sourceUrl>udp://239.19.3.100:4900/;raw=1</sourceUrl>
        <state>RECORDED</state>
        <duration>3</duration>
        <maxDuration>654</maxDuration>
        <progress>0.00</progress>
        <metadata>
            <title>The Goonies</title>
            <description>A group of kids embark on a wild adventure after finding
              a pirate treasure map.</description>
        </metadata>
        <link rel="self" type="application/xml" href=
          "https://example.haivision.com/apis/recorders/recorder-003018aafe23-
          4909-0/recordings/recording-99fe1d68-9ed6-4813-883b-
          1044f77a6b63"/>
        <link rel="recorder" type="application/xml" href=
          "https://example.haivision.com/apis/recorders/recorder-003018aafe23-
          4909-0"/>
        <link rel="hotmarks" type="application/xml" href=
          "https://example.haivision.com/apis/recorders/recorder-003018aafe23-
          4909-0/recordings/recording-99fe1d68-9ed6-4813-883b-
          1044f77a6b63/hotmarks"/>
        <link rel="publishes" type="application/xml" href=
          "https://example.haivision.com/apis/recorders/recorder-003018aafe23-
          4909-0/recordings/recording-99fe1d68-9ed6-4813-883b-
          1044f77a6b63/publishes"/>
        <link rel="reviews" type="application/xml" href=
          "https://example.haivision.com/apis/recorders/recorder-003018aafe23-
          4909-0/recordings/recording-99fe1d68-9ed6-4813-883b-
          1044f77a6b63/reviews"/>
    </recording>
</response>
```

Example Error Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/xml

<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
     <error>
          <code>1011</code>
          <message>Input XML data is poorly formatted</message>
     </error>
</response>
```

Sorting Response Content

Please note that the contents of a given container (for example, the tags within a <recording>, or the <recording> elements inside a <recorder>) are not guaranteed to be returned in any particular order.

To avoid unnecessary errors, it should not be assumed that the order of elements will follow those in the example (for example, <id> tags may not be the first tag in an element.)

If any sorting needs to be done (for example, to process <hotmark> items in chronological order), then this should be carried out by the application.

Diagnostics

In /opt/haivision/usr/www/html/apis/index.php, there is a $diagnostics variable defined at the top of the page. To enable diagnostics, set this variable to true.

When enabled, there will be a diagnostics XML entity similar to the following along with the regular response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
      <diagnostics>
            <systemTime>60.037</systemTime>
            <userTime>9.653</userTime>
      </diagnostics>

      ...

</response>
```

- <userTime> is the database execution time (or other execution time) measured in milliseconds
- <systemTime> is the total time spent during the API call measured in milliseconds

XML Entities

Each of the API references below contains details about the specific XML entities used.

NOTE {foobarID} is used throughout these examples to denote the unique identifiers referenced within the XML data. Keep in mind that this is not the syntax used in the actual results for these elements.

In general a request for a list of resources (/apis/resources) will return XML such as:

```
<resources>
     <resource>
          <id>abc</id>
     </resource>
     <resource>
          <id>xyz</id>
     <resource>
</resources>
```

A request for a single resource (/apis/resources/resource-xyz) will return XML such as:

```
<resource>
     <id>xyz</id>
</resource>
```

Generic entities you may come across are as follows:

<error>

```
<error>
     <code>1011</code>
     <message>Input XML data is poorly formatted</message>
</error>
```

<link>

```
<link rel="self" type="application/xml" href=
    "https://example.haivision.com/apis/demos/demo-{demoID}"/>
<link rel="publishes" type="application/xml" href=
    "https://example.haivision.com/apis/recorders/recorder-
    {recorderID}/publishes"/>
```

- rel: describes the relationship of the link to the current entity. Values vary depending on context.
- type: Content-Type of the linked data
- href: REST-navigable link to the indicated entity

CHAPTER 2: API Reference

This API command reference lists and describes the available resources for the Furnace API.

Topics In This Chapter

Summary of Furnace API Resources 22
Demo Resources 23
- Demo XML Entities 23
- Demo API Endpoints 23

Asset Resources 26
- Asset XML Entities 26
- Asset API Endpoints 27

Client Resources 32
- Client XML Entities 32
- Client API Endpoints 33

Command Resources 37
- Command XML Entities 37
- Command API Endpoints 42

Program Resources 43
- Program XML Entities 43
- Program API Endpoints 43

Recording Resources 45
- Recording XML Entities 45
- Recording API Endpoints 49

Recorder Resources 58
- Recorder XML Entities 58
- Recorder API Endpoints 59

Station Resources 63
- Station XML Entities 63
- Station API Endpoints 65

Volume Resources 72
- Volumes XML Entities 72
- Volumes API Endpoints 72

Summary of Furnace API Resources

The Furnace API consists of resources divided into the following categories:

Category	Description
Demo Resources	Use to explore the API in “demo” mode.
Asset Resources	For working with assets, such as video clips.
Client Resources	For working with clients connected to your Furnace system.
Command Resources	For executing commands on your Furnace system.
Program Resources	To get information about the scheduled programs in the system.
Recording Resources	For managing the output of a recorder process.
Station Resources	For managing stations/channels.
Recorder Resources	For managing the process of recording a video stream.
Volume Resources	For managing storage.

For a basic description of the XML entities referenced in these sections, see “XML Entities” on page 19.

“Endpoints” vs. “Methods”

In order to use this reference, keep in mind the following definitions:

- An endpoint is a URI (Uniform Resource Identifier) that points to a function or operation provided by the API, e.g., /apis/demos.
- A method, for the purposes of this document, refers to the HTTP methods GET, PUT, DELETE, or POST. An HTTP method acts on a Furnace API endpoint.

Demo Resources

The demos API is provided for testing purposes to help integrators become familiar with the API interaction process without touching any live components. It is designed to demonstrate the API, i.e., to allow you to make changes and test your own tools without affecting the system at large.

Demo XML Entities

A demo entity consists of two elements: a “name” and a “value”. A unique ID will be generated upon creation (using the POST method on /apis/demos – see below).

<demo>

```
<demo>
    <id>{demoID}</id>
    <name>myName</name>
    <value>myValue</value>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/demos/demo-0"/>
</demo>
```

Demo API Endpoints

/apis/demos

HTTP Method: GET

- Description: Gets a list of all <demo> entities.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <demos> entity, containing one or more <demo> entities.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.

HTTP Method: POST

- Description: Creates a new <demo> entity.
- Media Types:
  - application/xml
- Input Data:
  - <demo> entity containing at least one of the following tags:
    - <name>
    - <value>
- Return Data:
  - <link> entity referencing the newly added entry
- HTTP Return Codes:
  - 201: Request was successful.
  - 400: Client request is bad.
  - 500: Server error.

/apis/demos/demo-{id}

HTTP Method: GET

- Description: Gets the demo entity specified by {id}.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <demos> entity, containing one <demo> entity.
- HTTP Return Codes:
  - 200: Desired id found.
  - 404: Desired id not found.

HTTP Method: PUT

- Description: Updates the demo entity specified by {id}.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> entity referencing the newly updated demo entity
- HTTP Return Codes:
  - 200: Update is successful.
  - 400: Client request is bad.
  - 500: Server error.

HTTP Method: DELETE

- Description: Deletes the demo entity specified by {id}.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> entity to the parent level of the /demos API
- HTTP Return Codes:
  - 200: Update is successful.
  - 400: Client request is bad.
  - 500: Server error.

Asset Resources

The assets API is an interface to the assets published in the Furnace system. Currently only queries are supported.

Asset XML Entities

<asset>

```
<asset>
    <id>9c4fe054-50fe-4c71-a37f-8e437286c03d</id>
    <title>Wildlife</title>
    <description>Wildlife</description>
    <runtime>139</runtime>
    <created>20080815</created>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/assets/asset-9c4fe054-50fe-4c71-
        a37f-8e437286c03d"/>
    <link rel="thumbnail" type="image/png" href=
        "https://example.haivision.com/thumbnail.php?9c4fe054-50fe-4c71-a37f-
        8e437286c03d&width=90&height=68"/>
    <link rel="launch_link" type="text/html" href=
        "https://example.haivision.com/launch/9c4fe054-50fe-4c71-a37f-
        8e437286c03d"/>
    <tags>
        <tag>Classroom</tag>
        <tag>Nurses</tag>
        <tag>Students</tag>
    </tags>
    <metadata>
        <vfa_type>offline</vfa_type>
        <vfa_title_brief>Wildlife</vfa_title_brief>
        <vfa_comment>Aaron</vfa_comment>
        <uuid>9c4fe054-50fe-4c71-a37f-8e437286c03d</uuid>
        <hotmarks>
            <hotmark>
                <time>30000</time>
                <title>My HotMark</title>
            <hotmark>
         </hotmarks>
    </metadata>
</asset>
```

- <created> described creation date as YYYYMMDD. This tag is not required to have content (<created/> tag only is valid).
- <link rel="thumbnail"> provides the URL of the PNG thumbnail associated with this video asset. If no thumbnail was embedded, this link will not be available.
- <link rel="launch_link"> provides a url that can be used to launch this video asset.

- <asset> as returned will contain a <tags> element, if that asset has Tags specified on it. Inside <tags> will be <tag> elements for each tag on the asset.

The following elements are returned when directly accessing an asset at /apis/assets/asset-{id} and are not included when getting an asset list from /apis/assets:

- <metadata> is metadata contained within the asset file. The exact contents can change depending on what tags are present inside the file's metadata.
- <hotmark> contains the title and time in milliseconds

Asset API Endpoints

/apis/assets

HTTP Method: GET

- Description: Retrieves a list of published assets. Results are paginated and supports keyword query on title and description.
- URL Parameters:
  - page (optional)

    The page number to access. Default page is 1.
  - size (optional)

    The maximum number of results to return. Default size is 100. Size can range from 1 to 100.
  - Search Pattern (optional)
    - q Simple search string. Search is performed on the asset title, description and tags.
    - c Complex search string. Value can be either “and” or “or”, indicating how to combine search entries. The value is case insensitive.

      If c is used, q is ignored. Any additional query parameters are treated as search entries. The name of a query parameter indicates the field to match. Any field from the metadata is acceptable. The following fields are also accepted:
      - title
      - description
      - tag
      - runtime
      - created
    - The complex query supports equal, not equal, like and not like.
      - Equal is the normal mode.
      - Not equal is indicated by an exclamation point (!) following the equal sign.
      - Like is indicated with a tilde (~) following the equal sign.

    - Not like is indicated with a carrot (^) following the equal sign.
- Media Types:
    - application/xml
- Input Data:
    - none
- Return Data:
    - An <assets> entity, containing one or more partial <asset> entities (without <metadata> tag).
- HTTP Return Codes:
    - 200: Results found.
    - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/assets/?page=1&size=2&q=2mb

- Example Response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
<assets numResults="8" pageSize="2" page="1">
    <asset>
        <id>b863515b-85f5-4f71-af5f-0e67d06b0949</id>
        <title>2Mb720x480p30</title>
        <description>2Mb720x480p30</description>
        <runtime>300</runtime>
        <created>20080815</created>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/assets/asset-b863515b-85f5-4f71-
            af5f-0e67d06b0949"></link>
        <link rel="launch_link" type="text/html" href=
            "https://example.haivision.com/launch/b863515b-85f5-4f71-af5f-
            0e67d06b0949"></link>
        <tags>
            <tag>Classroom</tag>
            <tag>Nurses</tag>
            <tag>Students</tag>
        </tags>
    </asset>
    <asset>
        <id>85625601-9aab-4624-b929-2b6e86fbdc7d</id>
        <title>2mb1280x720p1fps</title>
        <description>2mb1280x720p1fps</description>
        <runtime>300</runtime>
        <created>20080815</created>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/assets/asset-85625601-9aab-
            4624-b929-2b6e86fbdc7d"></link>
        <link rel="thumbnail" type="image/png" href=
            "https://example.haivision.com/thumbnail.php?85625601-9aab-4624-
            b929-2b6e86fbdc7d"></link>
        <link rel="launch_link" type="text/html" href=
            "https://example.haivision.com/launch/85625601-9aab-4624-b929-
            2b6e86fbdc7d"></link>
    </asset>
</assets>
```

- Example Complex Request:

  https://server/apis/assets/?page=1&size=2&c=or&title=Sandlot

- Example Complex Request:

  https://server/apis/assets/?page=1&size=2&c=or&title=!Sandlot

- Example Complex Request:

  https://server/apis/assets/?page=1&size=2&c=and&title=
  ~Sand&vfa_nu_creator=Fox

/apis/assets/asset-{id}

HTTP Method: GET

- Description: Retrieves information for a specific asset.
- URL Parameters:
  - none.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - Complete <asset> entity (including <metadata> element).
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/assets/asset-85625601-9aab-4624-b929-2b6e86fbdc7d

- Example Response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
    <asset>
        <id>85625601-9aab-4624-b929-2b6e86fbdc7d</id>
        <title>2mb1280x720p1fps</title>
        <description>2mb1280x720p1fps</description>
        <runtime>300</runtime>
        <created>20080815</created>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/assets/asset-85625601-9aab-
            4624-b929-2b6e86fbdc7d"></link>
        <link rel="thumbnail" type="image/png" href=
            "https://example.haivision.com/thumbnail.php?85625601-9aab-4624-
            b929-2b6e86fbdc7d"></link>
        <link rel="launch_link" type="text/html" href=
            "https://example.haivision.com/launch/85625601-9aab-4624-b929-
            2b6e86fbdc7d"></link>
        <metadata>
            <uuid>85625601-9aab-4624-b929-2b6e86fbdc7d</uuid>
            <vfa_type>offline</vfa_type>
            <hotmarks>
                <hotmark>
                        <time>10000</time>
                        <title>HotMark_A</title>
                 <hotmark>
             </hotmarks>
        </metadata>
    </asset>
</response>
```

Client Resources

The clients API allows you to view clients that are currently connected to the system.

Client XML Entities

<client>

```
<client>
    <instance>8f36ef57-a686-4221-8fe9-7013322a932f</instance>
    <session>a6f1601d-844b-444c-86be-e913fa74736b</session>
    <app>INSTREAM</app>
    <app_version>5.8.0</app_version>
    <macaddr>33:33:33:33:33:33</macaddr>
    <ipaddr>3.3.3.3</ipaddr>
    <hostname>example.haivision.com</hostname>
    <platform>OS X</platform>
    <settings>0</settings>
    <url>uuid:68f8deca-59f7-11b6-9df0-000a95d96580</url>
    <callsign/>
    <channel>0</channel>
    <packets_corrected>0</packets_corrected>
    <packets_uncorrected>0</packets_uncorrected>
    <link rel="self" type="application/xml"href=
        "https://example.haivision.com/apis/clients/client-8f36ef57-a686-4221-
        8fe9-7013322a932f"/>
</client>
```

- appID is one of the following: UNKNOWN, INSTREAM, EDITOR, PILOT, MONITOR, ARCHIVE, COMMANDER, NVR, DECODER.
- platform is one of the following: UNKNOWN, Windows, OS X, Linux, Solaris, STB, Makito.

Client API Endpoints

/apis/clients

HTTP Method: GET

- Description: Gets the list of clients with details.
- URL Parameters:
  - page (optional)
    The page number to access. Default page is 1.
  - size (optional)
    The maximum number of results to return. Default size is 100. Size can range from 1 to 100.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <clients> element containing multiple <client> elements.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:
  https://example.haivision.com/apis/clients/?page=1&size=2

HTTP Method: Example Response:

```
<?xml version"1.0" encoding="ISO-8859-1"?>
<response>
    <clients numResults="2" pageSize="100" page="1">
        <client>
            <instance>8f36ef57-a686-4221-8fe9-7013322a932f</instance>
            <session>a6f1601d-844b-444c-86be-e913fa74736b</session>
            <app>INSTREAM</app>
             <app_version>5.8.0</app_version>
             <macaddr>33:33:33:33:33:33</macaddr>
             <ipaddr>3.3.3.3</ipaddr>
             <hostname>example.haivision.com</hostname>
              <platform>OS X</platform>
              <settings>0</settings>
              <url>uuid:68f8deca-59f7-11b6-9df0-000a95d96580</url>
              <callsign/>
              <channel>0</channel>
              <packets_corrected>0</packets_corrected>
              <packets_uncorrected>0</packets_uncorrected>
              <link rel="self" type="application/xml"
                 href="https://example.haivision.com/apis/clients/client-8f36ef57-a686-
                 4221-8fe9-7013322a932f"/>
        </client>
        <client>
            <instance>2336ef57-a686-4221-8fe9-7013322a932f</instance>
            <session>a6f1601d-844b-444c-86be-e913fa74736b</session>
            <app>INSTREAM</app>
            <app_version>5.8.0</app_version>
            <macaddr>33:33:33:33:33:33</macaddr>
            <ipaddr>3.3.3.3</ipaddr>
            <hostname>example.haivision.com</hostname>
            <platform>OS X</platform>
            <settings>0</settings>
            <url>uuid:68f8deca-59f7-11b6-9df0-000a95d96580</url>
            <callsign/>
            <channel>0</channel>
            <packets_corrected>0</packets_corrected>
            <packets_uncorrected>0</packets_uncorrected>
            <link rel="self" type="application/xml"
                href="https://example.haivision.com/apis/clients/client-2336ef57-a686-
                4221-8fe9-7013322a932f"/>
        </client>
    </clients>
</response>
```

/apis/clients/client-{id}

HTTP Method: GET

- Description: Gets the details of a client.
- URL Parameters:
  - none
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <client> element.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/clients/client-91605d67-829b-4034-b2af-5169c2358341

- Example Response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response>
    <client>
        <id>91605d67-829b-4034-b2af-5169c2358341</id>
        <session_id>a818b72c-7ded-4622-abe6-111bc4f8af5b</session_id>
        <app_id>INSTREAM</app_id>
        <app_version>5.8.0</app_version>
        <mac_address>00:11:22:33:44:AA</mac_address>
        <ip_address>192.168.1.101</ip_address>
        <hostname>tester.haivision.com</hostname>
        <platform>OS X</platform>
        <settings>0</settings>
        <url>uuid:00015f91-0000-0000-0000-000000000000</url>
        <callsign>Playback1</callsign>
        <channel>100</channel>
        <packets_corrected>0</packets_corrected>
        <packets_uncorrected>0</packets_uncorrected>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/clients/client-91605d67-829b-
            4034-b2af-5169c2358341"></link>
    </client>
</response>
```

Command Resources

The commands API allows you to issue commands to clients connected to the system.

IMPORTANT Not all client applications respond to all commands.

Command XML Entities

<command>

Below are the types of commands. The first is a simple action requiring only a type and a value. The others are examples requiring more complex parameters. Multiple actions can be placed inside <actions>.

HTTP Method: Simple Actions:

```
<command>
    <actions>
        <action type="volume">
            <value>100</value>
        </action>
    </actions>
</command>
```

Available actions (type <value>):

- lockinterface <on|off>
- channel <number>
- station <id>
- jump
- guide <on|off>
- mute <on|off>
- volume <0-100>
- cc <0|2|4>
  - 0 - CC1
  - 2 - CC3
  - 4 - off

- ontop <on|off>
- dashboard <on|off>
- thumbnail <on|off>
- minimize <on|off>
- activate
- show <on|off>
- fullscreen <on|off>
- power <on|off>
- sleeptimer <minutes>
- hdtv <SD|720p|1080i|1080p>
- settvmode <format>
- delay <ms>
- url <url|uuid> [use to launch a specific VOD asset or URL on a device]
  - URL (e.g., udp://239.10.10.10:4900) or
  - UUID identifier (uuid:xxxx-xxxx-xxxx-xxxxxxx)
- quit

HTTP Method: Video Overlay Message:

```
<command>
    <actions>
        <action type="message/video">
            <duration>30</duration>
            <priority>7</priority>
            <text>This is a test of the message overlay.</text>
            <font_size>32</font_size>
            <brightness>255</brightness>
            <color red= "255" green="2550" blue="255" alpha="255"/>
            <position>7</position>
            <scroll_speed>44.0</scroll_speed>
        </action>
    </actions>
</command>
```

This action causes an overlay to display in supporting clients.

Available parameters for Video Overlay Message:

- Duration - Time in seconds
- Priority - Message Priority 0=lowest, 254=highest
- Text - Message to display
- Font Size - Font size in pixels
- Brightness - Brightness value 0-255
- Color - Text Color
- Position - Location on screen
  - 0 - top left
  - 1 - top center
  - 2 - top right
  - 3 - middle left
  - 4 - middle center
  - 5 - middle right
  - 6 - bottom left
  - 7 - bottom center
  - 8 - bottom right
- Scroll Speed - Rate the text scrolls across the screen. Default: 44.0

HTTP Method: Dialog Message:

```
<command>
    <actions>
        <action type="message/dialog">
            <duration>30</duration>
            <priority>7</priority>
            <text>This is a test of the message dialog.</text>
        </action>
    </actions>
</command>
```

This causes a dialog to pop up in supporting clients.

Available parameters for Dialog Message:

- Duration - Time in seconds
- Priority - Message Priority 0=lowest, 254=highest
- Text - Message to display
- Title - Title for dialog box

HTTP Method: Network Config:

```
<command>
   <actions>
      <action type="network_config">
        <bootproto>static</bootproto>
        <ipaddress>192.168.0.3</ipaddress>
        <netmask>255.255.255.0</netmask>
        <gateway>192.168.0.1</gateway>
        <dns1>192.168.0.1</dns1>
        <dns2>192.168.0.2</dns2>
      </action>
   </actions>
   <restrict_to>
      <conditions operator='OR'>
        <condition type="instance">
          <value>{id}</value>
        </condition>
      </conditions>
   </restrict_to>
</command>
```

This action changes the network configuration on supporting clients. If bootproto is set to "dhcp", the other fields can be left blank. An "instance" restriction is required for this command to operate correctly. This command only targets set-top boxes.

Available parameters for Network Config:

- bootproto
- dhcp (if set, all other parameters can be left out)
- static
- ipaddress
- netmask
- dns1
- dns2

NOTE The "restrict_to" block is supported on all commands, not just the "network_config" command. The "restrict_to" block supports the following "type" parameters. Their values should match the responses in the client's API.

- app
- session
- instance
- callsign
- channel
- ipaddr
- macaddr
- platform

With the exception of the "instance" type, "restrict_to" is limited to a single condition in the current release. However, the "conditions" block is required when "restrict_to" is used. The operator on the "restrict_to" block MUST be set to "OR". If "restrict_to" is not specified, the command goes out to all clients.

Command API Endpoints

/apis/commands

HTTP Method: POST

- Description: Issues a command.
- URL Parameters:
  - None
- Media Types:
  - application/xml
- Input Data:
  - <command> element
- Return Data:
  - None.
- HTTP Return Codes:
  - 201: Request was successful.
  - 400: Request is bad.
  - 500: Server error.
- Example Request:

  https://example.haivision.com/apis/commands POST DATA:

```
<command>
    <actions>
        <action type="volume">
            <value>255</value>
        </action>
        <action type="channel">
            <value>100</value>
        </action>
    </actions>
</command>
```

Program Resources

The program API allows you to get information about the scheduled programs in the system.

NOTE Use the schedules request in the stations API to find programs. There is no collection containing all programs. For details, see "Station API Endpoints" on page 65.

Program XML Entities

<program>

```
<program>
    <id>b3692d4a-4150-4326-acfa-c420c145aa71</id>
    <title>rule mere lost</title>
    <description>region lot fortune</description>
    <numTracks>2</numTracks>
</program>
```

Program API Endpoints

/apis/programs/program-{id}

HTTP Method: GET

- Description: Gets information about a scheduled program.
- URL Parameters:
  - none
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <program> element
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.

- Example Request:

  https://example.haivision.com/apis/programs/program-b3692d4a-4150-4326-acfa-c420c145aa71

- Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
    <program>
        <id>b3692d4a-4150-4326-acfa-c420c145aa71</id>
        <title>rule mere lost</title>
        <description>region lot fortune</description>
        <numTracks>2</numTracks>
    </program>
</response>
```

Recording Resources

The recording API is an interface to VF NVR to manage recordings in the NVR system.

Recording XML Entities

<recording>

```
<recording>
    <id>{recorderID}</id>
    <sourceUrl>vftp://239.1.2.3:4900</sourceUrl>
    <state>RECORDED</state>
    <duration>1600</duration>
    <maxDuration>100</maxDuration>
    <progress>0.00</progress>
    <metadata>
        <title>myTitle</title>
        <description>myDescription</description>
    </metadata>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}"/>
    <link rel="recorder" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-{recorderID}"/>
    <link rel="hotmarks" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}/hotmarks"/>
    <link rel="publishes" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}/publishes"/>
    <link rel="reviews" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}/reviews"/>
</recording>
```

- <duration> and <maxDuration> are in seconds.
- <state> may be one of the following: RECORDING, PAUSED, FINALIZING, RECORDED, REVIEWING, PUBLISHING.
- <progress> represents the current progress of this recording's publish (if a publish is active.) Ranges from 0.00 to 1.00.

<review>

```
<review>
    <id>9b876a02-92cc-4047-bdce-9d644a63510f</id>
    <link rel="recorder" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-00505637c7b0-
        4909-0"></link>
    <link rel="recording" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-00505637c7b0-
        4909-0/recordings/recording-9b876a02-92cc-4047-bdce-
        9d644a63510f"></link>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-00505637c7b0-
        4909-0/recordings/recording-9b876a02-92cc-4047-bdce-
        9d644a63510f/reviews/review-9b876a02-92cc-4047-bdce-
        9d644a63510f"></link>
    <outputUrl>{outputUrl}</outputUrl>
</review>
```

{URL} specifies the streaming URL for the review. This may take one of two forms:

- A streaming output URL, multicast or unicast:
  e.g., vftp://239.19.3.100:4900

  -or-

  vftp://192.168.1.100:4900
- A "resource" access URL allowing for the VoD-style streaming of the asset to an InStream player via command-line arguments:
  e.g., vfms://example.haivision.com/{ID}

NOTE The "recorder" and "self" link is not available in Version 1.0 API.

The outputURL will only appear in response to a POST.

<publish> resource

```
<publish>
      <id>{publishID}</id>
      <progress>0.26</progress>
    <link rel="recorder"  type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-{recorderID}"/>
    <link rel="recording" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}"/>
    <link rel="self"   type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}/publishes/publish-
        {publishID}"/>
</publish>
```

- The <publish> resource is an output, unlike <publish> POST resource (see below) which is an optional input.
- <progress> represents the current progress of this publish. Ranges from 0.00 to 1.00.

<publish> POST resource

```
<publish>
    <volume>
        <id>eff3c133-3f17-441f-a6a0-7509126f0a17</id>
    </volume>
</publish>
```

- The <publish> POST resource is an optional input that may be used when creating a publish resource to indicate a desired volume.

<hotmark>

```
<hotmark>
      <id>{hotmarkID}</id>
      <time>147000</time>
      <title>MyHotmark</title>
      <type>TAG</type>
    <link rel="recorder" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-{recorderID}"/>
    <link rel="recording" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}"/>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings/recording-{recordingID}/hotmarks/hotmark-
        {hotmarkID}"/>
</hotmark>
```

- <time> is in milliseconds
- <type> may be one of: TAG (for normal HotMarks) or CHAPTER (for discontinuity/ Chaptermarks)

Recording API Endpoints

/apis/recordings

HTTP Method: GET

- Description: Gets a list of all recordings in the NVR Crash system.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <recordings> element containing multiple <recording> elements
- HTTP Return Codes:
  - 200: Recordings are found.
  - 404: No recordings found.

NOTE With API Version 1.0, you will not receive a 404 for an empty set of recordings. The return code will be 200 and the response will have an empty <recordings> element.

HTTP Method: POST

- Description: Starts a recording on the specified recorder. Takes XML with sourceUrl, maxDuration, title, and description elements.
- Media Types:
  - application/xml
- Input Data:
  - <recording> element containing the following tags:
    - <sourceUrl>
    - <maxDuration>
    - <metadata>, containing: <title>, <description>
    - Optionally, <tracks> containing one or more: <track> which contains an <id> of the tracks to limit the number of tracks included in a multitrack recording. If <tracks> is omitted, all tracks are recorded.

- Return Data:
  - <link> element to the newly created recording.
  - <tracks> will be returned with an attribute "numberOfTracks" set to the total number of tracks and an attribute "numberOfTracksRecording" set to the total that could be recorded.

    In addition, an attribute "licenseLimited" will be set to "true" or "false" to indicate that the number of tracks requested is more than the license allows for a single recording.

    If "licenseLimited" is "false" and "numberOfTracks" is greater than "numberOfTracksRecording," this indicates that you do not have enough recorders available.
- HTTP Return Codes:
  - 201: Recording successfully started.
  - 400: Client request is bad.
  - 500: Server error.

NOTE The url can be specified in one of three formats:

vftp://10.10.10.10:4900
udp://10.10.10.10:4900
uuid:0001737d-0000-0000-0000-000000000000

Replace ip addresses and ports as necessary. The uuid format refers to a station. The uuid can be found in the id field of the station record in the stations api.

/apis/recordings/recording-{id}

HTTP Method: GET

- Description: Gets the details of the requested recording.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <recording> element
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.

HTTP Method: PUT

- Description: Updates the recording with new data, and/or changes the state (Pause/Resume).
- Media Types:
  - application/xml
- Input Data:
  - <recording> element, containing each of the following tags:
    - <maxDuration>
    - <metadata>, containing: <title> and/or <description>
      and/or
    - <state>, containing one of the following states, PAUSED or RECORDING
- Return Data:
  - <link> element to the newly created recording.
- HTTP Return Codes:
  - 200: Recording successfully updated.
  - 400: Client request is bad.
  - 500: Server error.

NOTE Setting the maxDuration <= the current recorded duration will immediately "stop" the recording.

The id in the url is the relevant resource id. The two values should *not* be identical.

Updating title/description can be done during RECORDING/PAUSE/REVIEW state. Updating duration/state can only be done during RECORDING/PAUSE

HTTP Method: POST

- Description: Stops the recording.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> element to the updated recording.

- HTTP Return Codes:
  - 200: Recording successfully stopped.
  - 400: Client request is bad.
  - 500: Server error.

HTTP Method: DELETE

- Description: Deletes the recording.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> element, linking to the <recorder> associated with this recording.
- HTTP Return Codes:
  - 200: Recording successfully deleted.
  - 400: Client request is bad.
  - 500: Server error.

/apis/recordings/recording-{id}/hotmarks

HTTP Method: GET

- Description: Gets the list of all current HotMarks for this recording.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <hotmark> element containing multiple <hotmark> elements.
- HTTP Return Codes:
  - 200: HotMarks are found.
  - 404: No HotMarks found.

HTTP Method: POST

- Description: Creates a new HotMark for the recording.
- Media Types:
  - application/xml
- Input Data:
  - partial <hotmark> element containing:
    - <title> (required)
    - <time> (optional, omit to place HotMark at current recording time)
    - <type> (optional. TAG or CHAPTER, defaults to TAG)
- Return Data:
  - <link> to the recording containing the new HotMark.
- HTTP Return Codes:
  - 201: HotMark successfully created.
  - 400: Client request is bad.
  - 500: Server error.

/apis/recordings/recording-{id}/hotmarks/hotmark-{id}

HTTP Method: GET

- Description: Gets information about the desired HotMark.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <hotmark> element
- HTTP Return Codes:
  - 200: Requested HotMark was found.
  - 404: Requested HotMark not found.

/apis/recordings/recording-{id}/publishes

HTTP Method: GET

- Description: Gets current publishes on this recording, if existing.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <publishes> element containing one <publish> element.
- HTTP Return Codes:
  - 200: Publish was found.
  - 404: No publish was found.

NOTE Currently only one publish per recording is supported.

HTTP Method: POST

- Description: Begins publishing the recording to the media server.
- Media Types:
  - application/xml
- Input Data:
  - (Optional) You can use a <publish> element to indicate a desired volume.
- Return Data:
  - <link> to the recording being published.
- HTTP Return Codes:
  - 202: Publish was successfully started.
  - 400: Client request is bad.
  - 500: Server error.

/apis/recordings/recording-{id}/publishes/publish-{id}

HTTP Method: GET

- Description: Gets information on the specified publish event.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <publish> element.
- HTTP Return Codes:
  - 200: Publish was found.
  - 404: No publish found.

HTTP Method: DELETE

- Description: Stops publishing the recording and returns recording to RECORDED state.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> element, linking to the <recording> associated with this publish.
- HTTP Return Codes:
  - 200: Publish successfully canceled.
  - 400: Client request is bad.
  - 500: Server error.

/apis/recordings/recording-{id}/reviews

HTTP Method: GET

- Description: Gets all the reviews for the specified recording.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <reviews> element containing multiple <review> elements.
- HTTP Return Codes:
  - 200: Reviews are found.
  - 404: No reviews found.

HTTP Method: POST

- Description: Begins review on this recording.
- Media Types:
  - application/xml
- Input Data:
  - Optional:
    - <review> element containing only an <outputUrl> tag specifying a multi-cast or unicast address to stream to.
    - If omitted, provides outputUrl for VoD-style viewing with InStream player (vfms://example.haivision.com/{ID})
- Return Data:
  - <review> element
- HTTP Return Codes:
  - 201: Review was successfully started.
  - 400: Client request is bad.
  - 500: Server error.

NOTE When beginning a VoD-style review (no outputUrl specified), the REVIEWING state is only ended when explicitly deleted via the API. When reviewing via streaming address (multicast or unicast outputUrl), the REVIEWING state ends automatically when the stream completes playback.

/apis/recordings/recording-{id}/reviews/review-{id}

HTTP Method: GET

- Description: Gets details for the specified review.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <review> element.
- HTTP Return Codes:
  - 200: Review is found.
  - 404: Review not found.

HTTP Method: DELETE

- Description: Stops and removes the specified review.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <link> element, linking to the <recording> associated with this review.
- HTTP Return Codes:
  - 200: Review was successfully stopped.
  - 400: Client request is bad.
  - 500: Server error.

Recorder Resources

A recorder is the software/hardware resource that controls and manages the recording, publishing, and reviewing of video in the NVR system.

A given recorder:

- Supports a maximum of one active recording at a time (a stream currently being recorded).
- May have one or more finished, unpublished recordings that are available for review, publishing or deletion.

Recorder XML Entities

<recorder>

```
<recorder>
    <id>{recorderID}</id>
    <service>vfnvrd</service>
    <isRecording/>
    <isSlave/>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-{recorderID}"/>
    <link rel="publishes" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/publishes"/>
    <link rel="recordings" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/recordings"/>
    <link rel="reviews" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-
        {recorderID}/reviews"/>
    <link rel="master" type="application/xml" href=
        "https://example.haivision.com/apis/recorders/recorder-{recorderID}" />
</recorder>
```

- <isRecording> will either:
  - Be an empty tag (<isRecording/>) if there is no currently active recording on this recorder, or
  - Contain a value (<isRecording>1</isRecording>) if the recorder has an active recording and will not allow a new recording to start.

- <isSlave> will either:
  - Be an empty tag (<isSlave/>) if the recorder is not in use by another recorder.
  - Contain a value (<isSlave>1</isSlave>) if the recorder is currently in use by another recorder.
  - If the recorder is a slave, it will have a <link rel="master">. All commands must be directed at the master.

Recorder API Endpoints

/apis/recorders

HTTP Method: GET

- Description: Gets a list of all registered recorders on the system.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <recorders> element containing multiple <recorder> elements
- HTTP Return Codes:
  - 200: Results found.

/apis/recorders/recorder-{id}

HTTP Method: GET

- Description: Gets the recorder entity specified by {id}.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <recorder> element
- HTTP Return Codes:
  - 200: Desired id found.
  - 404: Desired id not found.

/apis/recorders/recorder-{id}/reviews

HTTP Method: GET

- Description: Gets all reviews associated with this recorder's recordings.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <reviews> element containing multiple <review> elements
- HTTP Return Codes:
  - 200: Reviews found.
  - 404: No reviews found.

/apis/recorders/recorder-{id}/reviews/review-{id}

HTTP Method: GET

- Description: Gets the specified <review> element.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <review> element
- HTTP Return Codes:
  - 200: Review event found.
  - 404: No review events found.

/apis/recorders/recorder-{id}/publishes

HTTP Method: GET

- Description: Gets all publishes associated with this recorder's recordings.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <publishes> element containing multiple <publish> elements
- HTTP Return Codes:
  - 200: Publishes are found.
  - 404: No publishes found.

/apis/recorders/recorder-{id}/publishes/publish-{id}

HTTP Method: GET

- Description: Gets the specified <publish> element with transfer status.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <publish> element
- HTTP Return Codes:
  - 200: Publish event is found.
  - 404: No publish events found.

Mirrrored Endpoints

The following endpoints are all mirrors of the Recording Endpoints, with results isolated to the selected recorder.

HTTP Method: /apis/recorders/recorder-{id}/recordings

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/hotmarks

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/hotmarks/hotmark-{id}

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/publishes

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/publishes/publish-{id}

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/reviews

HTTP Method: /apis/recorders/recorder-{id}/recordings/recording-{id}/reviews/review-{id}

Station Resources

The stations API allows you to get detailed information about stations available in the system.

Station XML Entities

The following example responses show two different possible constructions of <station> elements: the first for multi-stream and the second for single-stream.

For example, when a multi-stream <station> is returned within <stations>, it has <tracks> and no <outputUrl>.

<station> (multi-stream)

```
<station>
    <id>0001737f-0000-0000-0000-000000000000</id>
    <callsign>MultiTrack</callsign>
    <channel>103</channel>
    <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/stations/station-0001737f-0000-0000-
        0000-000000000000"></link>
    <link rel="schedule" type="text/html" href=
        "https://example.haivision.com/apis/stations/station-0001737f-0000-0000-
        0000-000000000000/schedule"></link>
    <link rel="launch_link" type="text/html" href=
        "https://example.haivision.com/launch/0001737f-0000-0000-0000-
        000000000000"></link>
    <tracks numberOfTracks="2" >
        <track>
            <id>0001737f-0000-0000-0000-000000000001</id>
            <title>Daft Punk</title>
            <outputUrl>vftp://239.35.55.101:4900</outputUrl>
            <link rel="launch_link" type="text/html" href=
                "https://example.haivision.com/launch/0001737f-0000-0000-0000-
                000000000001"></link>
          </track>
          <track>
            <id>0001737f-0000-0000-0000-000000000002</id>
            <title>Big Buck</title>
            <outputUrl>vftp://239.35.55.102:4900</outputUrl>
            <link rel="launch_link" type="text/html" href=
                "https://example.haivision.com/launch/0001737f-0000-0000-0000-
                000000000002"></link>
          </track>
      </tracks>
</station>
```

When a <station> element is part of a <stations> entry, the <tracks> element will be a sparse entry with no <track> elements displayed. If the <tracks> element is not present, the station is not a multi-track station.

<station> (single-stream)

```
<station>
    <id>0001737e-0000-0000-0000-000000000000</id>
     <callsign>Playback1</callsign>
     <channel>102</channel>
     <outputUrl>vftp://239.35.55.102:4900</outputUrl>
     <link rel="self" type="application/xml" href=
        "https://example.haivision.com/apis/stations/station-0001737e-0000-0000-
        0000-000000000000"></link>
     <link rel="schedule" type="text/html" href=
        "https://example.haivision.com/apis/stations/station-0001737e-0000-0000-
        0000-000000000000/schedule"></link>
     <link rel="launch_link" type="text/html" href=
        "https://example.haivision.com/launch/0001737e-0000-0000-0000-
        000000000000"></link>
 </station>
```

In contrast, single-stream stations have an <outputUrl> in <station> and no <tracks>.

Station API Endpoints

/apis/stations

HTTP Method: GET

- Description: Gets the list of stations with details.
- URL Parameters:
  - none.
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <stations> element containing multiple <station> elements.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/stations

- Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <stations>
        <station>
            <id>0001737e-0000-0000-0000-000000000000</id>
            <callsign>Playback1</callsign>
            <channel>102</channel>
            <outputUrl>vftp://239.35.55.102:4900</outputUrl>
            <link rel="self" type="application/xml" href=
                "https://example.haivision.com/apis/stations/station-0001737e-
                0000-0000-0000-000000000000"></link>
            <link rel="schedule" type="text/html" href=
                "https://example.haivision.com/apis/stations/station-0001737f-
                0000-0000-0000-000000000000/schedule"></link>
            <link rel="launch_link" type="text/html" href=
                "https://example.haivision.com/launch/0001737e-0000-0000-0000-
                000000000000"></link>
        </station>
        <station>
            <id>0001737f-0000-0000-0000-000000000000</id>
            <callsign>Dual Live</callsign>
            <channel>103</channel>
            <tracks numberOfTracks="2"></tracks>
            <link rel="self" type="application/xml" href=
                "https://example.haivision.com/apis/stations/station-0001737f-
                0000-0000-0000-000000000000"></link>
            <link rel="schedule" type="text/html" href=
                "https://example.haivision.com/apis/stations/station-0001737f-
                0000-0000-0000-000000000000/schedule"></link>
            <link rel="launch_link" type="text/html" href=
                "https://example.haivision.com/launch/0001737f-0000-0000-0000-
                000000000000"></link>
        </station>
    </stations>
</response>
```

/apis/stations/station-{id}

HTTP Method: GET

- Description: Gets the details of a station.
- URL Parameters:
  - none.
- Media Types:
  - application/xml

- Input Data:
  - none
- Return Data:
  - <stations> element.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/stations/station-00015f91-0000-0000-0000-000000000000

- Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <station>
        <id>0001737f-0000-0000-0000-000000000000</id>
        <callsign> Dual Live </callsign>
        <channel>103</channel>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/stations/station-0001737f-0000-
            0000-0000-000000000000"></link>
        <link rel="schedule" type="text/html" href=
            "https://example.haivision.com/apis/stations/station-0001737f-0000-
            0000-0000-000000000000/schedule"></link>
        <link rel="launch_link" type="text/html" href=
            "https://example.haivision.com/launch/0001737f-0000-0000-0000-
            000000000000"></link>
        <tracks numberOfTracks="2" >
            <track>
                <id>0001737f-0000-0000-0000-000000000001</id>
                <title>Front Camera</title>
                <outputUrl>vftp://239.35.55.101:4900</outputUrl>
                <link rel="launch_link" type="text/html" href=
                    "https://example.haivision.com/launch/0001737f-0000-0000-
                    0000-000000000001"></link>
            </track>
            <track>
                <id>0001737f-0000-0000-0000-000000000002</id>
                <title>Side Camera</title>
                <outputUrl>vftp://239.35.55.102:4900</outputUrl>
                <link rel="launch_link" type="text/html" href=
                    "https://example.haivision.com/launch/0001737f-0000-0000-
                    0000-000000000002"></link>
            </track>
        </tracks>
    </station>
</response>
```

/apis/stations/station-{id}/schedule

HTTP Method: GET

- Description: Gets the schedule for a station.
- URL Parameters:
  - page (optional)
    - The page number to access. Default page is 1.
  - size (optional)
    - The maximum number of results to return. Default size is 100. Size can range from 1 to 100.
  - Timeframe Pattern (optional)
    - t0: The exact time of playback for an asset or the start timeframe.
    - t1: The ending time of a timeframe. If t1 is specified, t0 must be specified.

NOTE Time is specified as timestamps in the Linux epoch format, e.g., 1327937409

To search for active programs or beginning at a specific time, specify that time with t0.

To search for programs that may be active or beginning in a range of time, specify the beginning/end of that time range with t0 and t1 respectively.

  - Search Pattern (optional)
    - q Simple search string. Search is performed on the asset title, description and tags.
    - c Complex search string. Value can be either “and” or “or”, indicating how to combine search entries. The value is case insensitive.

      If c is used, q is ignored. Any additional query parameters are treated as search entries. The name of a query parameter indicates the field to match. Any field from the metadata is acceptable. The following fields are also accepted:
      - title
      - description
      - tag
      - runtime
      - created
- Media Types:
  - application/xml
- Input Data:
  - none

- Return Data:
  - a <schedule> entity, containing one or more <scheduleItem> entities.
- HTTP Return Codes:
  - 200: Results found.
  - 400: Client request is bad.
  - 404: Unknown ID
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/stations/station-0001737d-0000-0000-0000-000000000000/schedule

- Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
    <schedule num_results="3" pageSize="100" page="1">
    <stationNum>95101</stationNum>
    <link rel="station" type="application/xml" href=
        "https://example.haivision.com/station-0001737d-0000-0000-0000-
        000000000000" />
    <scheduleItem>
        <id>1327492800</id>
        <airDateTime>1327492800</airDateTime>
        <airDuration>9</airDuration>
        <link rel="program" type="application/xml" href=
            "https://example.haivision.com/apis/programs/program-b3692d4a-
            4150-4326-acfa-c420c145aa71" />
    </scheduleItem>
    <scheduleItem>
        <id>1327492809</id>
        <airDateTime>1327492809</airDateTime>
        <airDuration>9</airDuration>
        <link rel="program" type="application/xml" href=
            "https://example.haivision.com/apis/programs/program-f74eb830-
            29d6-4107-9939-583b33f20d03" />
    </scheduleItem>
    <scheduleItem>
        <id>1327492818</id>
        <airDateTime>1327492818</airDateTime>
        <airDuration>9</airDuration>
        <link rel="program" type="application/xml" href=
            "https://example.haivision.com/apis/programs/program-fe484347-
            a788-410f-93a2-a06380338a2f" />
    </scheduleItem>
    </schedule>
</response>
```

Volume Resources

The volumes API allows you to get detailed information about the volumes in the system.

Volumes XML Entities

<volumes>

```
<volumes>
    <volume>
        <id>3a74932e-555d-11e0-8e15-00505637c7b1</id>
        <name>Default Volume</name>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/volumes/volume-3a74932e-555d-
            11e0-8e15-00505637c7b1"></link>
    </volume>
</volumes>
```

Volumes API Endpoints

/apis/volumes

HTTP Method: GET

- Description: Gets the list of volumes.
- URL Parameters:
  - none
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <volumes> element containing multiple <volume> elements.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/volumes

- Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <volumes>
        <volume>
            <id>3a74932e-555d-11e0-8e15-00505637c7b1</id>
            <name>Default Volume</name>
            <link rel="self" type="application/xml" href=
                "https://example.haivision.com/apis/volumes/volume-3a74932e-
                555d-11e0-8e15-00505637c7b1"></link>
        </volume>
    </volumes>
</response>
```

/apis/volumes/volume-{id}

HTTP Method: GET

- Description: Gets the details for a volume.
- URL Parameters:
  - none
- Media Types:
  - application/xml
- Input Data:
  - none
- Return Data:
  - <volume> element.
- HTTP Return Codes:
  - 200: Results found.
  - 404: No results found.
- Example Request:

  https://example.haivision.com/apis/volumes/volume-3a74932e-555d-11e0-8e15-00505637c7b1

- Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <volume>
        <id>3a74932e-555d-11e0-8e15-00505637c7b1</id>
        <name>Default Volume</name>
        <free_mb>164166</free_mb>
        <total_mb>175718</total_mb>
        <link rel="self" type="application/xml" href=
            "https://example.haivision.com/apis/volumes/volume-3a74932e-555d-
            11e0-8e15-00505637c7b1"></link>
    </volume>
</response>
```

CHAPTER 3: Error Codes

An error condition will return a specified HTTP status code on the requested action.

If the error is processed internally by the API, the returned data may also contain an <error> entity (see "XML Entities" on page 19).

XML <error> Code	HTTP Status Code	Error Message	Common Cause
1000	500 Bad Request	Failed to create new resource	Submitted data was not sufficient, missing required data.
1001	404 Not Found	No results found	Request completed successfully, but no results were found.
1002	404 Not Found	Unknown id	A specific resource was requested but not found (deleted or bad ID specified).
1003	500 Internal Server Error	Error executing SQL query	An internal function failed while executing the request.
1004	500 Bad Request	Failed to update resource	Tried to update a nonexistent entity (bad ID)
1005	500 Bad Request	Failed to delete resource	Tried to delete a nonexistent entity (bad ID)
1006	501 Not Implemented	Unknown API function requested	Tried to access a nonexistent / inactive API location
1007	400 Bad Request	Unknown HTTP method	Tried to use a non-standard HTTP method (other than GET, POST, PUT, or DELETE)
1008	400 Bad Request	Unrecognized URI structure	Too many levels in specified path (e.g., /demos////demo-1)
1009	501 Not Implemented	HTTP method not implemented	Requested an action not utilized by the target API (e.g., POST on a query-only API)
1010	501 Not Implemented	Function not implemented	
1011	400 Bad Request	Input XML data is poorly formatted	XML content submitted had syntax or validation problems.

XML <error> Code	HTTP Status Code	Error Message	Common Cause
1012	500 Internal Server Error	Error while executing	
1013	400 Bad Request	Unrecognized arguments	
1014	401 Not Authorized	Not Authorized	Authentication credentials are invalid, expired, or removed. -or- Accessing API site via HTTP protocol instead of HTTPS
1015	403 Forbidden	API functions not enabled	API access is disabled in server configuration
1016	503 Service Unavailable	Service provider for this API is unavailable	A server process that supports this API call was unable to be reached. It may be down, inaccessible, or overloaded.

CHAPTER 4: Example Implementation

PHP

This example implementation uses the PHP curl library to interface with the demos API documented above. In the example, the XML POST and PUT data is built as a simple string. In your production implementation, you may want to investigate DOMDocument or any of the other XML classes that are available in PHP.

NOTE This sample is provided for informational purposes only. It is not intended to be used in a production environment. In particular, we recommend that you avoid copying/pasting on platforms other than Windows to avoid formatting issues.

Also, this sample only applies when the API version is set to 1.0 (from the VF Admin module, Configuration page). Version 2.0 of the API requires OAuth, and it will not function properly.

```
<?php

$host = "server.mydomain.com";

//
// Build an HTTP request using cURL
//
function DoHTTPRequest($url, $method, $xmlData)
{
   // initialize curl
   $ch = curl_init();

   // configure curl options
   ////////////////////////////////
   // set the URL
   curl_setopt($ch, CURLOPT_URL, $url);
   // set the HTTP method (GET, POST, PUT, DELETE)
   curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
   // specify the content is XML
   curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/xml'));
   // specify the XML data to submit
   curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlData);
   // return a string from curl_exec(), on false the result is echoed out
   curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
```

```
    // execute our HTTP request
    $result = curl_exec($ch);

    // close curl resource
    curl_close($ch);

    // return the result
    return $result;
}


// Example handler that gets a function and id from the user as URL parameters
// Based on the parameters perform one of the CRUD tasks with our Demo API

// determine what action the user is trying to do
switch($_REQUEST['function'])
{
    case 'create_resource':
    {
        // specify we want to use an HTTP POST
        $method = "POST";
        // the demo database has two columns name and value
        // build an xml string to send with default values for our new record
        $data = "<data><name>MyName</name><value>MyValue</value></data>";
        // send the request
        $result = DoHTTPRequest("https://$host/apis/demos", $method, $data);
        // echo the result
        echo $result;
        break;
    }
    case 'get_resource':
    {
        // specify we want to use an HTTP GET
        $method = "GET";
        // this is a GET so there is no data to submit, some API calls may support a search
condition here
        $data = "";
        // check if the user wants a particular ID
        if(isset($_REQUEST['id']))
        {
            // get a particular record
            // send the request
            $result = DoHTTPRequest("https://$host/apis/demos/demo-{$_REQUEST['id']}",
$method, $data);
        }
        else
        {
            // get all records
            // send the request
            $result = DoHTTPRequest("https://$host/apis/demos", $method, $data);
        }
```

```
        // echo the result
        echo $result;
        break;
    }
    case 'update_resource':
    {
        // specify we want to use an HTTP PUT
        $method = "PUT";
        // the demo database has two columns name and value
        // build an xml string to send with the new values for our record
        $data =
"<data><name>MyUpdatedName</name><value>MyUpdatedValue</value></data>";
        // send the request, we expect the user to have submited an id number to edit
        $result = DoHTTPRequest("https://$host/apis/demos/demo-{$_REQUEST['id']}",
$method, $data);
        // echo the result
        echo $result;
        break;
    }
    case 'delete_resource':
    {
        // specify we want to use an HTTP DELETE
        $method = "DELETE";
        // this is a DELETE so there is no data to submit, some API calls may support a
condition here
        $data = "";
        // send the request, we expect the user to have submited an id number to delete
        $result = DoHTTPRequest("https://$host/apis/demos/demo-{$_REQUEST['id']}",
$method, $data);
        // echo the result
        echo $result;
        break;
    }
    default:
    {
        // print an error for unrecognized function requests
        echo "ERROR: unrecognized function - {$_REQUEST['function']}";
    }
}

?>
```

CHAPTER 5: Simple Testing From the Command Line

Some very basic interaction can be done with the command-line tool “cURL” (http://curl.haxx.se/) which is normally installed on most Unix/Linux or OSX systems.

A few examples are provided below to help you familiarize yourself with the system without the need for writing or debugging code:

Simple GET to an API location:

```
curl "https://SERVER/apis/demos"
```

POST a new entity:

```
curl -X POST -H "Content-Type: application/xml" -d
  "<demo><name>MyDemo</name></demo>" "https://SERVER/apis/demos"
```

DELETE an entity:

```
curl -X DELETE "https://SERVER/apis/demos/demo-1"
```