EchoSystem 5 Documentation

API Documentation

February 10, 2012
Echo360 is continually updating the documentation. This manual is a snapshot as of the date above.
Check the Echo360 website for the most current version: http://echo360.com/customer-support/download.asp

API Documentation

In this section:

Overview

This section provides detailed documentation about the APIs available to extend and integrate the EchoSystem at your institution.

Capture Appliance API

The API for the EchoSystem Capture Appliance provides a mechanism for an external system, such as an In-Room Control System (IRCS) or a custom desktop monitoring application, to interact with the capture device to control and monitor the capture functions. The API provides access to capture status information, allows the initiation of ad-hoc captures, and enables record/pause/resume/stop functions using a customizable interface.

The latest revisions to this API are not yet fully documented. In the meantime, the API specification from the EchoSystem 2.4 release is available. All commands documented for the 2.4 version are compatible with the 2.5 version.

Download the API package (500 KB)

Server-Side API for Scheduling

EchoSystem 2.5 introduced Server-Side APIs, starting with the Scheduling API. The Scheduing API enables programmatic EchoSystem scheduling via a RESTful interface.

Server-Side API for Scheduling documentation is available online in this section. The downloadable PDF version is coming soon.

2.5 Schedule API Specification

In this section:

Introduction

EchoSystem 2.5 introduces the first stage of the Server Side API, starting with the necessary requirements for scheduling related integrations. The API enables programmatic EchoSystem scheduling via a RESTful interface. This guide covers the various aspects of the API including a technical reference.

Facilitating the management and monitoring of EchoSystem is the central EchoSystem Server (ESS) component. The ESS centrally manages production workflows (such as capture schedules), EchoSystem capture components and presentation production. The ESS is a Java-based, modular server application which can provide, or be integrated with, other applications servicing the data infrastructure necessary to facilitate campus-wide deployment. Naturally, some of the data contained within the ESS is also contained within other systems in the university environment. Similarly, some of the data contained within the ESS is often related to information contained within other systems. For this reason it may be desirable to more closely integrate the ESS with these related systems. Server-Side APIs are the means by which EchoSystem data can be integrated with data from other systems. The Scheduling API is an extension to the ESS that provides external systems direct access to query, update and delete the schedule related data contained within the ESS.

An example of a third party application that would make use of the API is an existing timetabling or campus scheduling system where the schedule data in the external application is automatically used to create capture schedules in the EchoSystem, thereby integrating the EchoSystem with existing workflows. Another example would be an online booking form that enables Faculty/Academics to request their lectures to be captured. Such a web form might have a backend that integrates with the ESS to add the capture schedules dynamically.

Who Should Read This Document?

EchoSystem Server-Side APIs are programmatic interfaces to extend the capabilities of the system primarily for integration with other campus

systems. If you are responsible for projects related to extending EchoSystem via APIs, external systems related to EchoSystem integrations, or writing code to integrate with EchoSystem via APIs, particularly as these pertain to integrated scheduling, you should read this document.

Project Managers
Personnel responsible for EchoSystem integration projects.

Needs to consider

- EchoSystem integration requires a dedicated project plan and experienced personnel.
- EchoSystem will be integrated with another system, also requiring experienced personnel.

Programmer / Developers
Personnel responsible for writing code to integrate with EchoSystem APIs.

Needs to consider

- RESTful application development
- Existing schedule system data model
- EchoSystem schedule data model

Planning for Success

The EchoSystem Scheduling API is a programmatic interface to the EchoSystem. The purpose of utilizing the API is of course a successful integration between at least two systems, EchoSystem and an external system. Such projects have a set of well-known best practices to ensure a successful project. Below is a list of such items as they relate to the EchoSystem API.

- Your team should be familiar with the process of creating schedules in the EchoSystem Server UI and scheduling terminology. See Manage Schedules.
- Echo360 offers planning guidance for early adopters. We can help you get familiar with the API, set realistic expectations, identify key integration points, provide basic code examples, etc. Contact your regional sales manager, solutions engineer or technical support for details.
- Application development should have a project plan with measurable outcomes.
- Application development should have dedicated staff identified and time lines accounted for. The API is a programmatic interface to the EchoSystem and therefore requires the knowledge and experience of a programmer/application developer.
- The EchoSystem is just one of at least two systems affected during an integration. An expert on the other affected system(s) should be identified as well.
- Application development should be done on a test system before being used in production. Echo360 can provide a hosted sandbox environment as a test system should the resources not be available to you on campus. Please contact technical support for details.

REST overview

In this section:

Introduction

In many ways, REST (Representational State Transfer) is a deliberate return to the basics that made the World Wide Web itself successful. The web arguably became successful because it was conceptually simple, yet powerful.

In its purest form, there were web pages that had addresses (URLs). Web pages could be viewed using a web browser, which used HTTP (the HyperText Transfer Protocol) to retrieve the web pages. Furthermore, web pages could have links, which allowed users to traverse relationships between web pages.

In Echo360's implementation of RESTful web services, the conceptual basis of the API is resources, rather than web pages. A resource is anything that is important enough to be referenced as a distinct 'thing'. Rooms are resources, as are presenters, schedules, etc. Accordingly, Echo360's approach for web services may also be referred to as a resource?oriented architecture.

Like web pages, each resource has a distinct URL from which information about the resource can be retrieved. Unlike web pages, it is possible to perform actions related to resources, such as creating them, updating them, deleting them, etc. All of the actions that can be reasonably performed on a resource can be accomplished using the HTTP protocol.

HTTP Methods

The HTTP standard defines several different methods for performing operations on resources:

- GET ? Retrieve a resource.
- POST ? Create a new resource.
- PUT ? Update a resource.
- DELETE ? Delete a resource.

The GET and POST methods are already widely used by web browsers for retrieving web pages and for posting data from forms. The PUT and DELETE methods have not had widespread usage until recently, despite being specified in the HTTP standard more than twelve years ago. The advent of RESTful web services has changed this; their usage is rapidly becoming more common.

RESTful Operations

The four HTTP methods (GET, POST, PUT and DELETE) can be combined with distinct URLs to provide a web services interface that allows users to perform operations on a resource.

URL	HTTP Method	Operation
http://example.com/rooms	GET	Get a list of all rooms.
http://example.com/buildings/123/rooms	GET	Get a list of rooms within building "123"
http://example.com/buildings/123/rooms;	POST	Create a new room within building "123"
http://example.com/rooms/1	GET	GET detailed data on room "1"
http://example.com/rooms/1	PUT	Update data on room "1".
http://example.com/rooms/1	DELETE	Delete room "1".

In **Table 1**, the various URLs allow callers to perform actions on a room or, more specifically, the metadata associated with a room.

The primary advantage of this architecture is that it offers a regular and predictable interface for managing most types of resources. For developers, using a RESTful web service is as simple as sending a request to a specific URL using the appropriate HTTP method. HTTP even supports the capability to send arguments with a request.

Representation Formats and HTTP Status Codes

Each web service request sent via HTTP results in a response being sent back to the caller. At a minimum, the caller receives an HTTP status code that defines the status of the request. HTTP defines numerous standard status codes that can easily be leveraged by a web service to provide meaningful information to a caller, e.g. - 404 ("Not Found").

Both the request and the response may optionally include content. For example, when creating the metadata for a new schedule rule, the request will need to include information about the schedule. Most web services accept such content in easy?to?parse formats such as form?encoded parameters (the format for data that is POSTed from a traditional HTML form), XML and JSON. That data format is referred to as the representation format of the request.

Likewise, most web services respond to a request by sending back content in easy-to? parse formats such as JSON, XML or various flavors of XML such as Atom, RSS or XHTML. That data format is referred to as the representation format of the response. Callers can parse the content provided in the response and reuse it for their own purposes.

Many web services allow users to select from a choice of representation formats for the request and the response. For example, a caller might send a request that included JSON content and receive a response that included Atom content. Currently, the EchoSystem Scheduling API accepts XML content and produces XML responses.

REST vs. SOAP

REST and SOAP represent two very different strategies for implementing web services. A RESTful architecture makes web services as conceptually simple as possible, with a standard approach for representing operations on resources. REST also makes no assumptions about the representation formats, such as XML, used for content that may be sent back to callers.

SOAP is more elaborate than REST in many ways. There are numerous official standards layered on top of each other, providing features for remote procedure calls, security, etc. There is no common way of representing operations on resources. Instead, there is a standard approach for representing any type of content in XML in a generic fashion. With SOAP interfaces, the strategy is to have libraries that automatically parse SOAP?related XML and generate objects based on the content, which can then be interrogated to retrieve data elements as needed.

In simplistic terms, REST focuses on providing a standard interface for resources, while SOAP focuses on providing a standard and generic representation for content. With REST, developers may have to create custom parse logic for content received from a RESTful web service. With SOAP, content can be parsed automatically and made available to the caller. In practice, REST representation formats tend to be much simpler than SOAP?related XML, so custom parsing is not typically a problem.

In general, RESTful web services are usually simpler and easier to use than SOAP interfaces. Additionally, SOAP interfaces often do not work well in cross?language environments, e.g. - SOAP clients in languages such as Perl, Ruby, Python and other languages may not work properly with SOAP services. RESTful interfaces are usually less complex than SOAP interfaces, so language compatibilities are less of a problem.

Most importantly, implementing SOAP interfaces can be much more time consuming than implementing RESTful interfaces, particularly in a language such as Java. For these reasons, Echo360 has selected REST as the basis for this API.

REST References

The following references may be useful in understanding REST:

- Richardson, L. and S. Ruby (2007). RESTful web services. Farnham, O'Reilly.
- Pollack, G. (2007). ""The Future of Web Services" Video." Retrieved 4 May, 2009, from http://www.railsenvy.com/2007/12/17/the-future-of-web-services.

Schedule API Web Service Overview

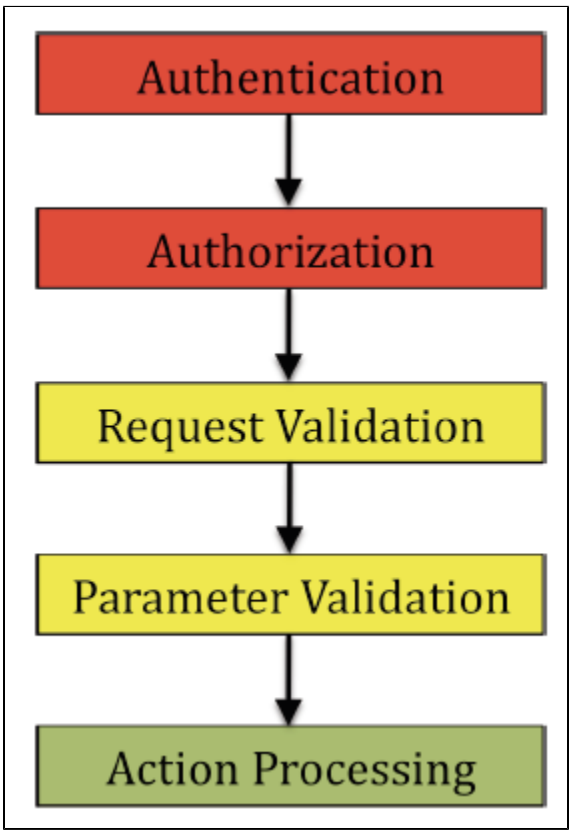
In this section:

Overview

This section provides information about the basic operation of the EchoSystem RESTful Scheduling Web Services API.

Anatomy of a Web Request

The EchoSystem Scheduling Web Service treats every incoming request the same way. The figure below illustrates the steps that a request goes through before the underlying action is performed. Each step is described in the sections that follow.



Authentication

You must have a valid trusted system API Key in order to use the API. The API Key must also be provided with every request. The key is used to determine the identity of the caller. If the API Key is not provided, or if an invalid API key is provided, the action will not be performed and the response will have a status code of 401 ("Not Authorized").

Trusted API Keys are created in the EchoSystem Server UI on the **Configuration** tab, ***Trusted Systems*** subtab.

Authorization

Having established the identity of the caller, the system next determines whether the caller is authorized to perform the action on the specified resource. EchoSystem Server manages authorizations through a system of roles, groups and privileges. In the EchoSystem 2.5 release these authorization levels are not exposed. There is only one level of authorization which has the ability to perform all API actions.

Request Validation

Once authorization has been verified, the request is validated. Typically, this means that any content provided with the request is parsed, resulting in a list of parameters. If the content cannot be parsed, then validation fails; the action will not be performed and the response will have a status code of 400 ("Bad Request").

Parameter Validation

After the request is validated, the parameters provided by the request are validated. Validation will fail if required parameters have not been provided or if unknown parameters have been provided. Each parameter value is also validated for 1) type, 2) length, and 3) acceptable values. If parameter validation fails, the action will not be performed and the response will have a status code of 511 ("Failed Validation").

Action Processing

The specified action is performed, such as creating a new playlist. If the action succeeds, the response will have a status of 200 ("OK"). In the event of a failure, the response will have an appropriate status code. The most common status code if an error occurs at this stage will be 500 ("Internal Server Failure")

API Reference - Getting Started

In this section:

Common Symbols In This Document

Curly Braces { }: This document encloses symbolic variables within curly braces. The API user will substitute the variable with a concrete value. For example: "GET /buildings/{building-id}/rooms/{room-id}" indicates the caller should provide a concrete Building ID and Room ID when issuing the stated HTTP GET.

Square Brackets [ ]: Items enclosed in square brackets indicate optional parameters. For example: " GET /buildings/{building-id}/rooms[?term="{name-pattern}"] " indicates that the list rooms query supports an option parameters called "term" which takes a name pattern value.

*{base-uri}: Within this document, the {base-uri} symbol represents the common or root HTTP URL used by all API requests.

- The actual base-uri will be determined by the system configuration.
- An example {base-uri} is "https://api.echo360.com:8443/ess/scheduleapi/v1"

*{entityName-id}: Many of the API request URLs require the caller to specify one of more entity ids. These ids are GUID strings. The documentation will simply use the {entityName-id} symbol to indicate that the caller provides an entity id.

Representations of entities are transferred by the scheduling RESTful API. Three standard levels of entity representation (XML) are supported by the API; link, detailed and summary. These levels are referred to in the API reference tables as "Rep. Levels".

Link

The "link" representation is the corresponding URL that will retrieve an entity's "detailed" representation. Links usually refer to associated parent or child entities. For instance: a "detailed" building representation will contain a link to its rooms. The XML element "link" along with its attributes "href", "rel", and "title", define the link representation.

Example: <link href="{base-uri}/buildings/{building-id}/rooms" rel="http://schemas.com.echo360.rooms" title="rooms"/>

There can be optional content for the link element. This is expected to be a short description of the link target.

Detailed

The "detailed" representation contains all fields of a given attribute. This representation is provided on "POST" requests and "GET" entity responses. The top level XML element will be the entity name; such as "room". Its children elements map to the entity's properties/attributes. For example:

```
<room>
   <id>938ab3edf432047a</id>
   <link href="{base-uri}/buildings/1939abdf982efac312"
     rel="http://schemas.com.ech360.building" title="building"/>
   <name>washington-101</name>
</room>
```

Summary

When lists of entities are requested, the API response is a list of entity summaries. Each summary contains a link to the detailed entity. This special link uses a "rel=self" link attribute.

Basic API Response

Following the RESTful API pattern, the EchoSystem Scheduling API returns HTTP response headers with a XML content/body. The returned XML is either a "collection" of summary entities or a single detailed "entity" XML element. Typically, a given collection element is simply plural name of the entities within the collection.

Entity vs Entity Collection: Each entity operated upon by the API has an associated entity collection. For instance: a "room" entity has an associated "rooms" collection.
Following the RESTful pattern, entities are accessed from their associated collection.

So a request for a specific room is addressed relative to the rooms collection. This pattern shows up in the API request URL. The basic URL pattern is "{base-uri}/{collection-name}/{entity-id}"

Example: "GET {base-uri}/rooms/{room-id}" returns the "detailed" representation of the given room.

The associated collection name is the English plural of the entity name. Collections are returned by the API when the URL does not specify an entity id.

Example: "GET {base-uri}/rooms" returns a collection of all rooms within the organization.

For instance the "rooms" collection is:

```
<rooms>
  <total-results>25</total-results>
  <items-per-page>25</items-per-page>
  <start-index>0</start-index>


<\!-\- each room element is the "summary" representation of the entity \-->

  <room><room/>
  <room><room/>
  <room><room/>
  ...
</rooms>
```

Authentication and Authorization

EchoSystem Server Scheduling API utilizes OAuth for handling system authentication and authorization. The implementation is similar to that implemented by NetFlix® .

Authentication is the process of verifying who sent a given API request. It also includes steps to verify that the request was not altered by an intermediate party.

Authentication does not address privacy of the data being transmitted.

EchoSystem Server requires each API request to be "signed" by an API User ID and shared secret. The OAuth standard defines how to sign the HTTP API requests; including the format of the HTTP Authorization header field.

For the Scheduling API the User is referred to as a "Trusted System". Each trusted system must register with the EchoSystem Server via the UI on the **Configuration** tab, ***Trusted Systems*** subtab ([https://]{ess-uri}/ess/security/ListTrustedSystems.html). The registration process generates a Trusted System ID (Key) and a shared secret. The Trusted system then signs all their associated requests with these credentials. The third party system needs to secure the credentials; to prevent another party from impersonating the trusted system.

Authorization is the process of determining if a given authenticated API user is allowed to perform the request action. Many systems define user roles which restrict the action that a user can take. Currently, the Scheduling API does not restrict the actions of an Authenticated API user.

Data Privacy

The data transmitted between the external system and the Echo360 System may be sensitive or deemed private. HTTP transmits data "in the clear"; while HTTPS encrypts all data transmitted. The Echo360 System supports both HTTP and HTTPS for scheduling API requests.

OAuth provides encryption of the authentication signature, but does not address data privacy. Using OAuth with HTTP will protect requests from tampering. The API User should use HTTPS if they want to protect their data privacy.

OAuth Specifics

The Scheduling API uses a simplified two-step OAuth authorization process. The OAuth standard also supports a more complex three-step process that uses session based request/auth tokens along with the consumer key/secret. The three-step process is targeted to internet hosted services that are accessed by many diverse consumer clients.

In the simplified two-step scheme, the consumer needs to register with the Echo360 System, in order to be assigned a unique consumer key and shared secret. These credentials are long lived and not limited to a session timeout. The consumer does not need to request a session token/secret (OAuth request & auth tokens/secrets) before initiating a request to the Echo360 System. Instead, the consumer just uses the consumer key/secret and an empty auth token/secret to sign all scheduling API requests.

As per the OAuth specification, the OAuth parameters and signature are passed in the HTTP "Authorization" header field.

OAuth Terms

Below are the terms defined by the OAuth specification and how they apply to the Scheduling API.

- **Producer:** The EchoSystem Server (ESS).
- **Consumer:** The system trusted by the ESS.
- **Consumer Key:** The key supplied by the API User when registering as a Trusted System.
- **Consumer Secret:** The shared secret generated by the ESS when the consumer was registered as a Trusted System.
- **Request Token/Secret:** The Scheduling API currently utilizes a simple two-step OAuth authorization process. The request token and request secret are not issued by the ESS.
- **Auth Token/Secret:** Since the Scheduling API requires a manual registration process for the consumer key/secret, the session based auth token is skipped and defaults to empty strings.

References

The following references may be useful in understanding OAuth:

- Official OAuth Documentation - http://oauth.net/documentation/getting-started
- NetFlix Authentication Overview - http://developer.netflix.com/docs/Security#0_18325

EchoSystem Server (ESS) Global Defaults and Inheritance

EchoSystem version 2.5 introduced a new set of global default values and object inheritance rules. The defaults are configured in the ESS UI, **Configuration** tab, ***Global Defaults*** subtab. Objects created using the Scheduling API, such as a section, will inherit the global default values. Likewise, settings configured for a section will be assumed for new schedule rules created via the API. We recommend reviewing the EchoSystem User Guide for details on how the global defaults and object inheritance rules work.

Requirements for a Schedule

The following resources are required for successful creation of capture schedules via the API:

- Room - see API Reference - Room Related Resources
- Section - see API Reference - Course Section Related Resources

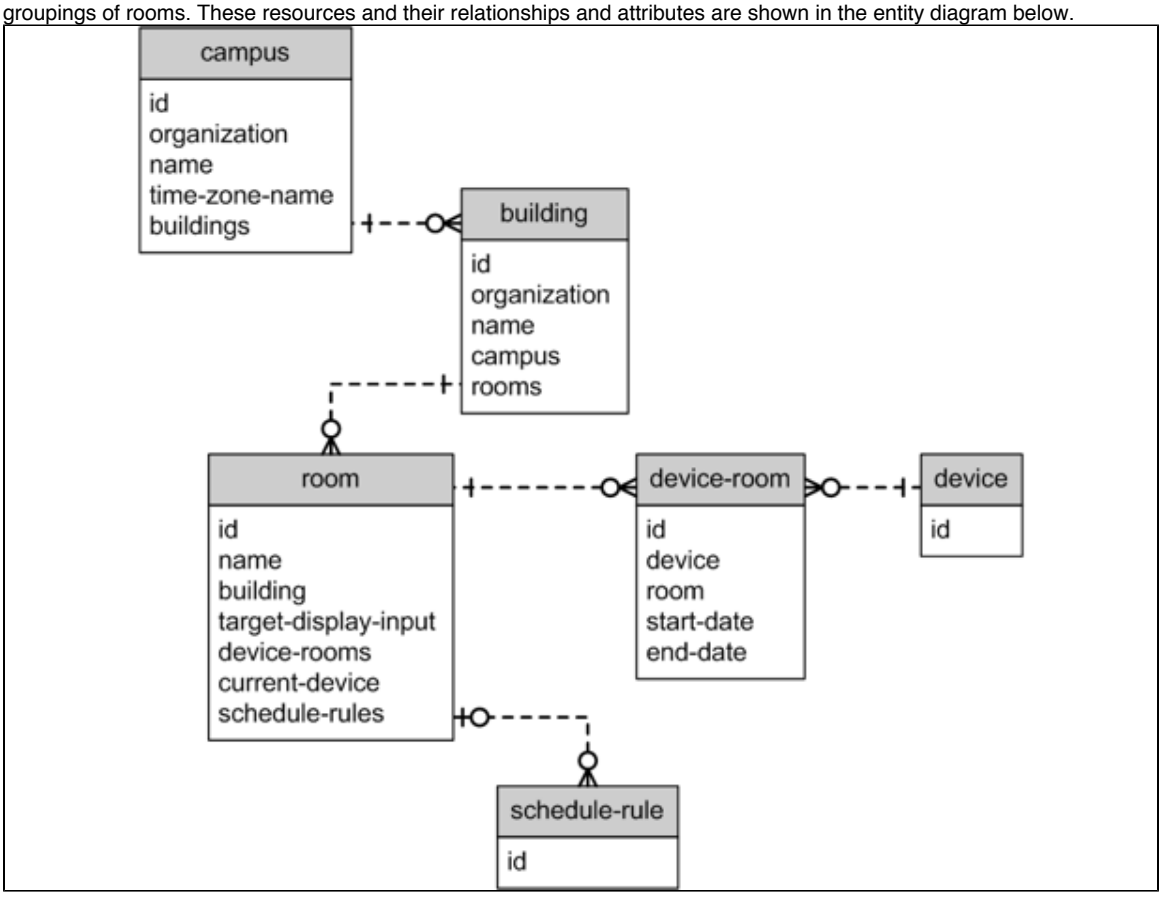
- Presenter - see API Reference - Person Related Resources
- Schedule Rule (date/time) - see API Reference - Schedule Related Resources

API Reference - Room Related Resources

In this section:

Room Related API Resources (Campus, Building, and Room)

The Room entity/resource is a required scheduling component. Capture Devices are assigned to rooms and rooms are actually scheduled for capture, resulting in a capture on a device assigned to a room. The Campus and Building are secondary entities/resources that provide natural groupings of rooms. These resources and their relationships and attributes are shown in the entity diagram below.



Campus Entity (campus) / Collection (campuses)

A Campus belongs to a parent Organization and has a collection of Buildings. The Campus entity primarily provides a means of grouping rooms. The campus time zone attribute is inherited by all buildings and associated rooms within the campus.

Campus Entity XML mapping

The below table defines the mapping of campus attributes/fields to/from its various XML representations (summary or detailed). Note: optional fields can be omitted from the XML representations if the value is null/empty. Omitting empty fields allows concise data representation.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.

organization	All	link	Link to the parent organization for this campus. Auto set to the caller's default organization.
name	All	string	Required, unique campus name.
time-zone-name	Detailed	string	Optional, defaults to caller's default time zone. [http://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones] Examples: "America/New_York", "Pacific/Honolulu"
buildings	Detailed	link	Link to the building collection. Initially empty at create.

Campus Collection Resource URL

Method	URL: {base-uri}/campuses[?term="{name-pattern}"]
GET	Retrieve the list of Campus entities. The optional "term" parameter limits the list to campus objects who's name is like the term value. **Reply content:** "campuses" collection XML containing "campus" summary elements.
POST / PUT	Add a new campus to the campus collection resource. **Request Content:** "campus" detail XML (name, time-zone-name elements) **Response Content:** "campus" summary XML ("id" element holds new ID) **Errors:** duplicate name, bad time-zone, client error (content invalid)

Campus Entity Resource URL

Method	URL: {base-uri}/campuses/{campus-id}
GET	Retrieve the detailed information on the Campus entity. **Reply content:** "campus" detailed XML. **Errors:** not found
POST / PUT	Update the specified campus entity. **Request Content:** "campus" detail XML. Only need to specify the fields that are being updated (name, time-zone-name). Non-up-datable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, bad time-zone, internal error
DELETE	Remove the campus. Marked as deleted and reaped at a later date. **Errors:** not found, client error (un-deletable).

Building Entity(building) / Collection(buildings)

A Building belongs to a parent Campus and has a collection of Rooms. The time zone attribute is inherited by the Building from the parent Campus.

Building Entity XML mapping

This table defines the mapping of building attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.

campus	All	link	Link to the parent Campus.
name	All	string	Required, unique name within the Campus.
time-zone-name	Detailed	string	This field is read-only and inherited from the parent Campus. It is included on "read" operations as a convenience.
rooms	Detailed	link	Link to the room collection. Initially empty at create.

Building Collection Resource URL

Method	URL: {base-uri}/campuses/{campus-id}/buildings[?term="{name-pattern}"]
GET	Retrieve the list of Buildings that are assigned to the Campus. The optional "term" parameter limits the list to buildings who's name is like the term value. **Reply content:** "buildings" collection XML containing "building" summary elements.
POST / PUT	Add a new building to the specified campus resource. **Request Content:** "building" detail XML (name element) **Response Content:** "building" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/buildings[?term="{name-pattern}"]
GET	Retrieve the list of all Buildings. The optional "term" parameter limits the list to buildings who's name is like the term value. **Reply content:** "buildings" collection XML containing "building" summary elements

Building Entity Resource URL

These REST URLs/Methods take action directly on the Building Entity.

Method	URL: {base-uri}/buildings/{building-id}
GET	Retrieve the detailed information on the Building entity. **Reply content:** "building" detailed XML. **Errors:** not found
PUT / POST	Update the specified Building entity. **Request Content:** "building" detail XML. Only need to specify the fields that are being updated (name). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the building. Marked as deleted and reaped at a later date. **Errors:** not found, client error (un-deletable).

Room Entity(room) / Collection(rooms)

A room is a central component of a lecture capture schedule. A room belongs to a parent Building. Rooms are reserved/scheduled for Sections and their associated capture events.

Room Entity's XML mapping

This table defines the mapping of room attributes/fields to its various XML representations (summary or detailed). A room contains many link fields due to it being reserved by Sections and Captures.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
building	All	link	Link to the parent Building.
name	All	string(32)	Required, unique name within the parent Building.
time-zone-name	Detailed	string	This field is read-only and inherited from the grandparent Campus. It is included on "read" operations as a convenience.
target-display-input	Detailed	string	Optional
device-rooms	Detailed	link	Link to fetch devices that are/were(history) used by a room. The typical case is that 1 device is dedicated to a given room. The device-room abstraction allows multiple devices within a room and device swapping for maintenance/upgrades. (see Device Resource)
current-device	Detailed	composite element	For convenience, the current device that is assigned to the room (read-only). If no device is assigned, then the XML element is empty. Otherwise, it contains the "id" and "key" fields of the assigned device.
schedule-rules	Detailed	link	Link to fetch the schedule-rules for this room. schedule-rules contain Capture reservations for Sections, as well as lecture capture events. (see Schedule Rules Resource) Link = "GET {base-uri}/rooms/{room-id}/schedule-rules"

"current-device" XML mapping

The Room's current device composite XML element is described by the below table.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated identifier of the Device.
key	All	string	unique device key used when addressing the device

Room Collection Resource URL

There are various ways to "get" a collection of rooms. For instance, rooms can be filtered by campus, building, or section. The [?term="name-pattern"] URL parameter filters rooms by matching the room name to the name pattern.

Method	URL: {base-uri}/buildings/{building-id}/rooms[?term="{name-pattern}"]
GET	Retrieve the list of Rooms that are assigned to the Building. The optional "term" parameter limits the list to rooms who's name is like the term value. **Reply content:** "rooms" collection XML containing "room" summary elements.
POST / PUT	Add a new Room to the specified Building resource. **Request Content:** "room" detail XML (name, target-display-input elements) **Response Content:** "room" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/campuses/{campus-id}/rooms[?term="{name-pattern}"]

GET	Get all rooms within the specified Campus. Optional "term" parameter limits list to rooms with the matching name pattern. **Reply content:** "rooms" collection XML containing "room" summary elements.

Method	URL: {base-uri}/rooms[?term="{name-pattern}"]
GET	Get all rooms within the caller's organization. Optional "term" parameter limits list to rooms with the matching name pattern. **Reply content:** "rooms" collection XML containing "room" summary elements.

Room Entity Resource URL

Assigning a Device to a Room is performed via the Device Resource. Similarly, the Schedule Rule Resource provides methods to manage the room assigned to the Section's schedule.

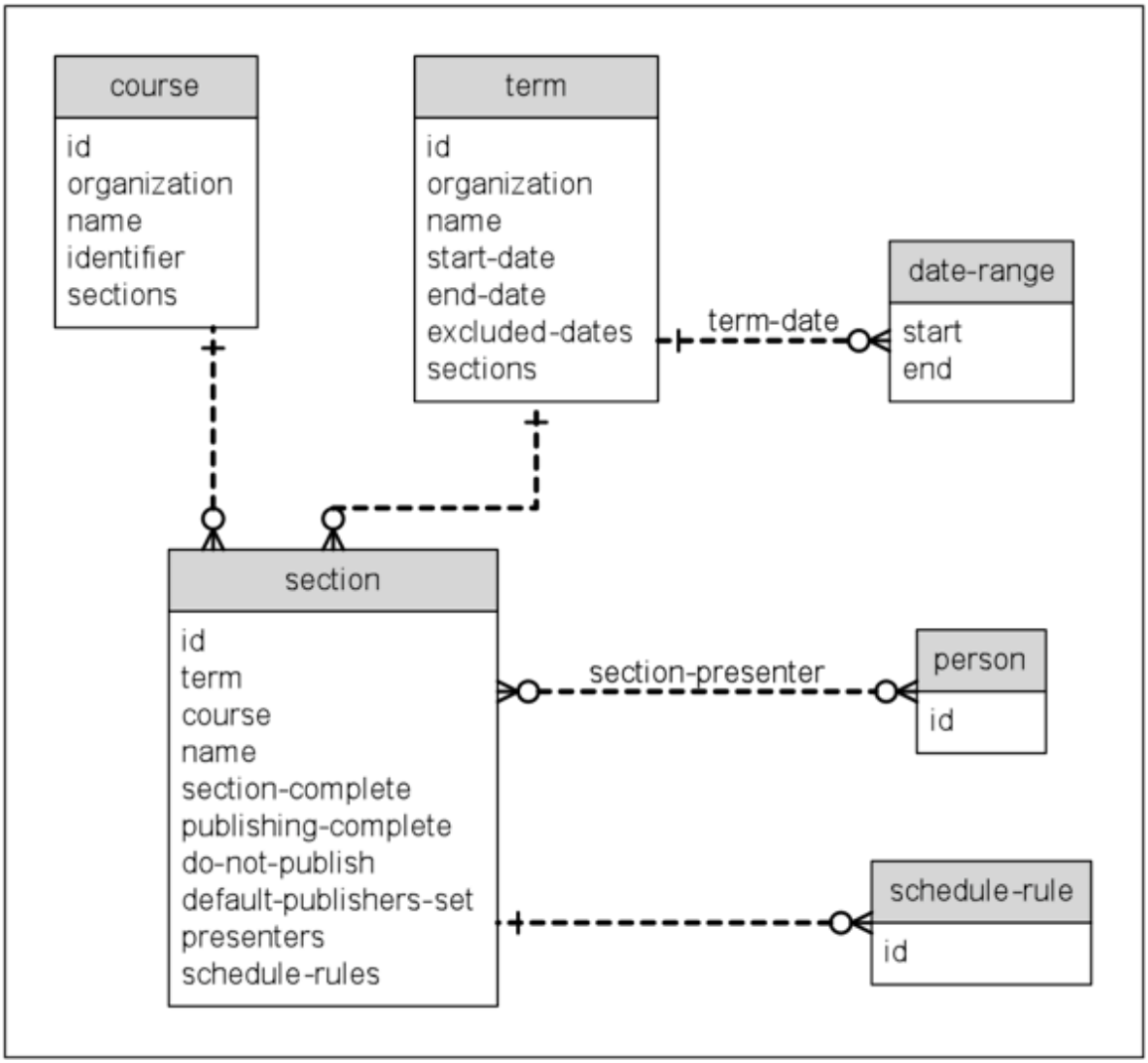
Method	URL: {base-uri}/rooms/{room-id}
GET	Retrieve the detailed information on the Room entity. **Reply content:** "room" detailed XML. **Errors:** not found
PUT / POST	Update the specified Room entity. **Request Content:** "room" detail XML. Only need to specify the fields that are being updated (name, target-display-input). Non-up-datable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Room. Marked as deleted and reaped at a later date. **Errors:** not found, client error (un-deletable).

API Reference - Section Related Resources

In this section:

Section Related API Resources (Term, Course, Section)

The "Section" entity is a required scheduling entity, since lecture captures are scheduled against these sections. The Term and Course provide two separate ways to group the sections. Courses can exist across Terms. These resources and their relationships and attributes are shown in the entity diagram below.



Term Entity(term) / Collection(terms)

A "term" is a period in time in which a collection of courses is offered to students. Typical terms are: Fall, Spring, Summer I, etc.

Term Entity's XML mapping

This table defines the mapping of term attributes/fields to/from it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Term. Auto set to the caller's default organization.
name	All	string(32)	Required, unique name.
start-date	All	iso-date	The date of the first day of the Term. Format: "YYYY-MM-DD"
end-date	All	iso-date	The date of the last day of the Term. Format: "YYYY-MM-DD"

excluded-dates	Detailed	composite collection	An inline collection of date ranges which indicate dates when the Term is not in session. These could be scheduled holiday and break dates. The collection child element is "date-range".
sections	Detailed	link	Link to fetch the collection of Sections offered within the academic term.

"date-range" collection item

The Term's entity contains an inline "excluded-dates" collection; which is a list of "date-range" XML elements. A "date-range" element is a simple container holding a "start", and "end" iso-date element. The format of the date-range element is in the below table.

Field / Element	Rep. Levels	Type (length)	Information
start	All	iso-date	The date of the first day of the date range. Format: "YYYY-MM-DD"
end	All	iso-date	The date of the last day of the date range. Format: "YYYY-MM-DD"

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters terms by matching the term name to the pattern. Don't confuse the "term" URL parameter (which stands for search term) with the "term" entity.

Method	URL: {base-uri}/terms[?term="{name-pattern}"]
GET	Get all Terms within the caller's organization. **Reply content:** "terms" collection XML containing "term" summary elements.
POST / PUT	Add a new Term to the caller's organization. **Request Content:** "term" detail XML (name, start/end date, exclude-dates elements) **Response Content:** "term" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Term Entity Resource URL

Method	URL: {base-uri}/terms/{term-id}
GET	Retrieve the detailed information on the Term entity. **Reply content:** "term" detailed XML. **Errors:** not found
PUT / POST	Update the specified Term entity. If the "excluded-date" element is present in the request content, then the current excluded dates are replaced by the request contents. **Request Content:** "term" detail XML. Only need to specify the fields that are being updated (name, start/end date, excluded-dates). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Term. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Term's Excluded Dates Collection Resource URL

On Term add or update API requests, the caller can include an excluded-dates collection within the uploaded Term XML. If present, the excluded

dates collection defines (on add) or replaces (on update) the current Term excluded dates. This inline collection support is a convenience provided by the API.

The alternate approach is to manage the Term's excluded dates separate from the Term. The table below defines the resource URL calls for managing the exclude dates collection separate from the Term entity.

Method	URL: {base-uri}/terms/{term-id}/excluded-dates
GET	Retrieve the excluded-dates collection for the specified Term. **Reply content:** "excluded-dates" collection that contains "date-range" elements. **Errors:** not found
POST/ PUT	Add a date range to the Term's excluded dates collection. **Request Content:** "date-range" XML. **Response Content:** "date-range" XML (can be ignored) **Errors:** not found, client error(bad data format)
DELETE	Remove all entries from the specified Term's excluded-date collection. **Errors:** not found

Course Entity(course) / Collection(courses)

A Course offered as one or more Sections within various academic Terms.

Course Entity's XML mapping

This table defines the mapping of Course attributes/fields to/from it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Course. Auto set to the caller's default organization.
name	All	string	Required, unique name.
identifier	All	string	identifier known externally by the organization
sections	Detail	link	Link to fetch the collection of Sections offered for this Course.

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters Courses by matching the course name to the pattern.

Method	URL: {base-uri}/courses[?term="{name-pattern}"]
GET	Get all Courses within the caller's organization. **Reply content:** "courses" collection XML containing "course" summary elements.
POST / PUT	Add a new Course to the caller's organization. **Request Content:** "course" detail XML (name, identifier) **Response Content:** "course" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/terms/{term-id}/courses[?term="{name-pattern}"]
GET	Get all Courses within the specified Term. Reply content: "courses" collection XML containing "course" summary elements.

Course Entity Resource URL

Method	URL: {base-uri}/courses/{course-id}
GET	Retrieve the detailed information on the Course entity. **Reply content:** "course" detailed XML. **Errors:** not found
PUT / POST	Update the specified Course entity. **Request Content:** "course" detail XML. Only need to specify the fields that are being updated (name, identifier). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Course. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Section Entity (section) / Collection (sections)

A section is an offering of a course for a particular academic term. Sections are scheduled against rooms and have lecture captures. The section's schedule is a set of associated schedule rules.

Section Entity's XML mapping

This table defines the mapping of section attributes/fields to/from it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
term	All	link	Link to the academic term for this Section.
course	All	link	Link to the Section's course.
name	All	string(32)	Required, unique section name.
section-complete	Detailed	boolean	Indicates if the Section is now complete. If complete, no more captures are scheduled for the section.
publishing-complete	Detailed	boolean	Have all the section captures been published.
do-not-publish	Detailed	boolean	Don't allow captures from the section to be published.
default-publishers-set	Detailed	boolean	Were the default publishers setup for the section.
presenters	Detailed	inline collection	Inline list of presenters associated with the Section. This is a "presenters" collection containing "presenter" items.

schedule-rules	Detailed	link	Link to fetch the schedule-rules for this Section. Schedule-rules contain Capture reservations for Sections, as well as lecture capture events. (see Schedule Rules Resource)

"presenter / person" inline collection item

The Section entity contains an inline "presenters" collection; which is a list of "presenter" XML elements. A "presenter" element is a simple container holding a "person" reference element. The format of the nested "person" xml element is in the below table.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	ID of the person which is a used as a "presenter.
first-name	All	string	The person's first name.
last-name	All	string	The person's last name.

Section Collection Resource URL

There are various ways to "get" a collection of Sections. The [?term="name-pattern"] URL parameter filters sections by matching the section name to the name pattern.

Method	URL: {base-uri}/ terms/{term-id}/courses/{course-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Course and Term. The optional "term" parameter limits the list to Sections matching on the name attribute. **Reply content:** "sections" collection XML containing "section" summary elements.
POST / PUT	Add a new Section to the specified Term and Course. **Request Content:** "section" detail XML (name, retention parameters, presenters) **Response Content:** "section" summary XML ("id" element holds new ID) **Errors:** duplicate name, course/term not found, client error (content invalid)

Method	URL: {base-uri}/courses/{course-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Course. The optional "term" parameter limits the list to Sections matching on the name attribute. **Reply content:** "sections" collection XML containing "section" summary elements.

Method	URL: {base-uri}/terms/{term-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Term. The optional "term" parameter limits the list to Sections matching on the name attribute. Reply content: "sections" collection XML containing "section" summary elements.

Section Entity Resource URL

Below are the Restful methods for managing a specified Section Entity.

Method	URL: {base-uri}/sections/{section-id}
GET	Retrieve the detailed information on the Section entity. **Reply content:** "section" detailed XML. **Errors:** not found

PUT / POST	Update the specified Section entity. **Request Content:** "section" detail XML. Only need to specify the fields that are being updated (retention attributes, presenters). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Section. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Section Presenters Collection URL

The section presenters can be specified within the Section entity XML, or via the following presenter URLs.

Method	URL: {base-uri}/sections/{section-id}/presenters
GET	Retrieve the list of section presenters for the specified Section. **Reply content:** "presenters" collection of "presenter" elements. **Errors:** not found
DELETE	Clear the Section's presenter list. **Errors:** not found

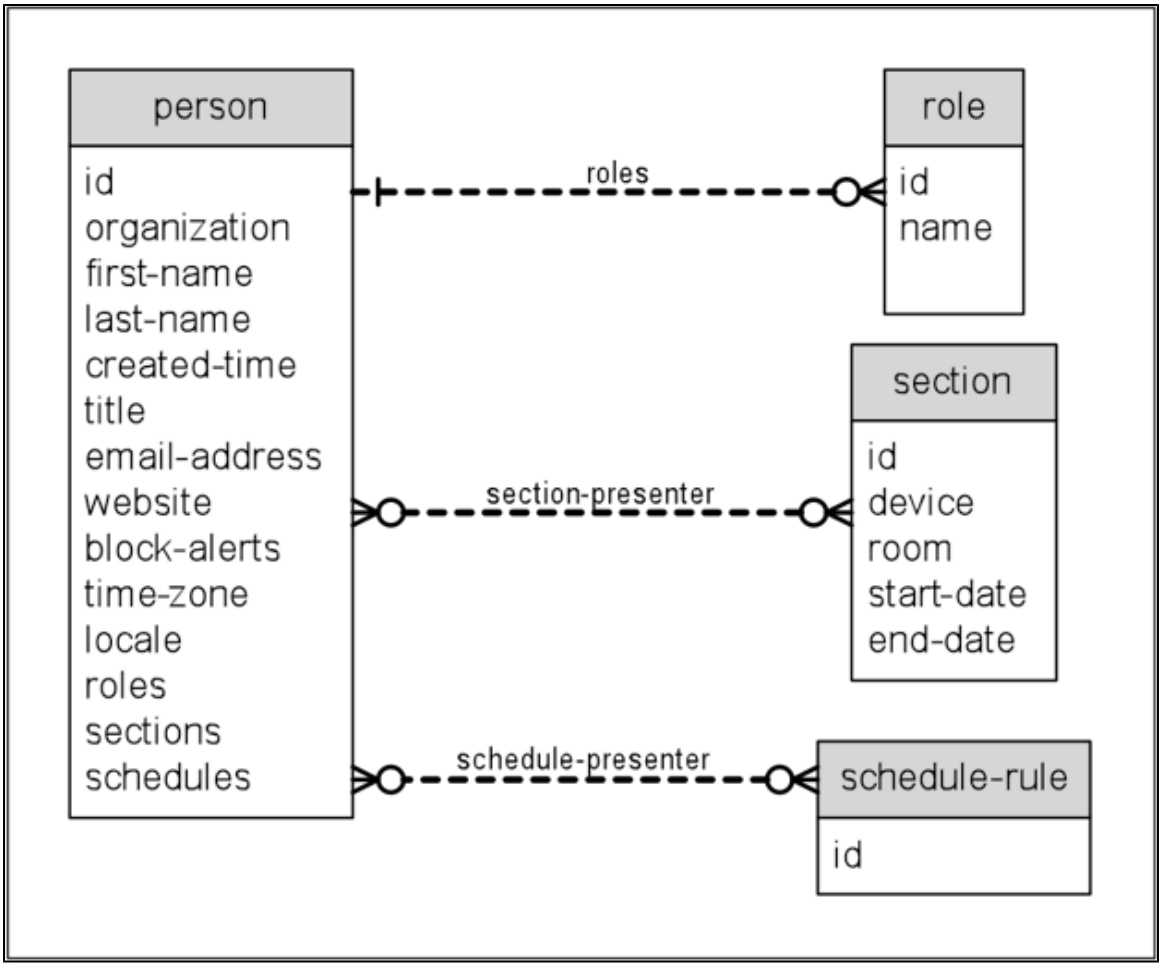
Method	URL: {base-uri}/sections/{section-id}/presenters/{person-id}
POST / PUT	Add the person to the Section's presenter list. **Request Content**: Empty **Response Content:** Empty **Errors:** not found
DELETE	Remove the Person from the Section's presenter list. **Errors:** not found

API Reference - Person Related Resources

In this section:

Person/User Related API Resources (Person, Role)

The "Person" entity with a "Presenter" role is a required scheduling entity. This defines the instructor or presenter for the course or offering being captured. In the EchoSystem Server (ESS) user interface, "persons" are referred to as "users". These resources and their relationships and attributes are shown in the entity diagram below.



Person Entity (person) / Collection (people)

A person in the Echo System can have one or more roles. Predefined user roles are presenter, and administrator. Presenters can be assigned to course sections.

Person Entity's XML mapping

This table defines the mapping of person attributes/fields to/from it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Person. Auto set to the caller's default organization.
last-name	All	string	Required Person's last name.
first-name	All	string	Required, Person's first name.
created-name	Detailed	iso-time	Auto populated with the creation time stamp.
title	Detailed	string	Optional formal title of the Person.

email-address	Detailed	string	Required email address for the Person.
website	Detailed	string	Optional URL for the person's reference website.
time-zone-name	Detailed	string	Optional time zone of the Person.
locale	Detailed	string	Optional Locale of the Person.
block-alert	Detailed	boolean	Relates to system alerts and requires e-mail address when true. Default is false.
roles	Detailed	roles collection	List of roles assigned to the person. Inline collection of "roles" containing "role" names. Inline since role entity is a simple structure.

"roles" inline collection XML mapping

The Person's assigned roles is a simple collection element called "roles"; which contains "role" names.

Field / Element	Rep. Levels	Type (length)	Information
role	All	string	Short or long role name. For example: "admin" or "role-admin" are the short/long role names for the system administrator role.

People Collection Resource URL

The [?term="name-pattern"] URL parameter filters people by matching the person's name to the name pattern.

Method	URL: {base-uri}/people[?term="{name-pattern}"]
GET	Get all people within the caller's organization. The optional "term" parameter limits the list to matching on the name (last, first) attribute. **Reply content:** "people" collection XML containing "person" summary elements.
POST / PUT	Add a new Person. **Request Content:** "person" detail XML **Response Content:** "person" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Person Entity Resource URL

Below are the Restful methods for managing a specified Person Entity.

Method	URL: {base-uri}/people/{person-id}
GET	Retrieve the detailed information on the Person entity. **Reply content:** "person" detailed XML. **Errors:** not found

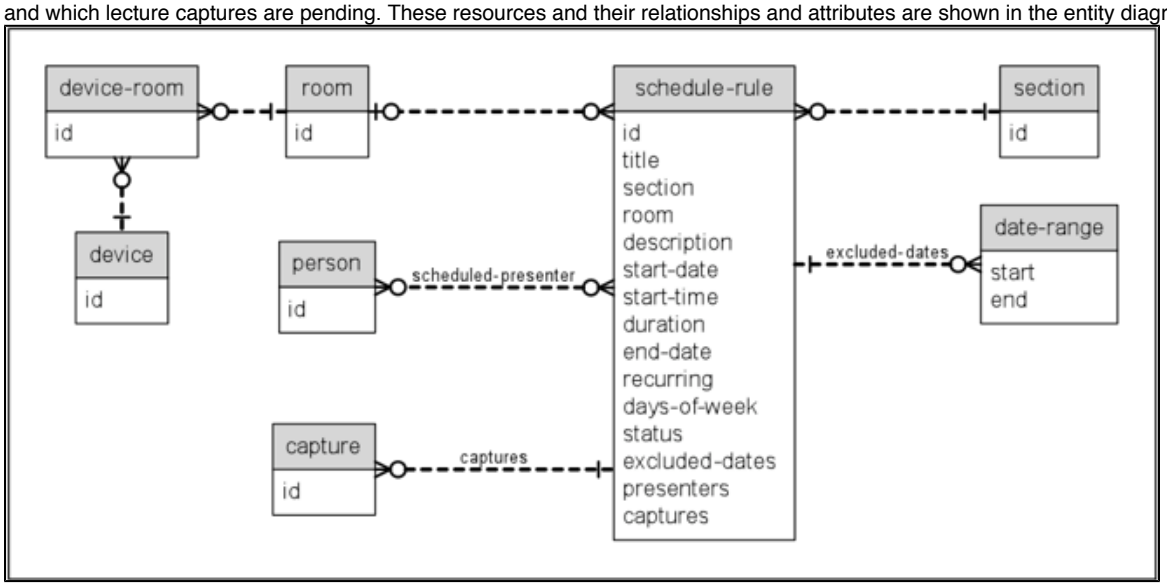
PUT / POST	Update the specified Person entity. **Request Content:** "person" detail XML. Only need to specify the fields that are being updated. Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Person. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

API Reference - Schedule Related Resources

In this section:

Schedule Related API Resource (Schedule Rule)

The central schedule related entity is the Schedule Rule. This entity specifies when a section meets, in which room, the presenters/instructors, and which lecture captures are pending. These resources and their relationships and attributes are shown in the entity diagram below.



Schedule Rule Entity (schedule-rule) / Collection (schedule-rules)

Schedule Rule Entity's XML mapping

This table defines the mapping of Schedule Rule attributes/fields for it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
section	All	link	link to the parent Section offering
room	All	link	Link to the room assignment for capturing the section. Note: the rule can have an unassigned room; which results in an absent room link.
title	All	string	Required name/title for the rule. Useful for searching.

status	All	string	Enumeration constant: (draft, active, complete, trashed)
description	Detail	string	Optional text description of the rule.
start-date	Detail	iso-date	The starting period for this rule. Can be empty if the rule is still in "draft" state. Format: "YYYY-MM-DD"
end-date	Detail	iso-date	If this is a recurring capture, then the end date of the rule is required. Format: "YYYY-MM-DD"
start-time	Detail	iso-time	The starting period for this rule. Can be empty.
duration-seconds	Detail	int	Each capture is scheduled to run this many seconds.
recurring	Detail	boolean	Is this rule a one time capture event or does it capture multiple events?
days-of-week	Detail	composite element	Group of Booleans that indicate which days of the week the recurring capture is performed. The field is empty if the recurring bit is false.
exclued-dates	Detail	date-range collection	Inline list of "date-range" elements representing dates when the capture should not take place. For example breaks or holidays. See Term Entity's "excluded-dates".
target-display-input-override	Detail	string	Optional setting for the capture input dimensions generated by this rule. For example: "640x720".
presenters	Detail	presenter collection	Inline list of presenters for the capture events. See Section Entity's "presenters".
captures	Detail	link	Link to the Capture Events generated by this schedule rule.

"schedule-rule/days-of-week" XML mapping

This table defines the mapping of the days-of-week data structure to its XML representations.

Field / Element	Rep. Levels	Type (length)	Information
sunday	All	boolean	True if the capture recurs every Sunday
monday	All	boolean	True if the capture recurs every Monday
tuesday	All	boolean	True if the capture recurs every Tuesday
wednesday	All	boolean	True if the capture recurs every Wednesday
thursday	All	boolean	True if the capture recurs every Thursday
friday	All	boolean	True if the capture recurs every Friday
saturday	All	boolean	True if the capture recurs every Saturday

Schedule Rule Entity REST Method/URL mapping

The following table describes the Restful Methods supported by the Schedule Rule Resource.

Method	URL: {base-uri}/sections/{section-id}/schedule-rules[?term="name-pattern"]
GET	Get all Rules associated with the specified Section offering. **Reply content:** "schedule-rules" collection XML containing "schedule-rule" summary elements. **Errors:** Section not found.
POST / PUT	Add a new Schedule Rule to the Section. **Request Content:** "schedule-rule" detail XML **Response Content:** "schedule-rule" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}
GET	Retrieve the detailed information on the Schedule Rule. Reply content: "schedule-rule" detailed XML.. Errors: not found.
PUT / POST	Update the specified Schedule Rule. **Request Content:**"schedule-rule" detail XML (description, start-date, start-time, end-date, duration-seconds, target-display-override, recurring, days-of-week, exclude-dates, status, presenters). Note: Only the fields that have changed are required to be present. **Errors:** duplicate name, client error (content invalid)
DELETE	Remove the schedule rule. May not be allowed given the schedule rule status. **Errors:** not found

Schedule Rule "room" assignment Resource URL

On Schedule Rule add or update API requests, the caller can include the assigned room within the uploaded entity XML. The alternate approach is to manage the room assignment operation separate from the Schedule Rule entity update operation. The table below defines the resource URL calls for directly managing the room assignment.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/rooms
GET	Retrieve a "rooms" collection for the specified Schedule Rule. The collection will contain zero or one room summary entity. **Reply content:** "rooms" collection that contains zero/one "room" element. **Errors:** Schedule Rule was not found.
POST / PUT	Not supported
DELETE	Set the room assignment to empty/null. **Errors:** Schedule Rule was not found.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/rooms/{room-id}
GET	Check if the given Room is already assigned to the specified Schedule Rule. Returns **200** if the room is already assigned to the rule. Returns **400(not found)** if the room is not assigned to the rule. Returns **400(not found)** if the Schedule Rule or Room are not found.
PUT / POST	Assign the given room to the specified schedule-rule. **Errors:** Schedule Rule or Room was not found.

DELETE	Remove the room from the Schedule Rule. **Errors:** Schedule Rule or Room was not found, or the Room was not assigned to the rule.

Schedule Rule "excluded-dates" Collection Resource URL

On Schedule Rule add or update API requests, the caller can include an excluded-dates collection within the uploaded entity XML. If present, the excluded dates collection defines (on add) or replaces (on update) the current Schedule Rule's excluded dates. This inline collection support is a convenience provided by the API.
The alternate approach is to manage the excluded dates separate from the Schedule Rule entity. The table below defines the resource URL calls for managing the exclude dates collection.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/excluded-dates
GET	Retrieve the excluded-dates collection for the specified Schedule Rule. **Reply content:** "excluded-dates" collection that contains "date-range" elements. **Errors:** Schedule Rule was not found.
POST / PUT	Add a date range to the Schedule Rule's excluded dates collection. **Request Content:** "date-range" XML. **Response Content:** "date-range" XML (can be ignored) **Errors:** not found, client error(bad data format)
DELETE	Remove all entries from the specified Schedule Rule's excluded-date collection. **Errors:** not found

Schedule Rule "presenters" Collection URL

The Schedule Rule's presenters can be specified within the "schedule-rule" entity XML, or via the following presenter URLs.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/presenters
GET	Retrieve the list of presenters for the specified Schedule Rule. **Reply content:** "presenters" collection of "presenter" elements. **Errors:** Schedule Rule was not found.
DELETE	Clear the Schedule Rule's presenter list. **Errors:** Schedule Rule was not found.

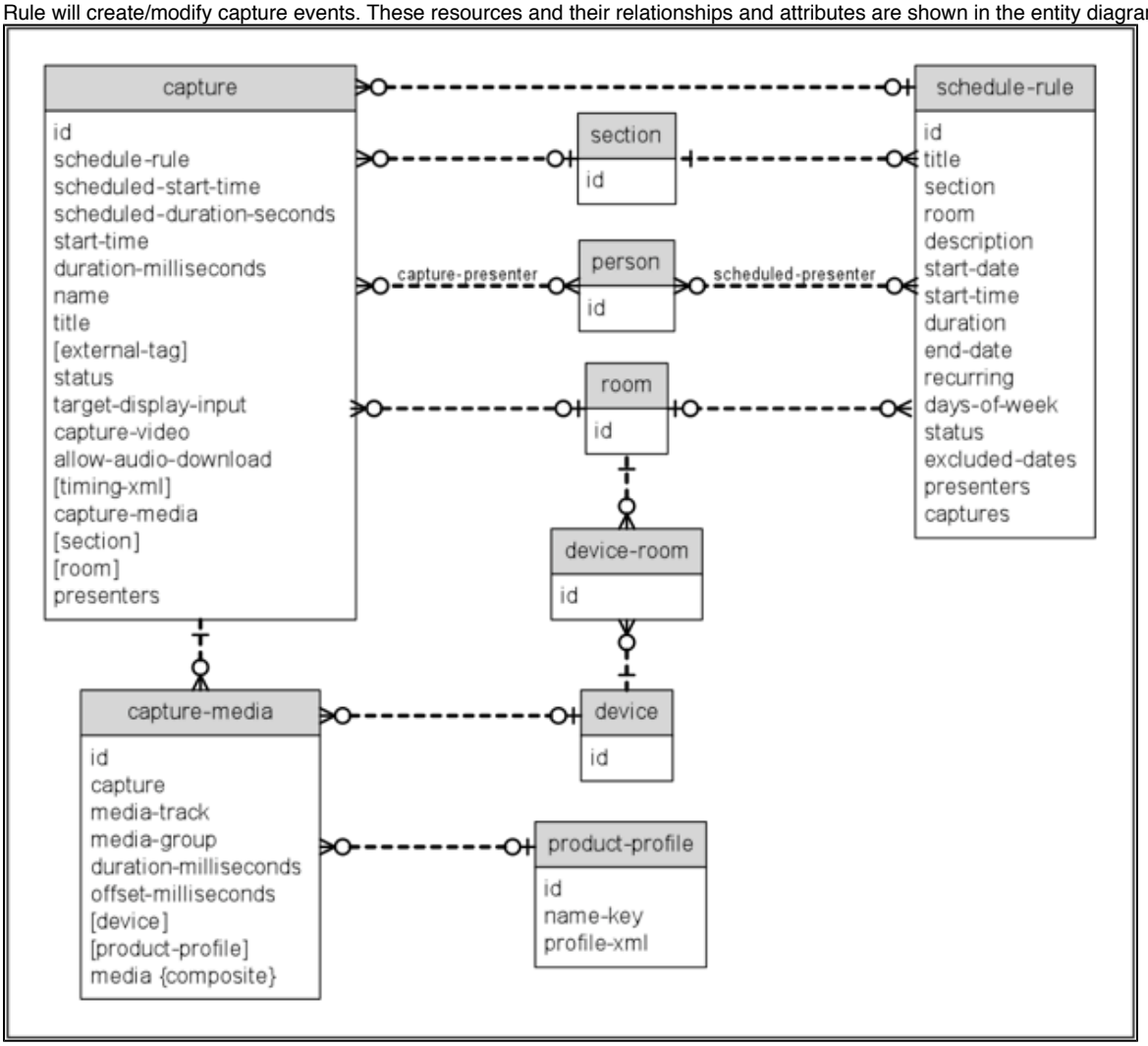
Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/presenters/{person-id}
POST / PUT	Add the person to the Schedule Rule's presenter list. **Request Content:** Empty **Response Content:** Empty **Errors:** Schedule Rule was not found.
DELETE	Remove the Person from the Schedule Rule's presenter list. **Errors:** Schedule Rule was not found.

API Reference - Capture Related Resources

In this section:

Capture Related API Resources

The Capture related entities are only available as read-only entities from the Scheduling API. They represent the capture history as well as the future scheduled capture events. The Scheduling Rule is the updatable entity that control scheduled capture events. Changes to the Schedule Rule will create/modify capture events. These resources and their relationships and attributes are shown in the entity diagram below.



Capture Entity (capture) / Collection (captures)

One or more capture entities are created when a Schedule Rule is entered into the Echo360 System. The Captures represent each instance of a lecture capture. The Scheduling API only allows read access to Capture records. Changes made to the parent Schedule Rule will update the related (future) Captures.

Capture Entity's XML mapping

This table defines the mapping of Capture Event attributes/fields for it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier
organization	All	link	Link to the parent organization for this Capture. Auto set to the caller's default organization.
schedule-rule	All	link	link to the parent Schedule Rule that created this capture event.

name	All	string(116)	Name used for search.
scheduled-start-time	All	iso-datetime	When the event is/was scheduled to start.
scheduled-duration-seconds	All	int	The scheduled length of the capture event.
start-time	Detail	iso-datetime	The actual start date/time of the captured event. This is set to the scheduled start time if this is a future event.
duration-milliseconds	Detail	int	The actual length of the captured event.
title	Detail	string(255)	Capture event title.
external-tag	Detail	string(50)	Optional field set to an external system's tag property.
status	Detail	string	Status enumeration value.
target-display-input	Detail	string(30)	–
capture-video	Detail	boolean	Should the event capture the video feed.
allow-audio-download	Detail	boolean	Is the audio only allowed to be downloaded.
timing-xml	Detail	CLOB	–
capture-medias	Detail	link	link to the list of media files (media data) created by this capture event.
[section]	Detail	link	In order to support event data archive, the Section captured is placed in its own capture field.
[room]	Detail	link	In order to support event data archive, the Room captured is placed in its own capture field.
[presenters]	Detail	inline-collection	In order to support event data archive, the Presenters captured is placed in its own collection field.

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters Capture by matching the Capture's name attribute to the pattern.

Method	URL: {base-uri}/schedule-rule/{schedule-rule-id}/captures[?term="{name-pattern}"]
GET	Get all Captures associated to the specified Schedule Rule. **Reply content:** "captures" collection XML containing "capture" summary elements. **Errors:** schedule-rule was not found.

Method	URL: {base-uri}/rooms/{room-id}/captures[?term="{name-pattern}"]
GET	Get all Captures associated with a given room. **Reply content:** "captures" collection XML containing "capture" summary elements. **Errors:** room was not found.

Method	URL: {base-uri}/captures/{capture-id}
GET	Retrieve the detailed information on the Capture event. **Reply content:** "capture" detailed XML. **Errors:** not found

Capture Media Entity (capture-media) / Collection (capture-medias)

A single Capture Event can generate multiple media files. Each generated file is represented by the Capture Media Entity. As with the Capture event entity, the Scheduling API only allows read access to the Capture Media records.

Capture Media Entity's XML mapping

This table defines the mapping of Capture Media attributes/fields for it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
capture	All	link	link to the parent Capture Event that created the media.
media-track	All	string	Enumerated constant that indicates how this media is used. values (audio, audio-vga, audio-video, flash-movie, flipbook, thumbnail, vga, video)
media-group	All	string	Enumerated constant that indicates the major media group for this media. values (edit-proxy, master, primary, projector).
duration-milliseconds	Detail	int	Running length of the media in milliseconds.
offset-milliseconds	Detail	int	–
device	Detail	link	Link to the Device which generated/captured this media.
product-profiles	Detail	inline collection	Inline list of "product-profile" items. These are the Product Profiles used when generating the media file.
media	Detail	composite element	Detailed media attributes of the raw file.

"media" composite element XML mapping

This table defines the mapping of Media attributes/fields for it's XML representation. A Media entity contains information on a capture generated media file.

Field / Element	Rep. Levels	Type (length)	Information
path	All	string	File path of the media.
type	All	string	The media file type: jpg, mp3, aac, m4b, rm, mov, wmv, flv, swf, m4v, h264, mp4
hash	All	composite element	Hash value of the associated capture media file.

file-size-bytes	All	long	Bytes size of the associated media file.
technical	All	composite element	Grouping of file capture technical stats.
tag	All	composite element	Grouping of information attributes about the media contents.

"media/hash" composite element XML mapping

This table defines the mapping of Hash attributes/fields for its XML representation. Each file generated by a capture has an associated hash value that helps verify the file integrity after file transfer.

Field / Element	Rep. Levels	Type (length)	Information
algorithm	All	string	An enumerated string that indicates the hash algorithm used to generate the hash vale. (Adler32, CRC32, MD2, MD5, SHA_1, SHA_256, SHA_384, SHA_512)
value	All	string	The actual hash value generated for the associated capture media file.

"media/technical" XML mapping

This table defines the mapping of media-technical attributes/fields to its XML representations.

Field / Element	Rep. Levels	Type (length)	Information
frame-rate	All	numeric	frame rate of the captured media.
bit-rate	All	int	Captured bit-rate of the media.
width	All	int	line/pixel width of the media frames.
height	All	int	line/pixel height of the media frames.
file-server	All	string	–

"media/tag" XML mapping

This table defines the mapping of media tag attributes/fields to its XML representations.
Media Tag is a collection of metadata attributes that can be assigned to a captured media. The "tag" attributes are set/edited by a Echo360 user via the presentation UI component.

Field / Element	Rep. Levels	Type (length)	Information
year	All	int	Optional year when the content was produced/recorded
comment	All	string	Optional text describing the content
copyright	All	string	Optional copyright statement covering the content.
album	All	string	Optional album name used to group content.
artist-name	All	string	Optional person who produced the content.

genre	All	string	Optional category of the content.

REST Method/URL mapping

The following tables describe the Restful Methods supported by the Capture Media Resource.

Method	URL: {base-uri}/captures/{capture-id}/capture-medias
GET	Get all Media associated with the specified Capture Event. **Reply content:** "capture-medias" collection XML containing "capture-media" summary elements. **Errors:** Capture Event not found.

Method	URL: {base-uri}/capture-medias/{capture-media-id}
GET	Retrieve the detailed information on the Capture Media. **Reply content:** "capture-media" detailed XML.. **Errors:** not found.

Product Profile Entity (product-profile) / Collection (product-profiles)

A Product Profile contain the specific instructions and settings for generating "products" for Capture Events. The system comes with a predefined set of profiles. The scheduling API only allows read access to the Product Profiles.

Product Profile Entity's XML mapping

This table defines the mapping of Product Profile attributes/fields for it's various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
name-key	All	string	Unique profile name.
product-profile-xml	Detail	CLOB	The profile XML.

REST Method/URL mapping

The following tables describe the Restful Methods supported by the Product Profile Resource.

Method	URL: {base-uri}/product-profiles[?term="{name-pattern}"]
GET	Get all Product Profiles. **Reply content:** "product-profiles" collection XML containing "product-profile" summary elements. **Errors:**

Method	URL: {base-uri}/product-profiles/{product-profile-id}

GET	Retrieve the detailed information on the specified Profile. **Reply content:** "product-profile" detailed XML.. **Errors:** not found.

API Reference - Known Limitations

In this section:

Section and Schedule Configuration Options

There are currently two configuration options for sections/schedules that are available in the user interface (UI) but not yet available via the application programming interface (API).

The first is **Products**. Products define the types of student viewable content the system will generate for the capture. There are four product types, each with 2 options. Products cannot be defined for a section or schedule rule via the API. They can however be set as global default values and inherited by new sections and/or schedules created via the API.

The second is **Publishers**. Publishers enable the posting of links or media files into a Course Management System / Learning Management System / Virtual Learning Environment system (Blackboard, Moodle, iTunes U, etc). While publishers cannot currently be configured via the API, they can be inherited from the global defaults when new sections are created via the API. In addition, CMS-IDs cannot currently be defined for a publisher via the API. CMS-IDs are the identifiers for a course CMS/LMS/VLE. The IDs can however be defined via the EchoSystem Server (ESS) UI after creating the section via the API. Once defined for the section, all new schedules for that section will inherit the value. So in effect, there is one manual step to properly set up publishing when using the API.

Device Room Assignment

The API does not currently support assigning a device to a room, changing a device room assignment or licensing a room.

2.6 Schedule API

In this section:

Introduction

EchoSystem 2.5 introduced the first stage on its Server API, starting with the necessary requirements for scheduling related integrations. The API enables programmatic EchoSystem scheduling via a RESTful interface. This guide is the initial draft of the API reference.

The EchoSystem Server (ESS) is the central component of the EchoSystem platform. It stores all data for capture schedules, manages user access to data contained within the system, as well as controls the capturing and processing activities performed by the related capture devices and media processors.

Some of the data contained within the ESS is also contained within other systems in the university environment. Similarly, some of the data contained within the ESS is often related to information contained within other systems. For this reason it may be desirable to more closely integrate the ESS with these related systems.

The API is an extension to the ESS application that provides external systems direct access to query, update and delete the data contained within the ESS.

An example of a third party application that would make use of the API is a timetabling system where data is automatically create capture schedules based on lecture timetables, thereby integrating the EchoSystem with their existing workflow. Another example would be an online booking form that enables Faculty/Academics to request their lectures to be captured. Such a web form might have a backend that integrates with the ESS to add the capture schedules dynamically.

REST Overview

Introduction

In many ways, REST (Representational State Transfer) is a deliberate return to the basics that made the World Wide Web itself successful. The web arguably became successful because it was conceptually simple, yet powerful.

In its purest form, there were web pages that had addresses (URLs). Web pages could be viewed using a web browser, which used HTTP (the Hypertext Transfer Protocol) to retrieve the web pages. Furthermore, web pages could have links, which allowed users to traverse relationships between web pages.

In Echo360's implementation of RESTful web services, the conceptual basis of the API is resources, rather than web pages. A resource is anything that is important enough to be referenced as a distinct "thing." Rooms are resources, as are presenters, schedules, etc. Accordingly, Echo360's approach for web services may also be referred to as a resource-oriented architecture.

Like web pages, each resource has a distinct URL from which information about the resource can be retrieved. Unlike web pages, it is possible to perform actions related to resources, such as creating them, updating them, deleting them, etc. All of the actions that can be reasonably performed on a resource can be accomplished using the HTTP protocol.

HTTP Methods

The HTTP standard defines several different "methods" for performing operations on resources:

- GET - Retrieve a resource.
- POST - Create a new resource.
- PUT - Update a resource.
- DELETE - Delete a resource.

The GET and POST methods are already widely used by web browsers for retrieving web pages and for posting data from forms. The PUT and DELETE methods have not had widespread usage until recently, despite being specified in the HTTP standard more than twelve years ago. The advent of RESTful web services has changed this; their usage is rapidly becoming common.

RESTful Operations

The four HTTP methods (GET, POST, PUT and DELETE) can be combined with distinct URLs to provide a web services interface that allows users to perform operations on a resource.

Table 1: An Example of RESTful Operations

URL	HTTP Method	Operation
http://example.com/rooms	GET	Get a list of all rooms
http://example.com/buildings/123/rooms	GET	Get a list of rooms within building "123"
http://example.com/buildings/123/rooms	POST	Create a new room within building "123"
http://example.com/rooms/1	GET	GET detailed data on room "1"
http://example.com/rooms/1	PUT	Update data on room "1"
http://example.com/rooms/1	DELETE	Delete room "1"

In **Table 1**, the various URLs allow callers to perform actions on a room or, more specifically, the metadata associated with a room.

The primary advantage of this architecture is that it offers a regular and predictable interface for managing most types of resources. For developers, using a RESTful web service is as simple as sending a request to a specific URL using the appropriate HTTP method. HTTP even supports the capability to send arguments with a request.

Representation Formats and HTTP Status Codes

Each web service request sent via HTTP results in a response being sent back to the caller. At a minimum, the caller receives an HTTP status code that defines the status of the request. HTTP defines numerous standard status codes that can easily be leveraged by a web service to provide meaningful information to a caller, e.g. - 404 ("Not Found").

Both the request and the response may optionally include content. For example, when creating the metadata for a new schedule rule, the request will need to include information about the schedule. Most web services accept such content in easy?to?parse formats such as form?encoded

parameters (the format for data that is POSTed from a traditional HTML form), XML and JSON. That data format is referred to as the representation format of the request.

Likewise, most web services respond to a request by sending back content in easy-to? parse formats such as JSON, XML or various flavors of XML such as Atom, RSS or XHTML. That data format is referred to as the representation format of the response. Callers can parse the content provided in the response and reuse it for their own purposes.

Many web services allow users to select from a choice of representation formats for the request and the response. For example, a caller might send a request that included JSON content and receive a response that included Atom content. Currently, the EchoSystem Scheduling API accepts XML content and produces XML responses.

REST vs. SOAP

REST and SOAP represent two very different strategies for implementing web services. A RESTful architecture makes web services as conceptually simple as possible, with a standard approach for representing operations on resources. REST also makes no assumptions about the representation formats, such as XML, used for content that may be sent back to callers.

SOAP is more elaborate than REST in many ways. There are numerous official standards layered on top of each other, providing features for remote procedure calls, security, etc. There is no common way of representing operations on resources. Instead, there is a standard approach for representing any type of content in XML in a generic fashion. With SOAP interfaces, the strategy is to have libraries that automatically parse SOAP?related XML and generate objects based on the content, which can then be interrogated to retrieve data elements as needed.

In simplistic terms, REST focuses on providing a standard interface for resources, while SOAP focuses on providing a standard and generic representation for content. With REST, developers may have to create custom parse logic for content received from a RESTful web service. With SOAP, content can be parsed automatically and made available to the caller. In practice, REST representation formats tend to be much simpler than SOAP?related XML, so custom parsing is not typically a problem.

In general, RESTful web services are usually simpler and easier to use than SOAP interfaces. Additionally, SOAP interfaces often do not work well in cross-language environments, e.g. - SOAP clients in languages such as Perl, Ruby, Python and other languages may not work properly with SOAP services. RESTful interfaces are usually less complex than SOAP interfaces, so language compatibilities are less of a problem.

Most importantly, implementing SOAP interfaces can be much more time consuming than implementing RESTful interfaces, particularly in a language such as Java.

For these reasons, Echo360 has selected REST as the basis for this API.

REST References

The following references may be useful in understanding REST:
• Restful Web Services, 2007 O'Reilly, by Leonard Richardson and Sam Ruby
• "The Future of Web Services", Online Video presentation by Gregg Pollack, http://www.railsenvy.com/2007/12/17/the?future?of?web?services

API Reference

This section provides detailed documentation for all of the actions made available by Echo360's RESTful Scheduling API.

The Basics

Common symbols

Curly Braces {}: This document encloses symbolic variables within curly braces. The API user will substitute the variable with a concrete value. For example: "GET /buildings/{building-id}/rooms/{room-id}" indicates the caller should provide a concrete Building ID and Room ID when issuing the stated HTTP GET.

Square Brackets []: Items enclosed in square brackets indicate optional parameters. For example: " GET /buildings/{building-id}/rooms[?term="{name-pattern}"] " indicates that the list rooms query supports an option parameters called "term" which takes a name pattern value.

{base-uri}: Within this document, the {base-uri} symbol represents the common or root HTTP URL used by all API requests.
The actual base-uri will be determined by the system configuration.

- An example {base-uri} is "https://api.echo360.com:8443/ess/restapi/v1"

{entityName-id}: Many of the API request URLs require the caller to specify one of more entity ids. These ids are GUID strings. The documentation will simply use the {entityName-id} symbol to indicate that the caller provides an entity id.
Representations of entities are transferred by the scheduling RESTful API. Three standard "levels" of entity representation (XML) are supported by the API; detailed, summary, and link. These levels are referred to in the API reference tables as "Rep. Levels".

- Link: The "link" representation is the corresponding URL that will retrieve an entity's "detailed" representation. Links usually refer to associated parent or child entities. For instance: a "detailed" building representation will contain a link to its rooms. The XML element

"link" along with its attributes "href", "rel", and "title", define the link representation.
- Example: <link href="{base-uri}/buildings/{building-id}/rooms" rel="http://schemas.com.echo360.rooms" title="rooms"/>

There can be optional content for the link element. This is expected to be a short description of the link target.

- Detailed: The "detailed" representation contains all fields of a given attribute. This representation is provided on "POST" requests and "GET" entity responses. The top level XML element will be the entity name; such as "room". Its children elements map to the entity's properties/attributes. For example:

```
<room>
<id>938ab3edf432047a</id>
<link href="\{base-uri\}/buildings/1939abdf982efac312"
rel="http://schemas.com.ech360.building" title="building"/>
<name>washington-101</name>
</room>
```

Summary: When lists of entities are requested, the API response is a list of entity summaries. Each summary contains a link to the detailed entity. This special link uses a "rel=self" link attribute.

Basic API Response

Following the RESTful API pattern, the EchoSystem Scheduling API returns HTTP response headers with a XML content/body. The returned XML is either a "collection" of summary entities or a single detailed "entity" XML element. Typically, a given collection element is simply plural name of the entities within the collection.

Entity vs. Entity Collection: Each entity operated upon by the API has an associated entity collection. For instance: a "room" entity has an associated "rooms" collection.

Following the RESTful pattern, entities are accessed from their associated collection.

So a request for a specific room is addressed relative to the rooms collection. This pattern shows up in the API request URL. The basic URL pattern is "{base-uri}/{collection-name}/{entity-id}"

- Example: "GET {base-uri}/rooms/{room-id}" returns the "detailed" representation of the given room.

The associated collection name is the English plural of the entity name. Collections are returned by the API when the URL does not specify an entity id.

- Example: "GET {base-uri}/rooms" returns a collection of all rooms within the organization.

For instance the "rooms" collection is:

```
<rooms>
<total-results>25</total-results>
<items-per-page>25</items-per-page>
<start-index>0</start-index>
<\!-\- each room element is the "summary" representation of the entity \-->
<room><room/>
<room><room/>
<room><room/>
...
</rooms>
```

Authentication and Authorization

EchoSystem Server Scheduling API utilizes OAuth for handling system authentication and authorization. The implementation is similar to that implemented by NetFlix® .

Authentication is the process of verifying who sent a given API request. It also includes steps to verify that the request was not altered by an intermediate party. Note: Authentication does not address privacy of the data being transmitted.

The Echo360 System requires each API request to be "signed" by an API User ID and shared secret. The OAuth standard defines how to sign the HTTP API requests; including the format of the HTTP Authorization header field.

For the Scheduling API the User is referred to as a "Trusted System". Each trusted system must register with the Echo360 System via the Administration User Interface page at ([https://{ess-uri}/ess/security/ListTrustedSystems.html). The registration process generates a Trusted System ID and a shared secret. The Trusted system then signs all their associated requests with these credentials. The third party system needs

to secure the credentials; to prevent another party from impersonating the trusted system.

Authorization is the process of determining if a given Authenticated API user is allowed to perform the request action. Many systems define user roles which restrict the action that a user can take. Currently, the Scheduling API does not restrict the actions of an Authenticated API user.

Data Privacy
The data transmitted between the external system and the Echo360 System may be sensitive or deemed private. HTTP transmits data "in the clear"; while HTTPS encrypts all data transmitted. The Echo360 System supports both HTTP and HTTPS for scheduling API requests.
OAuth provides encryption of the authentication signature, but does not address data privacy. Using OAuth with HTTP will protect requests from tampering. The API User should use HTTPS if they want to protect their data privacy.

OAuth Specifics
The Scheduling API uses a simplified two-step OAuth authorization process. The OAuth standard also supports a more complex three-step process that uses session based request/auth tokens along with the consumer key/secret. The three-step process is targeted to internet hosted services that are accessed by many diverse consumer clients.
In the simplified two-step scheme, the consumer needs to register with the Echo360 System, in order to be assigned a unique consumer key and shared secret. These credentials are long lived and not limited to a session timeout. The consumer does not need to request a session token/secret (OAuth request & auth tokens/secrets) before initiating a request to the Echo360 System. Instead, the consumer just uses the consumer key/secret and an empty auth token/secret to sign all scheduling API requests.

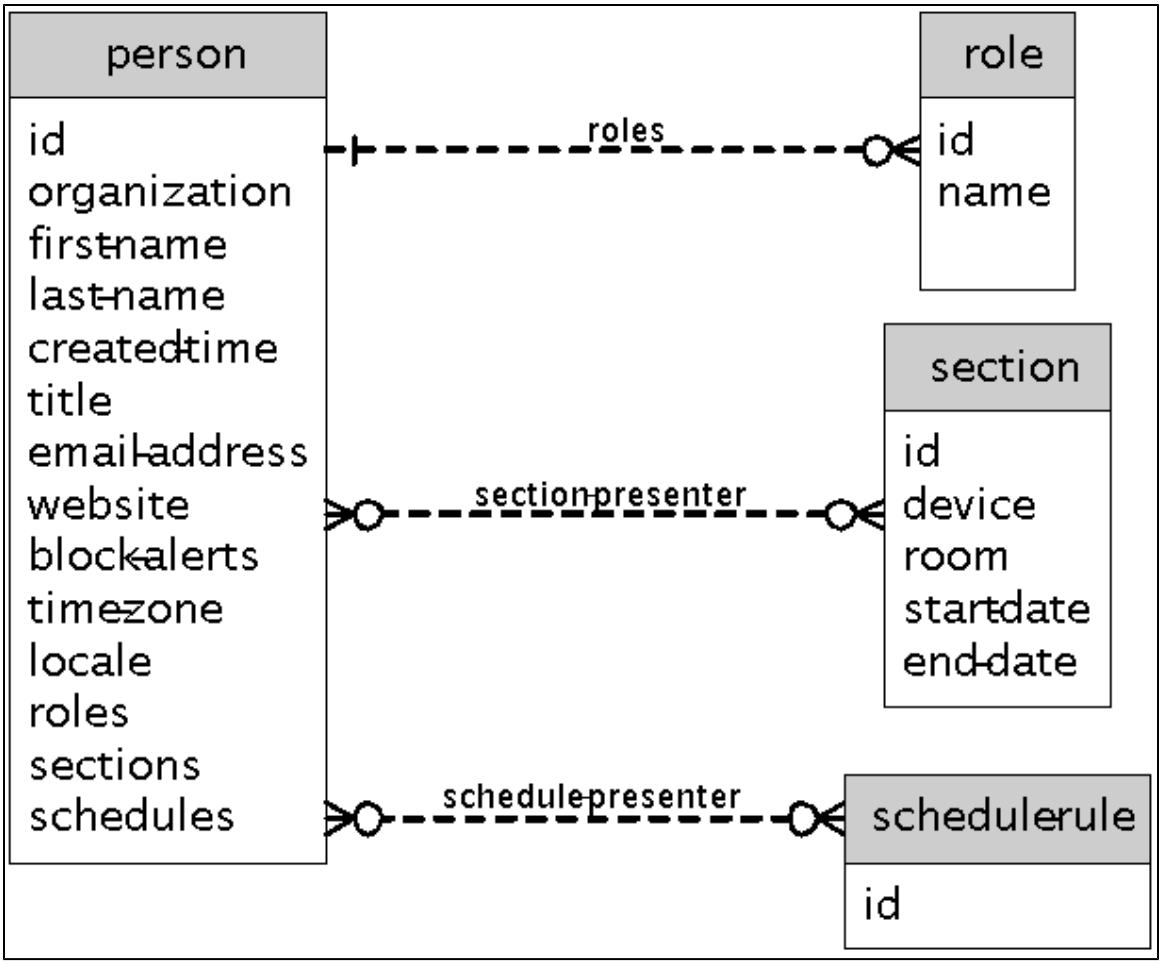
As per the OAuth specification, the OAuth parameters and signature are passed in the HTTP "Authorization" header field.

OAuth Terms
Below are the terms defined by the OAuth specification and how they apply to the Scheduling API.

Producer: The Echo360 System.
Consumer: The system trusted by the Echo360 System.
Consumer Key: The key supplied by the API User when registering as a Trusted System.
Consumer Secret: The shared secret generated by the Echo360 System when the consumer was registered as a Trusted System.
Request Token/Secret: The Scheduling API currently utilizes a simple two-step OAuth authorization process. The request token and request secret are not issued by the Echo360 System.
Auth Token/Secret: Since the Scheduling API requires a manual registration process for the consumer key/secret, the session based auth token is skipped and defaults to empty strings.
References
The following references may be useful in understanding OAuth:
• Official OAuth Documentation. [http://oauth.net/documentation/getting-started|http://oauth.net/documentation/getting-started].
• NetFlix Authentication Overview. http://developer.netflix.com/docs/Security#0_18325.

Person/User Related API Resources (Person, Role)

A "person" in the Echo System can have multiple "roles": as a presenter, student, administrator, scheduler, ... Presenters have the additional association to Sections and Schedule Event Rules. These resources and their relationships and attributes are shown in the entity diagram below.



Person Entity (person) / Collection (people)

A person in the Echo System can have one or more roles. Predefined roles are presenter, scheduler, administrator, av tech, and server admin. Presenters can be assigned to Section and/or Schedule Event Rule entities.

Person Entity's XML mapping

This table defines the mapping of person attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Person. Auto set to the caller's default organization.
last-name	All	string	Required Person's last name
first-name	All	string	Required, Person's first name
created-time	Detailed	iso-time	Auto populated with the creation time stamp.
title	Detailed	string	Optional formal title of the Person

email-address	Detailed	string	Required email address for the Person
website	Detailed	string	Optional URL for the person's reference website.
time-zone-name	Detailed	string	Optional time zone of the Person.
locale	Detailed	string	Optional Locale of the Person.
block-alert	Detailed	boolean	Relates to system alerts and requires e-mail address when true. Default is false.
credentials / user-name	Detailed	string	Required on create, the login user-name string.
credentials / password	Detailed	string	Required on create, the login password string. Due to security concerns, the password is never included in the Person response message.
roles	Detailed	roles collection	List of roles assigned to the person. Inline collection of "roles" containing "role" names. Inline since role entity is a simple structure.

"roles" inline collection XML mapping

The Person's assigned roles is a simple collection element called "roles"; which contains "role" names. each "role" element has a text role name.

Field / Element	Rep. Levels	Type (length)	Information
role	All	enum string	Short or long role name. For example: "admin" or "role-admin" are the short/long role names for the System Administrator Role. Short Role Names: "admin", "server-admin", "av-tech", "presenter", *"scheduler" • (new in 2.6).

People Collection Resource URL

The [?term="name-pattern"] URL parameter filters people by matching the person's name to the name pattern.

Method	URL: {base-uri}/people[?term="{name-pattern}"]
GET	Get all people within the caller's organization. The optional "term" parameter limits the list to matching on the name (last, first) attribute. **Reply Content:** "people" collection XML containing "person" summary elements.
POST/ PUT	Add a new Person. **Request Content:** "person" detail XML **Response Content:** "person" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Person Entity Resource URL

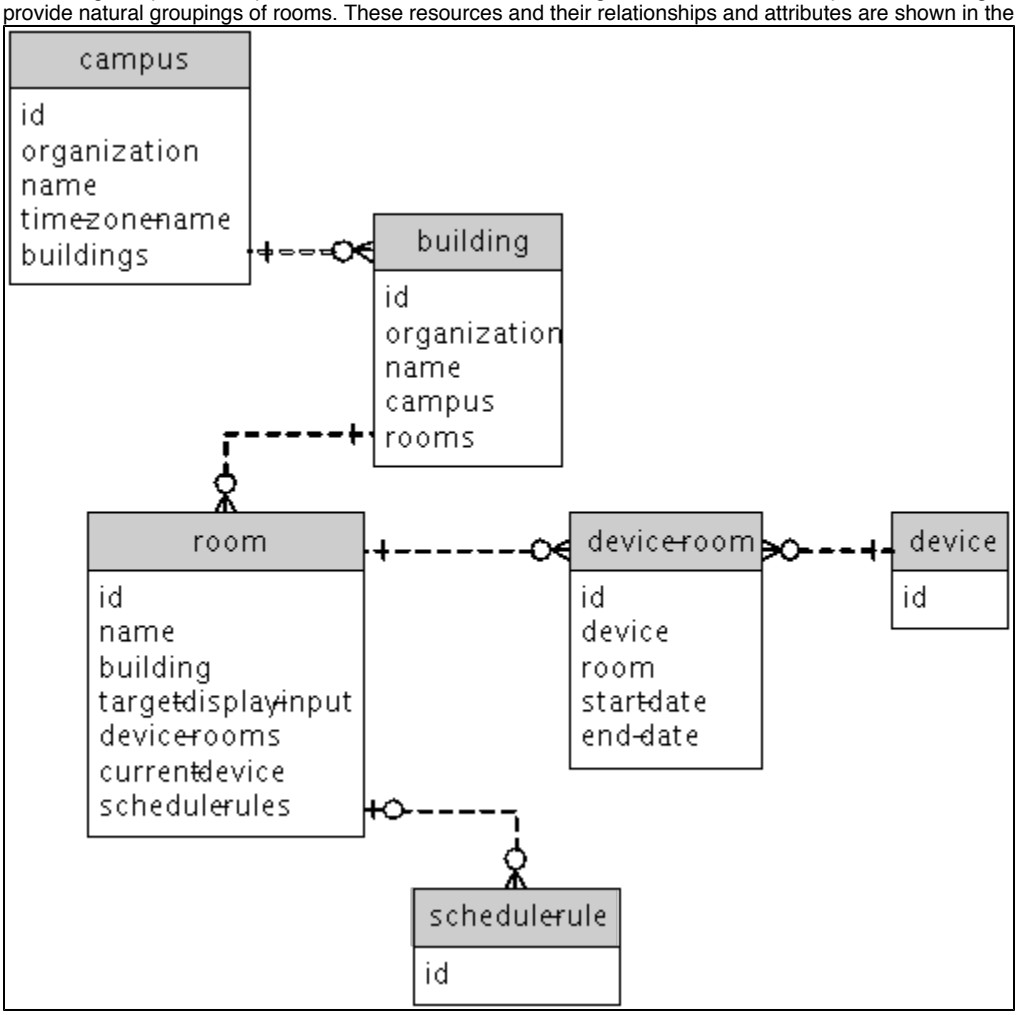
Below are the Restful methods for managing a specified Person Entity.

Method	URL: {base-uri}/people/{person-id}
GET	Retrieve the detailed information on the Person entity. **Reply Content:** "person" detailed XML. **Errors:** not found

PUT/ POST	Update the specified Person entity. **Request Content:** "person" detail XML. Only need to specify the fields that are being updated. Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Person. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Room Related API Resources (Campus, Building, and Room)

The information on the university's/organization's meeting or classrooms is maintained by the Echo System. The Room entity/resource is a key scheduling component. Capture Devices and Sections are assigned to rooms. The Campus and Building are secondary entities/resources that provide natural groupings of rooms. These resources and their relationships and attributes are shown in the entity diagram below.



Campus Entity(campus) / Collection(campuses)

A Campus belongs to a parent Organization and has a collection of Buildings. The Campus entity primarily provides a means of grouping rooms. The campus time zone attribute is inherited by all buildings and associated rooms within the campus.

Campus Entity XML mapping

This table defines the mapping of Campus attributes/fields to/from its various XML representations (summary or detailed). Note: optional fields can be omitted from the XML representations if the value is null/empty. Omitting empty fields allows concise data representation.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this campus. Auto set to the caller's default organization.
name	All	string	Required, unique campus name.
time-zone-name	Detailed	string	Optional, defaults to caller's default time zone. http://en.wikipedia.org/wiki/List_of_zoneinfo http://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones http://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones_time_zones Examples: "America/New_York", "Pacific/Honolulu"
buildings	Detailed	link	Link to the building collection. Initially empty at create.

Campus Collection Resource URL

Scheduling API URLs for accessing/managing the system's Campus Collection.

Method	URL: {base-uri}/campuses[?term="{name-pattern}"]
GET	Retrieve the list of Campus entities. The optional "term" parameter limits the list to campus objects who's name is like the search term value. **Reply Content:** "campuses" collection XML containing "campus" summary elements.
POST/ PUT	Add a new campus to the campus collection resource. **Request Content:** "campus" detail XML (name, time-zone-name elements) **Response Content:** "campus" summary XML ("id" element holds new ID) **Errors:** duplicate name, bad time-zone, client error (content invalid)

Campus Entity Resource URL

Scheduling API URLs for accessing/managing an existing Campus entity.

Method	URL: {base-uri}/campuses/{campus-id}
GET	Retrieve the detailed information on the Campus entity. **Reply Content:** "campus" detailed XML. **Errors:** not found
PUT/ POST	Update the specified campus entity. **Request Content:** "campus" detail XML. Only need to specify the fields that are being updated (name, time-zone-name). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, bad time-zone, internal error
DELETE	Remove the campus. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Building Entity(building) / Collection(buildings)

A Building belongs to a parent Campus and has a collection of Rooms. The time zone attribute is inherited by the Building from the parent Campus.

Building Entity XML mapping

This table defines the mapping of building attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
campus	All	link	Link to the parent Campus.
name	All	string	Required, unique name within the Campus.
time-zone-name	Detailed	string	This field is read-only and inherited from the parent Campus. It is included on "read" operations as a convenience.
rooms	Detailed	link	Link to the room collection. Initially empty at create.

Building Collection Resource URL

These tables list the Scheduling API URLs that operate on the Building Collections (one per Campus).

Method	URL: {base-uri}/campuses/{campus-id}/buildings[?term="{name-pattern}"]
GET	Retrieve the list of Buildings that are assigned to the Campus. The optional "term" parameter limits the list to buildings who's name is like the search term value. **Reply Content:** "buildings" collection XML containing "building" summary elements.
POST/ PUT	Add a new building to the specified campus resource. **Request Content:** "building" detail XML (name element) **Response Content:** "building" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/buildings[?term="{name-pattern}"]
GET	Retrieve the list of all Buildings across all Campuses. The optional "term" parameter limits the list to buildings who's name is like the search term value. **Reply Content:** "buildings" collection XML containing "building" summary elements.

Building Entity Resource URL

These REST URLs/Methods take action directly on an existing Building Entity.

Method	URL: {base-uri}/buildings/{building-id}
GET	Retrieve the detailed information on the Building entity. **Reply Content:** "building" detailed XML. **Errors:** not found
PUT/ POST	Update the specified Building entity. **Request Content:** "building" detail XML. Only need to specify the fields that are being updated (name). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)

DELETE	Remove the building. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Room Entity(room) / Collection(rooms)

A room is a central component of a lecture capture schedule. A room belongs to a parent Building. Rooms are reserved/scheduled for Sections and their associated capture events.

Room Entity's XML mapping

This table defines the mapping of room attributes/fields to its various XML representations (summary or detailed). A room contains many link fields due to it being reserved by Sections and Captures.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
building	All	link	Link to the parent Building.
name	All	string(32)	Required, unique name within the parent Building.
time-zone-name	Detailed	string	This field is read-only and inherited from the grandparent Campus. It is included on "read" operations as a convenience.
target-display-input	Detailed	string	Optional: "800x600", "1024x768", "1280x720", "1280x768", "1280x800", "1280x1024", "1366x800", "1440x900", "1600x1200", "1680x1050"
device-rooms	Detailed	link	Link to fetch devices that are/were(history) used by a room. The typical case is that 1 device is dedicated to a given room. The device-room abstraction allows moving a device to another room, or device swapping for maintenance/upgrades. (see Device Resource)
current-device	Detailed	composite element	For convenience, the current device that is assigned to the room (read-only). If no device is assigned, then the XML element is empty. Otherwise, it contains the "id" and "key" fields of the assigned device.
schedule-rules	Detailed	link	Link to fetch the schedule-rules for this room. schedule-rules contain Capture reservations for Sections, as well as lecture capture events. (see Schedule Rules Resource) Link = "GET {base-uri}/rooms/{room-id}/schedule-rules"

"current-device" XML mapping

The Room's current device composite XML element is described by the below table.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated identifier of the Device.
key	All	string	unique device key used when addressing the device

Room Collection Resource URL

There are various ways to "get" a collection of rooms. For instance, rooms can be filtered by campus, building, or section. The [?term="name-pattern"] URL parameter filters rooms by matching the room name to the name search pattern.

Method	URL: {base-uri}/buildings/{building-id}/rooms[?term="{name-pattern}"]
GET	Retrieve the list of Rooms that are assigned to the Building. The optional "term" parameter limits the list to rooms who's name is like the search term value. **Reply Content:** "rooms" collection XML containing "room" summary elements.
POST/ PUT	Add a new Room to the specified Building resource. **Request Content:** "room" detail XML (name, target-display-input elements) **Response Content:** "room" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/campuses/{campus-id}/rooms[?term="{name-pattern}"]
GET	Get all rooms within the specified Campus. Optional "term" parameter limits list to rooms with the matching name search pattern. **Reply Content:** "rooms" collection XML containing "room" summary elements.

Method	URL: {base-uri}/rooms[?term="{name-pattern}"]
GET	Get all rooms within the caller's organization. Optional "term" parameter limits list to rooms with the matching name pattern. **Reply Content:** "rooms" collection XML containing "room" summary elements.

Room Entity Resource URL

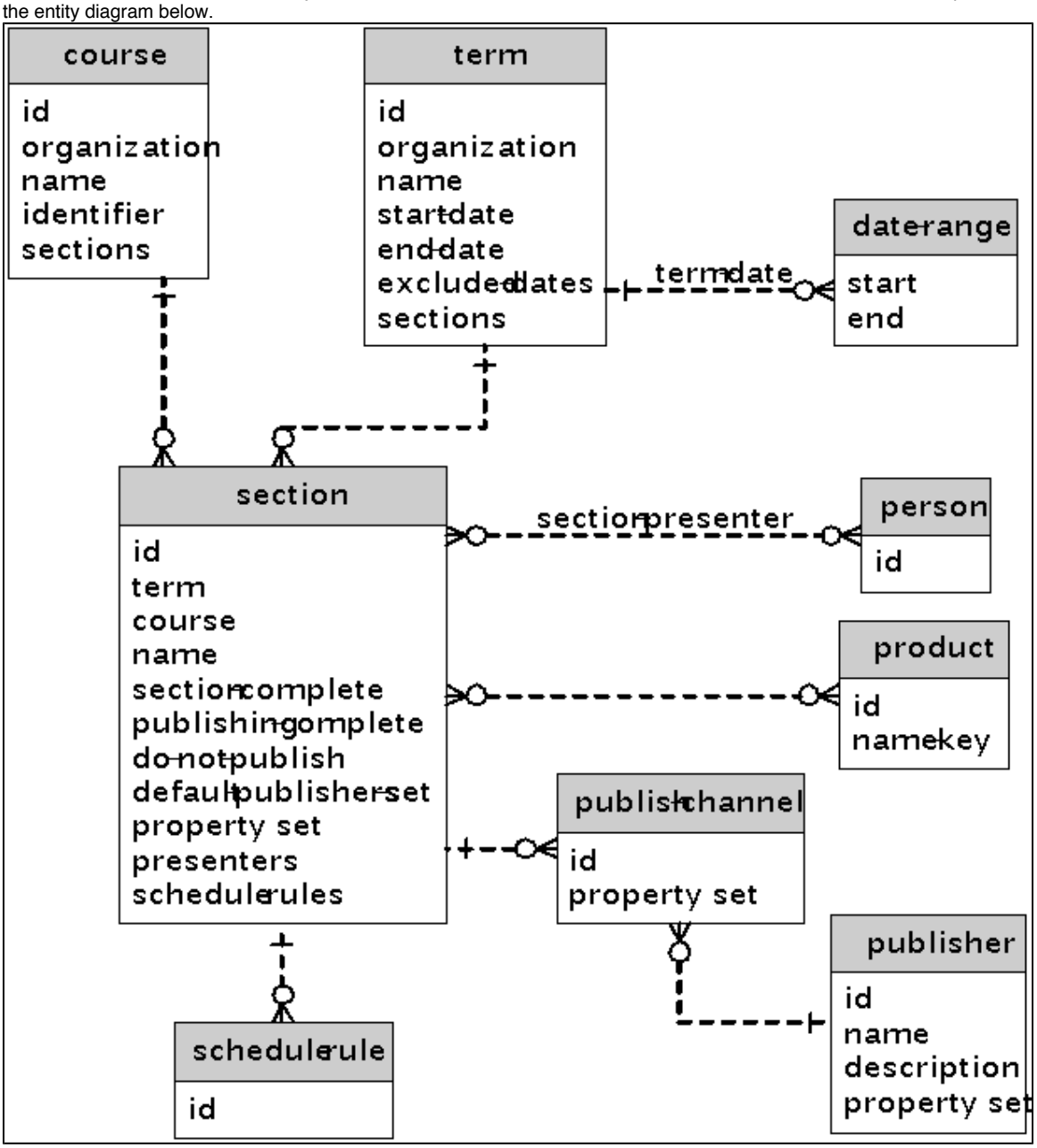
Note: Assigning a Device to a Room is performed via the Device Resource. Similarly, the Schedule Rule Resource provides methods to manage the room assigned to the Section's schedule.

Method	URL: {base-uri}/rooms/{room-id}
GET	Retrieve the detailed information on the Room entity. **Reply Content:** "room" detailed XML. **Errors:** not found
PUT/ POST	Update the specified Room entity. **Request Content:** "room" detail XML. Only need to specify the fields that are being updated (name, target-display-input). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Room. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Section Related API Resources (Product, Publisher, Term, Course, Section)

The "Section" is a key scheduling entity, since lecture captures are scheduled against these sections. The Term and Course provide two separate ways to group the sections. Courses can exist across Terms. The Section's meetings/sessions are "what" is captured by the Echo System. The

"when and where" are determined by the Section's Scheduled Event Rules. These resources and their relationships and attributes are shown in the entity diagram below.



Capture Product Entity (product) / Collection (products) [since 2.6]

Each Lecture Capture performed by the Echo System produces 1 or more Capture Products. Which media inputs and the format/quality of the recorded media is determined by the Capture Product selection. The system comes with a predefined set of profiles. The Scheduling API only allows read access to these Product Profiles.
The desired Capture Products are assigned to Section and/or Schedule Event Rule entities. The system internal "id" attribute of the target Capture Product is passed to the appropriate Section or Schedule Rule management operations.

Note: Another way to assign Capture Products is at the system configuration level. The Echo System can be configured with a default set of Capture Products that are automatically assigned to all new Sections.

Product Entity's XML mapping

The Capture Product entity is simply a list of product names. This table defines the mapping of Product Profile attributes/fields for its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information

id	All	GUID	System generated product identifier.
name-key	All	string	Unique profile name.

REST Method/URL mapping

The API supports listing the Capture Product entries. The following tables describe the Restful Methods supported by the Product Profile Resource.

Method	URL: {base-uri}/products
GET	Get a list of all Product Profiles defined in the Echo System. **Reply Content:** "products" collection XML containing "product" summary elements.

Method	URL: {base-uri}/products/{product-id}
GET	Retrieve the detailed information on the specified Product. **Reply Content:** "product" detailed XML.. **Errors:** Product not found.

Publisher Entity (publisher) / Collection (publishers) [since 2.6]

The Echo System supports distributing presentations to various media outlets via the Publisher Plug-In Framework. Each registered publisher can have a set of publishing "properties" that are specific to the given publisher. Example properties are the credentials to connect to the target media outlet, the media group identifier, ... Some of the properties can be set at the publisher plug-in level and are not presentation/content specific. Other properties will need to be specified on the presentation via Section publish properties. The Scheduling API provides read-only access to the set of registered Publisher plug-ins.

Publisher Entity's XML mapping

The Publisher entity is simply a list of Publisher key names and the associated property settings. These tables define the mapping of Publisher attributes/fields for its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated publisher identifier.
name	All	string	Unique Publisher name.
description	All	string	Optional text description.
properties	Detailed	collection	Inline collection of publisher "property" items.

Publisher Property Item's XML mapping

The Publisher's "properties" collection is a simple list of "property" items. Below, is the "property" XML mapping table .

Field / Element	Rep. Levels	Type (length)	Information
key	All	string	Unique Publisher property key.
value	All	string	Value bound/assigned to the property.

Publisher REST Method/URL mapping

The API supports listing the Publisher entries. The following tables describe the Restful Methods supported by the Publisher Resource.

Method	URL: {base-uri}/publishers
GET	Get a list of all Publishers registered in the Echo System. **Reply Content:** "publishers" collection XML containing "publisher" summary elements.

Method	URL: {base-uri}/publishers/{publisher-id}
GET	Retrieve the detailed information on the specified Publisher. **Reply Content:** "publisher" detailed XML.. **Errors:** Specified publisher was not found.

Publish Channel Entity (publish-channel) / Collection (publish-channels) [since 2.6]

When a Publisher assignment is added to a Section, a Publish Channel is created. The Publish Channel holds the publisher property settings that are specific to the Section usage. The actual property names are defined by the Publisher type/plug-in. A common property found on most Publisher types is the CMS-ID; which defines the target Course ID to group the resulting presentations.

Note: A Section can have multiple channels with the same Publisher. This assumes the Channel property set will contains different "values".

Publish Channel Entity's XML mapping

The Publish Channel is simply a set of "properties" and links to the associated Section and Publisher. The Property Set is a collection of name/value pairs.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated channel identifier.
section	All	link	URL Link to get the associated Section detail.
publisher	All	link	URL Link to get the associated Publisher detail.
properties	Detailed	collection	Inline collection ("properties") of publisher "property" items.

Publish Channel REST Method/URL mapping

The API supports listing the Publish Channel entries for a Section, and the management of the associated collection. The following tables describe the Restful Methods supported by the Publish Channel Resource.

Method	URL: {base-uri}/sections/{section-id}/publish-channels
GET	Get a summary list of all Publish Channels assigned to the Section. **Reply Content:** "publish-channels" collection XML containing "publish-channel" summary elements. **Errors:** Section not found.
DELETE	Remove(clear) all associated Publish Channels from the Section. **Errors:** Section not found.

Method	URL: {base-uri}/publish-channels/{publish-channel-id}

GET	Retrieve the detailed information on the specified Publish Channel. **Reply Content:** "publish-channel" detailed XML.. **Errors:** Publish Channel was not found.
PUT/ POST	Update(modify) the existing Publish Channel. **Request Content:** "publish-channel" detail XML. If present, the uploaded property set will replace the property set currently stored in the Channel. **Response Content:** Status. **Errors:** Channel not found, client error(bad data format)
DELETE	Remove the Publish Channel. **Errors:** Channel not found.

Method	URL: {base-uri}/sections/{section-id}/publishers/{publisher-id}/publish-channels
GET	Get a summary list of all Channels assigned to the Section for the given Publisher. **Reply Content:** "publish-channels" collection XML containing "publish-channel" summary elements.
PUT/ POST	Add a new Publish Channel to the Section targeting the Publisher. **Request Content:** "publish-channel" detail XML (contains the new property set). **Response Content:** "publish-channel" detail record. The "id" field holds the new channel id. **Errors:** Section not found, client error(bad data format)
DELETE	Remove(clear) associated Publish Channels from the Section matching the Publisher. **Errors:** Section not found.

Method	URL: {base-uri}/publishers/{publisher-id}/publish-channels
GET	Get a summary list of all Channels that target the Publisher. This may be a large result. **Reply Content:** "publish-channels" collection XML containing "publish-channel" summary elements.

Academic Term Entity(term) / Collection(terms)

An academic "term" is a period in time in which a collection of courses is offered to students. Typical terms are: Fall, Spring, Summer I, etc.

Term Entity's XML mapping

This table defines the mapping of term attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Term. Auto set to the caller's default organization.
name	All	string(32)	Required, unique name.
start-date	All	iso-date	The date of the first day of the Term. Format: "YYYY-MM-DD"
end-date	All	iso-date	The date of the last day of the Term. Format: "YYYY-MM-DD"

excluded-dates	Detailed	composite collection	An inline collection of date ranges which indicate dates when the Term is not in session. These could be scheduled holiday and break dates. The collection child element is "date-range".
sections	Detailed	link	Link to fetch the collection of Sections offered within the academic term.

"date-range" collection item

The Term contains an inline "excluded-dates" collection; which is a list of "date-range" XML elements. A "date-range" element is a simple container holding a "start", and "end" ISO-date element. The format of the date-range element is in the below table.

Field / Element	Rep. Levels	Type (length)	Information
start	All	iso-date	The date of the first day of the date range. Format: "YYYY-MM-DD"
end	All	iso-date	The date of the last day of the date range. Format: "YYYY-MM-DD"

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters Terms by matching the Term's name to the search term pattern. Don't confuse the search "term" URL parameter with the "Term" entity.

Method	URL: {base-uri}/terms[?term="{name-pattern}"]
GET	Get all Terms within the caller's organization. **Reply Content:** "terms" collection XML containing "term" summary elements.
POST/ PUT	Add a new Term to the caller's organization. **Request Content:** "term" detail XML (name, start/end date, exclude-dates elements) **Response Content:** "term" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Term Entity Resource URL

Method	URL: {base-uri}/terms/{term-id}
GET	Retrieve the detailed information on the Term entity. **Reply Content:** "term" detailed XML. **Errors:** not found
PUT/ POST	Update the specified Term entity. If the "excluded-date" element is present in the request content, then the current excluded dates are replaced by the request contents. **Request Content:** "term" detail XML. Only need to specify the fields that are being updated (name, start/end date, excluded-dates). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Term. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Term's Excluded Dates Collection Resource URL

On Term add or update API requests, the caller can include an excluded-dates collection within the uploaded Term XML. If present, the excluded

dates collection defines (on add) or replaces (on update) the current Term excluded dates. This inline collection support is a convenience provided by the API.

The alternate approach is to manage the Term's excluded dates separate from the Term. The table below defines the resource URL calls for managing the exclude dates collection separate from the Term entity.

Method	URL: {base-uri}/terms/{term-id}/excluded-dates
GET	Retrieve the excluded-dates collection for the specified Term. **Reply Content:** "excluded-dates" collection that contains "date-range" elements. **Errors:** not found
POST/ PUT	Add a date range to the Term's excluded dates collection. **Request Content:** "date-range" XML. **Response Content:** "date-range" XML (can be ignored) **Errors:** not found, client error(bad data format)
DELETE	Remove(clear) all entries from the specified Term's excluded-date collection. **Errors:** not found

Course Entity(course) / Collection(courses)

A Course is offered as one or more Sections within various academic Terms.

Not all schools divide Courses into Sections. In this case, the Course will have only one section. To facilitate this use case, the EchoSystem can be configured to create a Section record in the "current" academic Term for any newly created Course. If this is the case at your installation, then adding a Course via the API will also create the default Section.

Course Entity's XML mapping

This table defines the mapping of Course attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Course. Auto set to the caller's default organization.
name	All	string	Required, unique course name.
identifier	All	string	Course identifier known externally by the organization.
sections	Detail	link	Link to fetch the collection of Sections offered for this Course.

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters Courses by matching the course name to the search pattern.

Method	URL: {base-uri}/courses[?term="{name-pattern}"]
GET	Get all Courses within the caller's organization. **Reply Content:** "courses" collection XML containing "course" summary elements.

POST/ PUT	Add a new Course to the caller's organization. **Request Content:** "course" detail XML (name, identifier) **Response Content:** "course" summary XML ("id" element holds new ID) **Errors:** duplicate "name", client error (content invalid)

Method	URL: {base-uri}/terms/{term-id}/courses[?term="{name-pattern}"]
GET	Get all Courses with at least one Section within the specified Academic Term. **Reply Content:** "courses" collection XML containing "course" summary elements. **Errors:** Term not found

Course Entity Resource URL

Standard API URLs for operating on an existing Course Entity.

Method	URL: {base-uri}/courses/{course-id}
GET	Retrieve the detailed information on the Course entity. **Reply Content:** "course" detailed XML. **Errors:** Course not found
PUT/ POST	Update the specified Course entity. **Request Content:** "course" detail XML. Only need to specify the fields that are being updated (name, identifier). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** not found, duplicate name, client error(bad data format)
DELETE	Remove the Course. Marked as deleted and reaped at a later date. **Errors:** not found, client error (undeletable).

Section Entity (section) / Collection (sections)

A Section is an offering of a Course for a particular Academic Term. Sections are scheduled against rooms and have lecture captures. The Section's schedule is a set of associated Schedule Event Rules.

Section Entity's XML mapping

This table defines the mapping of section attributes/fields to/from its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
term	All	link	Link to the academic term for this Section.
course	All	link	Link to the Section's parent course.
name	All	string(32)	Required, unique section name.
section-complete	Detailed	boolean	Indicates if the Section is now complete. If complete, no more captures are scheduled for the section.
publishing-complete	Detailed	boolean	Have all the section captures been published.

do-not-publish	Detailed	boolean	Don't allow captures from the section to be published.
default-publishers-set	Detailed	boolean	Were the default publishers setup for the section.
properties	Detailed	inline collection	Inline collection of Section "property" items. [New in v2.6]
presenters	Detailed	inline collection	Inline list of presenters associated with the Section. This is a "presenters" collection containing "presenter" items.
schedule-rules	Detailed	link	Link to fetch the schedule-rules for this Section. Schedule-rules contain Capture reservations for Sections, as well as lecture capture events. (see Schedule Rules Resource)

Section Property Item's XML mapping [New in v2.6]

A Section has a set of Media Property assignments. These settings are implemented as a Section property set. A property set is a collection name/value pairs. The Section record contains the following property key names: " presentation-initially-unavailable", "delete-original-media", "delete-original-media-days", "copyright-owner", and "security-module". Below, is the "property" XML mapping table .

Field / Element	Rep. Levels	Type (length)	Information
key	All	string	Unique Section property key.
value	All	string	Value bound/assigned to the property.

<properties>

<property>
<key>presentation-initially-unavailable</key>
<value>false</value>
</property>
<property>
<key>delete-original-media</key>
<value>true</value>
</property>
<property>
<key>delete-original-media-days</key>
<value>112</value>
</property>
<property>
<key>copyright-owner</key>
<value>John Smith</value>
</property>
<property>
<key>security-module</key>
<value>Allow All</value>
</property>
</properties>Sample Section Property Set XML

"presenter / person" inline collection item

The Section entity contains an inline "presenters" collection; which is a list of "presenter" XML elements. A "presenter" element is a simple container holding a "person" reference element. The format of the nested "person" xml element is in the below table.

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	ID of the person which is a used as a "presenter.
first-name	All	string	The person's first name.

last-name	All	string	The person's last name.

Section Collection Resource URL

There are various ways to "get" a collection of Sections. The [?term="name-pattern"] URL parameter filters Sections by matching the Section's name to the search name pattern.

Method	URL: {base-uri}/terms/{term-id}/courses/{course-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Course and Academic Term. The optional search "term" parameter limits the list to Sections matching on the name attribute. **Reply Content:** "sections" collection XML containing "section" summary elements. **Errors:** course/term not found
POST/ PUT	Add a new Section to the specified Course and Academic Term. **Request Content:** "section" detail XML (name, retention parameters, presenters) **Response Content:** "section" summary XML ("id" element holds new ID) **Errors:** duplicate name, course/term not found, client error (content invalid)

Method	URL: {base-uri}/courses/{course-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Course. The optional "term" parameter limits the list to Sections matching on the search name attribute. **Reply Content:** "sections" collection XML containing "section" summary elements. **Errors:** course not found

Method	URL: {base-uri}/terms/{term-id}/sections[?term="{name-pattern}"]
GET	Retrieve the list of Sections for the given Academic Term. The optional search "term" parameter limits the list to Sections matching on the name attribute. **Reply Content:** "sections" collection XML containing "section" summary elements. **Errors:** Term not found

Section Entity Resource URL

Below are the standard Restful methods for managing an existing Section Entity.

Method	URL: {base-uri}/sections/{section-id}
GET	Retrieve the detailed information on the Section entity. **Reply Content:** "section" detailed XML. **Errors:** Section not found
PUT/ POST	Update the specified Section entity. **Request Content:** "section" detail XML. Only need to specify the fields that are being updated (retention attributes, presenters). Non-updatable fields are ignored. **Response Content:** Status Only **Errors:** Section not found, duplicate name, client error(bad data format)
DELETE	Remove the Section. Marked as deleted and reaped at a later date. **Errors:** Section not found, client error (undeletable).

Section Presenters Collection URL

The section presenters can be specified within the Section entity XML, or via the following presenter URLs. These URLs allow direct manipulation of a Section's presenter list.

Method	URL: {base-uri}/sections/{section-id}/presenters
GET	Retrieve the list of section presenters for the specified Section. **Reply Content:** "presenters" collection of "presenter" elements. **Errors:** Section not found
DELETE	Clear the Section's presenter list. Removes all presenters currently assigned to the Section. **Errors:** Section not found

Method	URL: {base-uri}/sections/{section-id}/presenters/{person-id}
GET	Can be used to confirm that a person is a presenter of the Section. **Reply Content:** "presenter" element. **Errors:** Section/Person was not found, or the person was not a presenter.
POST/ PUT	Add the person to the Section's presenter list (if they are not already assigned). **Request Content:** Empty **Response Content:** Empty **Errors:** Section/Person was not found
DELETE	Remove the Person from the Section's presenter list. **Errors:** Section/Person was not found. Person was not a presenter.

Section Products Collection URL [New in v2.6]

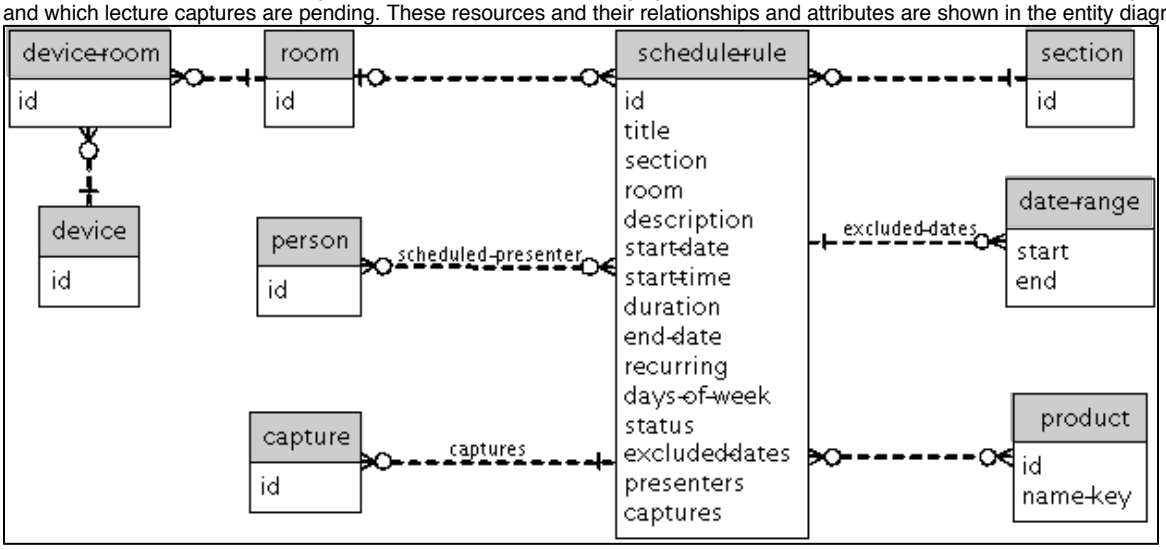
A Section can optionally have one or more Capture Product assignments. These Product assignments determine the different types of presentations generated when a Section's lecture is captured. These assignments are managed via the following collection direct manipulation URLs.

Method	URL: {base-uri}/sections/{section-id}/products
GET	Retrieve the list of Product assignments for the specified Section. **Reply Content:** "products" collection of "product" elements. **Errors:** Section not found
DELETE	Clear the Section's product list. Removes all products currently assigned to the Section. **Errors:** Section not found

Method	URL: {base-uri}/sections/{section-id}/products/{product-id}
GET	Can be used to confirm that a given product is assigned to the Section. **Reply Content:** "product" detailed element. **Errors:** Section/Product was not found, the product was not assigned.
POST/ PUT	Add the Product to the Section (if it is not already assigned). **Request Content:** Empty **Response Content:** Empty **Errors:** Section/Product not found
DELETE	Remove the Product from the Section. **Errors:** Section/Product was not found, or Product was not assigned.

Schedule Related API Resource (schedule rule)

The central schedule related entity is the Schedule Rule. This entity specifies when a section meets, in which room, the presenters/instructors, and which lecture captures are pending. These resources and their relationships and attributes are shown in the entity diagram below.



Schedule Rule Entity (schedule-rule) / Collection (schedule-rules)

Schedule Rule Entity's XML mapping

This table defines the mapping of Schedule Rule attributes/fields for its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
section	All	link	link to the parent Section offering.
room	All	link	Link to the room assignment for capturing the section. Note: the rule can have an unassigned room; which results in an absent room link.
title	All	string	Required name/title for the rule. Useful for searching.
status	All	string	Enumeration constant: (draft, active, complete, trashed)
description	Detail	string	Optional text description of the rule.
start-date	Detail	iso-date	The starting period for this rule. Can be empty if the rule is still in "draft" state. Format: "YYYY-MM-DD"
end-date	Detail	iso-date	If this is a recurring capture, then the end date of the rule is required. Format: "YYYY-MM-DD"
start-time	Detail	iso-time	The starting period for this rule. Can be empty.
duration-seconds	Detail	int	Each capture is scheduled to run this many seconds.
recurring	Detail	boolean	Is this rule a one time capture event or does it capture multiple events?

days-of-week	Detail	composite element	Group of Booleans that indicate which days of the week the recurring capture is performed. The field is empty if the recurring bit is false.
excluded-dates	Detail	date-range collection	Inline list of "date-range" elements representing dates when the capture should not take place. For example breaks or holidays. See Term Entity's "excluded-dates".
target-display-input-override	Detail	string	Optional setting for the capture input dimensions generated by this rule. For example: "640x720".
presenters	Detail	presenter collection	Inline list of presenters for the capture events. See Section Entity's "presenters".
captures	Detail	link	Link to the Capture Events generated by this schedule rule.

"schedule-rule/days-of-week" XML mapping

This table defines the mapping of the days-of-week data structure to its XML representations.

Field / Element	Rep. Levels	Type (length)	Information
sunday	All	boolean	True if the capture recurs every Sunday.
monday	All	boolean	True if the capture recurs every Monday.
tuesday	All	boolean	True if the capture recurs every Tuesday.
wednesday	All	boolean	True if the capture recurs every Wednesday.
thursday	All	boolean	True if the capture recurs every Thursday.
friday	All	boolean	True if the capture recurs every Friday.
saturday	All	boolean	True if the capture recurs every Saturday.

Schedule Rule Entity REST Method/URL mapping

The following tables describe the Restful Methods supported by the Schedule Rule Resource.

Method	URL: {base-uri}/sections/{section-id}/schedule-rules[?term="name-pattern"]
GET	Get all Rules associated with the specified Section offering. **Reply content:** "schedule-rules" collection XML containing "schedule-rule" summary elements. **Errors:** Section not found.
POST/ PUT	Add a new Schedule Rule to the Section. **Request Content:** "schedule-rule" detail XML **Response Content:** "schedule-rule" summary XML ("id" element holds new ID) **Errors:** duplicate name, client error (content invalid)

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}

GET	Retrieve the detailed information on the Schedule Rule. **Reply content:** "schedule-rule" detailed XML.. **Errors:** not found.
PUT/ POST	Update the specified Schedule Rule. **Request Content:** "schedule-rule" detail XML (description, start-date, start-time, end-date, duration-seconds, target-display-override, recurring, days-of-week, exclude-dates, status, presenters). Note: Only the fields that have changed are required to be present. **Errors:** duplicate name, client error (content invalid)
DELETE	Remove the schedule rule. May not be allowed given the schedule rule status. **Errors:** not found

Schedule Rule "room" assignment Resource URL

On Schedule Rule add or update API requests, the caller can include the assigned room within the uploaded entity XML. The alternate approach is to manage the room assignment operation separate from the Schedule Rule entity update operation. The tables below define the resource URL calls for directly managing the room assignment.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/rooms
GET	Retrieve a "rooms" collection for the specified Schedule Rule. The collection will contain zero or one room summary entity. **Reply Content:** "rooms" collection that contains zero/one "room" element. **Errors:** Schedule Rule was not found.
POST/ PUT	Not supported
DELETE	Set the room assignment to empty/null. **Errors:** Schedule Rule was not found.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/rooms/{room-id}
GET	Check if the given Room is already assigned to the specified Schedule Rule. **Returns 200** if the room is already assigned to the rule. **Returns 400(not found)** if the room is not assigned to the rule. **Returns 400(not found)** if the Schedule Rule or Room are not found.
PUT/ POST	Assign the given room to the specified schedule-rule. **Errors:** Schedule Rule or Room was not found.
DELETE	Remove the room from the Schedule Rule. **Errors:** Schedule Rule or Room was not found, or the Room was not assigned to the rule.

Schedule Rule "excluded-dates" Collection Resource URL

On Schedule Rule add or update API requests, the caller can include an excluded-dates collection within the uploaded entity XML. If present, the excluded dates collection defines (on add) or replaces (on update) the current Schedule Rule's excluded dates. This inline collection support is a convenience provided by the API.

The alternate approach is to manage the excluded dates separate from the Schedule Rule entity. The table below defines the resource URL calls for managing the exclude dates collection.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/excluded-dates

GET	Retrieve the excluded-dates collection for the specified Schedule Rule. **Reply content:** "excluded-dates" collection that contains "date-range" elements. **Errors:** Schedule Rule was not found.
POST/ PUT	Add a date range to the Schedule Rule's excluded dates collection. **Request Content:** "date-range" XML. **Response Content:** "date-range" XML (can be ignored) **Errors:** not found, client error(bad data format)
DELETE	Remove all entries from the specified Schedule Rule's excluded-date collection. **Errors:** not found

Schedule Rule "presenters" Collection URL

The Schedule Rule's presenters can be specified within the "schedule-rule" entity XML, or via the following presenter URLs.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/presenters
GET	Retrieve the list of presenters for the specified Schedule Rule. **Reply content:** "presenters" collection of "presenter" elements. **Errors:** Schedule Rule was not found.
DELETE	Clear the Schedule Rule's presenter list. **Errors:** Schedule Rule was not found.

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/presenters/{person-id}
POST/ PUT	Add the person to the Schedule Rule's presenter list. **Request Content:** Empty **Response Content:** Empty **Errors:** Schedule Rule was not found.
DELETE	Remove the Person from the Schedule Rule's presenter list. **Errors:** Schedule Rule was not found.

Schedule Rule Products Collection URL [New in v2.6]

A Schedule Event Rule can optionally have one or more Capture Product assignments. These Product assignments determine the different types of presentations generated when the associated lecture is captured. These assignments are managed via the following collection direct manipulation URLs.

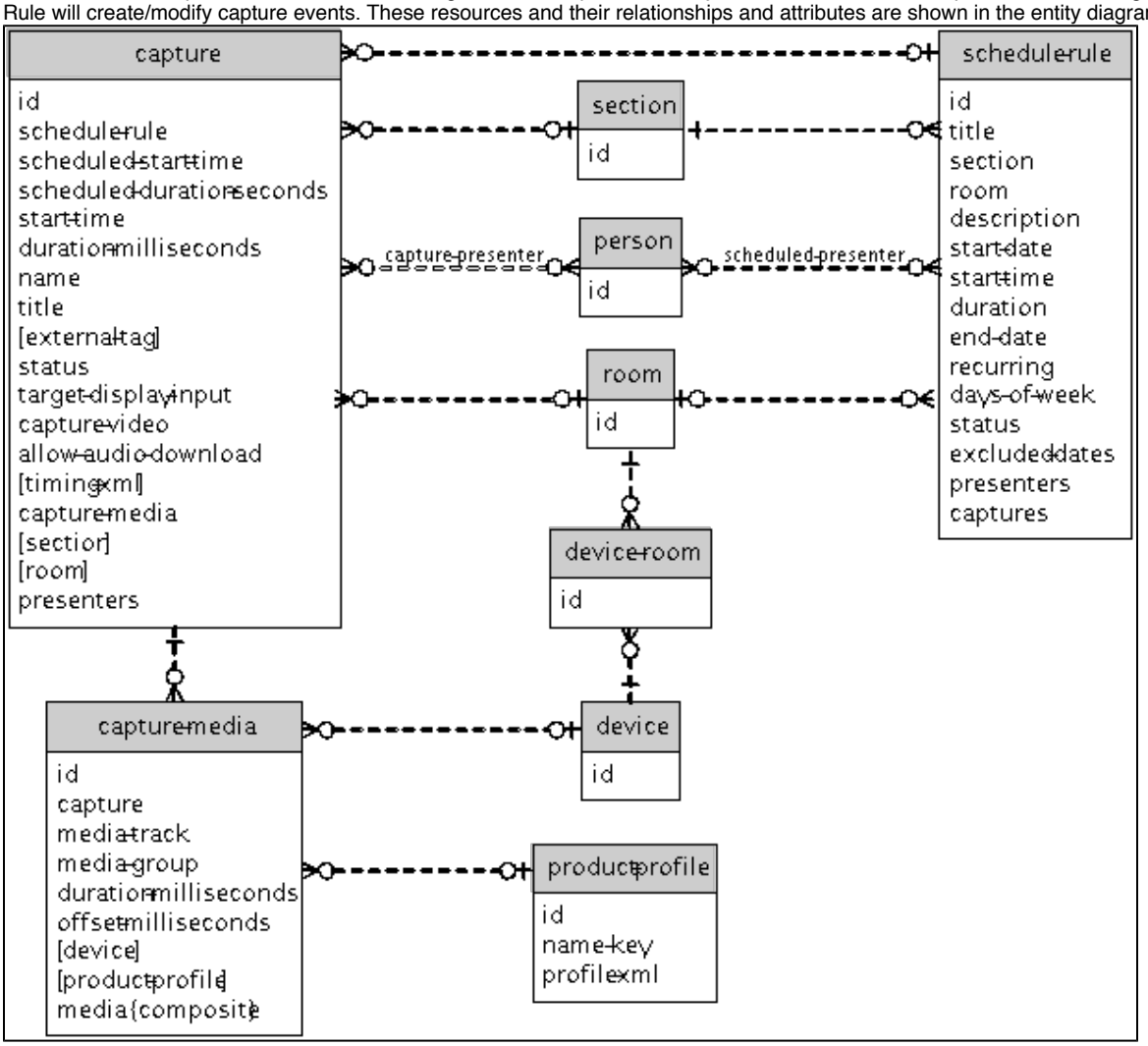
Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/products
GET	Retrieve the list of Product assignments for the specified Schedule Rule. **Reply Content:** "products" collection of "product" elements. **Errors:** Schedule Rule not found
DELETE	Clear the Schedule Rule's product list. Removes all products currently assigned to the Rule. **Errors:** Schedule Rule not found

Method	URL: {base-uri}/schedule-rules/{schedule-rule-id}/products/{product-id}

GET	Can be used to confirm that a given product is assigned to the Schedule Rule. **Reply Content:** "product" detailed element. **Errors:** Schedule Rule /Product was not found, the product was not assigned.
POST/ PUT	Add the Product to the Schedule Rule (if it is not already assigned). **Request Content:** Empty **Response Content:** Empty **Errors:** Schedule Rule /Product not found
DELETE	Remove the Product from the Schedule Rule. **Errors:** Schedule Rule /Product was not found, or Product was not assigned.

Capture Related API Resources

The Capture related entities are only available as read-only entities from the Scheduling API. They represent the capture history as well as the future scheduled capture events. The Scheduling Rule is the updatable entity that control scheduled capture events. Changes to the Schedule Rule will create/modify capture events. These resources and their relationships and attributes are shown in the entity diagram below.



Capture Entity (capture) / Collection (captures)

One or more capture entities are created when a Schedule Rule is entered into the Echo360 System. The Captures represent each instance of a lecture capture. The Scheduling API only allows read access to Capture records. Changes made to the parent Schedule Rule will update the related (future) Captures.

Capture Entity's XML mapping

This table defines the mapping of Capture Event attributes/fields for its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated entity identifier.
organization	All	link	Link to the parent organization for this Capture. Auto set to the caller's default organization.
schedule-rule	All	link	link to the parent Schedule Rule that created this capture event.
name	All	string(116)	Name used for search.
scheduled-start-time	All	iso-datetime	When the event is/was scheduled to start.
scheduled-duration-seconds	All	int	The scheduled length of the capture event.
start-time	Detail	iso-datetime	The actual start date/time of the captured event. This is set to the scheduled start time if this is a future event.
duration-milliseconds	Detail	int	The actual length of the captured event.
title	Detail	string(255)	Capture event title.
external-tag	Detail	string(50)	Optional field set to an external system's tag property.
status	Detail	string	Status enumeration value.
target-display-input	Detail	string(30)	–
capture-video	Detail	boolean	Should the event capture the video feed.
allow-audio-download	Detail	boolean	Is the audio only allowed to be downloaded.
timing-xml	Detail	CLOB	–
capture-medias	Detail	link	link to the list of media files (media data) created by this capture event.
[section]	Detail	link	In order to support event data archive, the Section captured is placed in its own capture field.
[room]	Detail	link	In order to support event data archive, the Room captured is placed in its own capture field.
[presenters]	Detail	inline-collection	In order to support event data archive, the Presenters captured is placed in its own collection field.

REST Method/URL mapping

The [?term="name-pattern"] URL parameter filters Capture by matching the Capture's name attribute to the pattern.

Method	URL: {base-uri}/schedule-rule/{schedule-rule-id}/captures[?term="{name-pattern}"]

GET	Get all Captures associated to the specified Schedule Rule. **Reply Content:** "captures" collection XML containing "capture" summary elements. **Errors:** schedule-rule was not found.

Method	**URL: {base-uri}/rooms/{room-id}/captures[?term="{name-pattern}"]**
GET	Get all Captures associated with a given room. **Reply Content:** "captures" collection XML containing "capture" summary elements. **Errors:** room was not found.

Method	**URL: {base-uri}/captures/{capture-id}**
GET	Retrieve the detailed information on the Capture event. **Reply Content:** "capture" detailed XML. **Errors:** not found

Capture Media Entity (capture-media) / Collection (capture-medias)

A single Capture Event can generate multiple media files. Each generated file is represented by the Capture Media Entity. As with the Capture event entity, the Scheduling API only allows read access to the Capture Media records.

Capture Media Entity's XML mapping

This table defines the mapping of Capture Media attributes/fields for its various XML representations (summary or detailed).

Field / Element	**Rep. Levels**	**Type (length)**	**Information**
id	All	GUID	System generated entity identifier.
capture	All	link	link to the parent Capture Event that created the media.
media-track	All	string	Enumerated constant that indicates how this media is used. values(audio, audio-vga, audio-video, flash-movie, flipbook, thumbnail, vga, video)
media-group	All	string	Enumerated constant that indicates the major media group for this media. values (edit-proxy, master, primary, projector).
duration-milliseconds	Detail	int	Running length of the media in milliseconds.
offset-milliseconds	Detail	int	–
device	Detail	link	Link to the Device which generated/captured this media.
product-profiles	Detail	inline collection	Inline list of "product-profile" items. These are the Product Profiles used when generating the media file.
media	Detail	composite element	Detailed media attributes of the captured file.

"media" composite element XML mapping

This table defines the mapping of Media attributes/fields for its XML representation. A Media entity contains information on a capture generated

media file.

Field / Element	Rep. Levels	Type (length)	Information
path	All	string	File path of the media.
type	All	string	The media file type: jpg, mp3, aac, m4b, rm, mov, wmv, flv, swf, m4v, h264, mp4
hash	All	composite element	Hash value of the associated capture media file.
file-size-bytes	All	long	Bytes size of the associated media file.
technical	All	composite element	Grouping of file capture technical stats.
tag	All	composite element	Grouping of information attributes about the media contents.

"media/hash" composite element XML mapping

This table defines the mapping of Hash attributes/fields for its XML representation. Each file generated by a capture has an associated hash value that helps verify the file integrity after file transfer.

Field / Element	Rep. Levels	Type (length)	Information
algorithm	All	string	An enumerated string that indicates the hash algorithm used to generate the hash vale. (Adler32, CRC32, MD2, MD5, SHA_1, SHA_256, SHA_384, SHA_512)
value	All	string	The actual hash value generated for the associated capture media file.

"media/technical" XML mapping

This table defines the mapping of media-technical attributes/fields to its XML representations.

Field / Element	Rep. Levels	Type (length)	Information
frame-rate	All	numeric	frame rate of the captured media.
bit-rate	All	int	Captured bit-rate of the media.
width	All	int	line/pixel width of the media frames.
height	All	int	line/pixel height of the media frames.
file-server	All	string	–

"media/tag" XML mapping

This table defines the mapping of media tag attributes/fields to its XML representations. Media Tag is a collection of metadata attributes that can be assigned to a captured media. The "tag" attributes are set/edited by a Echo360 user via the presentation UI component.

Field / Element	Rep. Levels	Type (length)	Information
year	All	int	Optional year when the content was produced/recorded

comment	All	string	Optional text describing the content
copyright	All	string	Optional copyright statement covering the content.
album	All	string	Optional album name used to group content.
artist-name	All	string	Optional person who produced the content.
genre	All	string	Optional category of the content.

REST Method/URL mapping

The following tables describe the Restful Methods supported by the Capture Media Resource.

Method	URL: {base-uri}/captures/{capture-id}/capture-medias
GET	Get all Media associated with the specified Capture Event. **Reply Content:** "capture-medias" collection XML containing "capture-media" summary elements. **Errors:** Capture Event not found.

Method	URL: {base-uri}/capture-medias/{capture-media-id}
GET	Retrieve the detailed information on the Capture Media. **Reply Content:** "capture-media" detailed XML.. **Errors:** not found.

Device Entity (device) / Collection (devices) [New in 2.6]

The Device entity represents the Lecture Capture and Presentation Processing devices registered with the Echo System. Most of the Device attributes are read-only status information used to monitor the device health. The Scheduling API currently supports read-only access to the device entities.

Device Entity's XML mapping

This table defines the mapping of Device attributes/fields for its various XML representations (summary or detailed).

Field / Element	Rep. Levels	Type (length)	Information
id	All	GUID	System generated product identifier.
key	All	string	Unique MAC address of the device.
type	All	enum string	Choices are: "software-capture", "hardware-capture", "pc-capture", "media-processor", "personal-capture".
description	Detailed	string	Optional text description of the Device.
address	Detailed	string	Device network address.
external-address	Detailed	string	Optional, alternative network address.

operational-mode	Detailed	enum string	Choices are: "fixed".
last-capture-status	Detailed	enum string	Choices are: "active", "complete", "error", "idle", "na", "new", "overdue", "error-time-sync", "paused", "unknown", "waiting".
last-log-upload-status	Detailed	enum string	See last-capture-status.
first-status-time	Detailed	timestamp	When the Echo System first received a status package from the Device.
last-status-time	Detailed	timestamp	When the Echo System last received a status package from the Device. Used to determine overdue devices.
next-status-time	Detailed	timestamp	Max threshold when the Echo System should receive the Device's next Status packet.
last-start-time	Detailed	timestamp	When the Device last cycled power/operation.
last-client-status-time	Detailed	timestamp	Last time when a device client (Ad hoc) communicated directly with the device.
time-out-of-sync-for-alert	Detailed	boolean	True if the Device internal clock has drifted from the Echo System clock and an Alert was generated for this condition.
time-out-of-sync-for-reject	Detailed	boolean	True if the Device internal clock has drifter from the Echo System clock; to the point that the Echo system will reject all communication from the device.
platform/os	Detailed	string	The Device's Operating System type/version.
platform/processor	Detailed	string	The Device's CPU brand.
os-details	Detailed	string	More details on the Device OS.

REST Method/URL mapping

The API supports listing the Device entries. The following tables describe the Restful Methods supported by the Device Resource.

Method	URL: {base-uri}/devices
GET	Get a list of all Devices defined in the Echo System. **Reply Content:** "devices" collection XML containing "device" summary elements.

Method	URL: {base-uri}/rooms/{room-id}/devices
GET	Get a list of Devices associated with the given Room. **Reply Content:** "devices" collection XML containing "device" summary elements.

Method	URL: {base-uri}/buildings/{building-id}/devices
GET	Get a list of Devices associated with the given Building. **Reply Content:** "devices" collection XML containing "device" summary elements.

Method	URL: {base-uri}/devices/{device-id}

GET	Retrieve the detailed information on the specified Device. **Reply Content:** "device" detailed XML.. **Errors:** Specified Device was not found.

Appendix A: Summary Request URL Listing

"/v1/people" Resource: Person - List all People, or Add a new Person.
"/v1/people/{person-id}" Resource: Person - Get, Modify, or Delete a Person.
"/v1/campuses" Resource: Campus - List all Campuses, or Add a new Campus.
"/v1/campuses/{campus-id}" Resource: Campus - Get, Modify, or Delete a given Campus.
"/v1/campuses/{campus-id}/buildings" Resource: Building - List Buildings within a Campus, or Add new Building to a Campus.
"/v1/buildings" Resource: Building - List (only) all Buildings across all Campuses.
"/v1/buildings/{building-id}" Resource: Building - Get, Modify, or Delete a Building.
"/v1/rooms" Resource: Room - List (only) all Rooms.
"/v1/campuses/{campus-id}/rooms" Resource: Room - List Rooms within a specified Campus.
"/v1/buildings/{building-id}/rooms" Resource: Room - List Rooms within a Building, or Add a new Room to a Building.
"/v1/rooms/{room-id}" Resource: Room - Get, Modify, or Delete a Room.
"/v1/products" Resource: Product - List (only) all Capture/Presentation Products.
"/v1/products/{product-id}" Resource: Product - Get (only) details of a Product.
"/v1/publishers" Resource: Publisher - List (only) all Publishers.
"/v1/publishers/{publisher-id}" Resource: Publisher - Get (only) details of a Publisher.

"/v1/terms" Resource: Term - List all Academic Terms, or Add a new Term.
"/v1/terms/{term-id}" Resource: Term - Get, Update, or Delete an Academic Term.
"/v1/terms/{term-id}/excluded-dates" Resource: Term.ExcludedDates collection. List the Term's excluded dates, Clear the Term's excluded dates collection, or Add a date range to the Term's excluded date collection.
"/v1/courses" Resource: Course - List all Courses, or Add a new Course. *Depending on the System Configuration, Adding a new Course can auto generate a default Section.
"/v1/terms/{term-id}/courses" Resource: Course - List (only) all Courses with at least 1 section within the Academic Term.
"/v1/courses/{course-id}" Resource: Course - Get, Update, or Delete a Course.
"/v1/terms/{term-id}/sections" Resource: Section - List (only) all Sections within the Academic Term.
"/v1/courses/{course-id}/sections" Resource: Section - List (only) all Sections for a Course (across Terms).
"/v1/terms/{term-id}/courses/{course-id}/sections" Resource: Section - List all Sections for a given Course and Academic Term, or Add a new Section to the Course for the given Term.
"/v1/sections/{section-id}" Resource: Section - Get, Modify, or Delete a given Section.
"/v1/sections/{section-id}/presenters" Resource: Section.Presenter collection - List or Clear the Section's Presenter collection.
"/v1/sections/{section-id}/presenters/{presenter-id}" Resource: Section.Presenter collection - Direct (empty content) Add or Remove of the Person to/from the Section's Presenter list. (presenter-id = person-id). A GET can be used to confirm the Person is already a Presenter of the Section.
"/v1/sections/{section-id}/publish-channels" Resource: PublishChannel - List or Clear the Section's Publish Channel collection.
"/v1/publish-channels/{publish-channel-id}" Resource: PublishChannel - Get, Modify, or Delete the Section's Publish Channel.
"/v1/publishers/{publisher-id}/sections/{section-id}/publish-channels" Resource: Section.PublishChannel - List, Add, or Clear the Section's Publishing Channels that target the given Publisher.
"/v1/sections/{section-id}/publishers/{publisher-id}/publish-channels" Resource: Section.PublishChannel - List, Add, or Clear the Section's Publishing Channels that target the given Publisher.
"/v1/publishers/{publisher-id}/publish-channels" Resource: Section.PublishChannel - List (only) all Publish Channel that use the given Publisher. (could be a big data set)
"/v1/sections/{section-id}/products" Resource: Section.Product collection - List or Clear the Section's assigned Capture Products.

"/v1/sections/{section-id}/products/{product-id}" Resource: Section.Product collection - Direct (empty content) Add or Delete of the Product to/from the Section's associated Products. A GET can be used to confirm the presents of Capture Product.
"/v1/sections/{section-id}/schedule-rules" Resource: ScheduleRule - List all Schedule event Rules for a Section, or Add a new Schedule event Rule to the Section.
"/v1/terms/{term-id}/schedule-rules" Resource: ScheduleRule - List (only) Schedule event Rules for an academic Term.
"/v1/schedule-rules/{schedule-rule-id}" Resource: ScheduleRule - Get, Modify, or Delete a given Schedule event Rule.
"/v1/schedule-rules/{schedule-rule-id}/products" Resource: ScheduleRule.Product collection - List the Products assigned to the Schedule event Rule, or Clear the entire assigned Product collection.
"/v1/schedule-rules/{schedule-rule-id}/products/{product-id}" Resource: ScheduleRule.Product collection - Direct (empty content) Add or Delete of a Product assignment for a Schedule event Rule. A Get could be used to confirm product assignment.
"/v1/schedule-rules/{schedule-rule-id}/rooms" Resource: ScheduleRule.Room collection - List the Rooms that are/were assigned to the Schedule event Rule. DELETE will Unassign the currently assigned room.
"/v1/schedule-rules/{schedule-rule-id}/rooms/{room-id}" Resource: ScheduleRule.Room collection - Direct (empty content) Add or Delete of the currently assigned room.
"/v1/schedule-rules/{schedule-rule-id}/excluded-dates" Resource: ScheduleRule.ExcludedDate collection - List or Clear the given schedule event Rule's Excluded Dates collection. Also, Add a new date range to the excluded date collection.
"/v1/schedule-rules/{schedule-rule-id}/presenters" Resource: ScheduleRule.Presenter collection - List or Clear the Schedule event Rule's currently assigned Presenters.
"/v1/schedule-rules/{schedule-rule-id}/presenters/{presenter-id}" Resource: ScheduleRule.Presenter collection - Direct Add or Remove of a Presenter to/from the Schedule event Rule's Presenter collection. A Get can be used to confirm the Person is a Presenter for the Schedule event Rule.
"/v1/schedule-rules/{schedule-rule-id}/captures" Resource: Capture - List (only) all Captures for the given Schedule event Rule.
"/v1/rooms/{room-id}/captures" Resource: Capture - List (only) all Captures for the given Room.
"/v1/captures/{capture-id}" Resource: Capture - Get (only) details on a given Capture.
"/v1/captures/{capture-id}/capture-medias" Resource: Capture.Media - List (only) the media which was generated by a given Capture.

"/v1/capture-medias/{capture-media-id}"	Resource: Capture.Media - Get (only) details for a given Capture Media.
"/v1/devices"	Resource: Device - Summary List (only) of all Devices.
"/v1/devices/{device-id}"	Resource: Device - Get (only) the details of the Device.
"/v1/rooms/{room-id}/devices"	Resource: Device - List (only) of Devices associated to the Room.
"/v1/buildings/{building-id}/devices"	Resource: Device - List (only) of Devices associated to the Building.

Appendix B: Release Change History

Release 2.5.x - Introduction of the scheduling REST API in the Echo System.
Release 2.6.x: Since Pre-Release (RC1) - Support for Capture Product and Publishing.

1. Expose the Capture "Product" Resource as read-only.
2. Support assigning "Products" to Sections.
3. Support assigning "Products" to Schedule event Rules.
4. Expose the "Publisher" Resource as read-only.
5. Support assigning "Publish" Channels to Sections.
6. Support generic Section Properties.
7. Support generic Section.PublishChannel Properties.
8. Readonly support for Device information.

4.0 Schedule API

In this section:

Overview

This page describes the differences between the 4.0 Schedule API and earlier versions of the API. See the 2.5 Schedule API Specification and 2.6 Schedule API for details.

Products Replaced with Product Group(s)

In 3.0 and earlier, there were API calls to obtain products for a specific Section/ScheduleRule/ExternalMedia ID and a specific product and all products. In 4.0, they are replaced with API calls to obtain product groups for Section/ScheduleRule/ExternalMedia ID and a specific product group and all product groups.

URI pattern in 3.0	URI pattern in 4.0	Descript
/v1/products	/v1/product-groups	Get all pr groups
/v1/products/(product-id)	/v1/product-groups/(product-group-id)	Get a pro group in ID (the d includes name)

/v1/sections/(section-id)/products	/v1/sections/(section-id)/product-groups	Get/Dele product g section b ID. This A returns c with a sir product g for GET operatior
/v1/schedule-rules/(schedule-rule-id)/products	/v1/schedule-rules/(schedule-rule-id)/product-groups	Get/Dele product g schedule schedule This API returns c with a sir product g for GET operatior
/v1/external-medias/(external-media-id) /products	/v1/external-medias/(external-media-id)/product-groups	Get prod for extern by extern ID. This A returns c with a sir product g it.
/v1/sections/(section-id)/products/(product-id)	/v1/sections/(section-id)/product-groups/(product-group-id)	Add/upda product g a section
/v1/schedule-rules/(schedule-rule-id)/products/(product-id)	/v1/schedule-rules/(schedule-rule-id)/product-groups/(product-group-id)	Add/upda product g a schedu

In order to create an active schedule, product group is a required field in 4.0, while it was optional in 3.0. This means that, when creating a new active schedule using API call, the product group name needs to be enclosed in 'product-group' xml tags and sent in the xml request. Below is the sample XML to create an active schedule rule.

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule-rule>
<title>test_adding_schedule_rule</title>
<status>active</status>
<description>test</description>
<start-date>2011-08-22</start-date>
<start-time>14:00</start-time>
<duration-seconds>60</duration-seconds>
<recurring>0</recurring>
<presenters>3c0d918d-dcf0-4f5b-8fc0-b9a2fbb9b9b0</presenters>
<product-group>
<name>my_product_group_name</name>
</product-group>
<aspect-ratio-override1>Standard</aspect-ratio-override1>
<aspect-ratio-override2>Standard</aspect-ratio-override2>
</schedule-rule>
```

Aspect Ratio Overrides

When creating a new schedule, the user can override the aspect ratio for primary and secondary display now by passing aspect ratio value as a string within 'aspect-ratio-override1' and 'aspect-ratio-override2' xml tags. 'aspect-ratio-override1' corresponds to overriding aspect ratio for primary display and 'aspect-ratio-override2' to overriding aspect ratio for secondary display.

Device Configurations

The API call to get a device's details will now additionally enclose the device profile name in 'device-profile' xml tags.

DLA

New API calls for organization

URI pattern in 4.0	Description
/v1/organizations	Get all organizations
/v1/organizations/(organization-id)	Get an organization in detail by ID
/v1/organizations	Add an organization

Add Organization XML

```
<organization>
          <name>(name of the organization-required)</name>
          <parent-id>(parent organization id-required)</parent-id>
          <description>(not required)</description>
          <customer-identifier>(Customer License identifier- not required and not applicable for
4.0)</customer-identifier>
       </organization>
```

Add Course XML

URL: v1/courses

```
<course>
   <name>(name of the course-required)</name>
   <organization-id>(organization id of the organization where course belongs
to-required)</organization-id>
   <identifier>(course identifier-required)</identifier>
</course>
```

Add Person XML

URL: v1/people

```
<person>
   <first-name></first-name>
   <last-name></last-name>
   <email-address></email-address>
   <website></website>
   <credentials>
     <user-name></user-name>
     <password></password>
   </credentials>
   <organization-roles>
      <organization-role>
         <organization-id>(organization id where person is assigned a role)</organization-id>
         <role>(role in this organization)</role>
      </organization-role>
      <organization-role>
          <organization-id></organization-id>
          <role></role>
      </organization-role>
    </organization-roles>
 </person>
```

Add Room

URL: v1/buildings/(building-ID)/rooms

```
<room>
  <name>(room name)</name>
  <organization-id>(organization room belongs to)</organization-id>
</room>
```

Add Section

URL: v1/terms/(term-id)/courses/(course-id)/sections</url>

```
<section>
   <name>(section name-required)</name>
   <section-roles>
      <section-role>
         <person-id>(UUID of the person being assigned this role)</person-id>
         <role-name>(section role name)</role-name>
      </section-role>
      <section-role>
         <person-id></person-id>
         <role-name></role-name>
      </section-role>
  </section-roles>
</section>
```

Add Section User

URL: v1/sections/(section-id)/presenters/(presenter-id)

presenter should belong to "Academic Staff" group for the section's organization. Also API call will add this person only as a GUEST_PRESENTER.

Add Term

URL: v1/terms

```
<term>
   <name>(Term Name)</name>
   <organization-id>(UUIOD of Organization for this Term)</organization-id>
   <start-date>2011-06-12</start-date>
   <end-date>2011-07-30</end-date>
   <excluded-dates>
     <date-range><start>2009-07-04</start><end>2009-07-04</end></date-range>
     <date-range><start>2009-07-05</start><end>2009-07-05</end></date-range>
     <date-range><start>2009-07-06</start><end>2009-07-06</end></date-range>
   </excluded-dates>
  </term>
```

Other New API calls

URI pattern in 4.0	Description
/v1/sections/(section-id)/captures	Get all captures for a section
/v1/captures/(capture-id)/presentations	Get all presentations for a capture
/v1/presentations/(presentation-id)	Get a presentation in detail by ID

Capture Appliance API Specification

In this section:

Overview

The API for the EchoSystem capture appliance provides a mechanism for an external system, such as an In-Room Control System (IRCS) or a custom desktop monitoring application, to interact with the capture device to control and monitor the capture functions. The API provides access to capture status information, allows the initiation of ad-hoc captures, and enables record/pause/resume/stop functions using a customizable interface.

There are two commonly accepted ways of communicating between an IRCS and any device: serial communication or IP-based communication. The EchoSystem capture appliance only supports IP-based communication. This IP-based communication is by way of a web service, or REST API. There are several reasons for this choice:

1. Serial communication is much slower, and the latency incurred by this slower communication would be unacceptable for pause/resume and start/stop commands.
2. Serial communication requires a dedicated cabling solution for each device that will be controlled, whereas IP-based communication can occur across cabling resources that are already present in both the capture device and the IRCS (the Ethernet).
3. We want to provide a common interface for all EchoSystem capture devices. The capture device is sometimes a capture appliance, but might also be an installation of Classroom Capture on a dedicated podium PC. Many new PCs don't have serial ports at all.
4. Most vendors of IRCS solutions have indicated that IP-based communication is the "future proof" path for their current and future generation IRCS products.

This document will describe how to use each of the REST API functions and describes the communication between the capture device and IRCS. The target audience for this document is primarily programmers of IRCS solutions who are designing the IRCS touch panel interface, or programmers building a custom application (applet, web application, etc.) for integrating with the capture device.

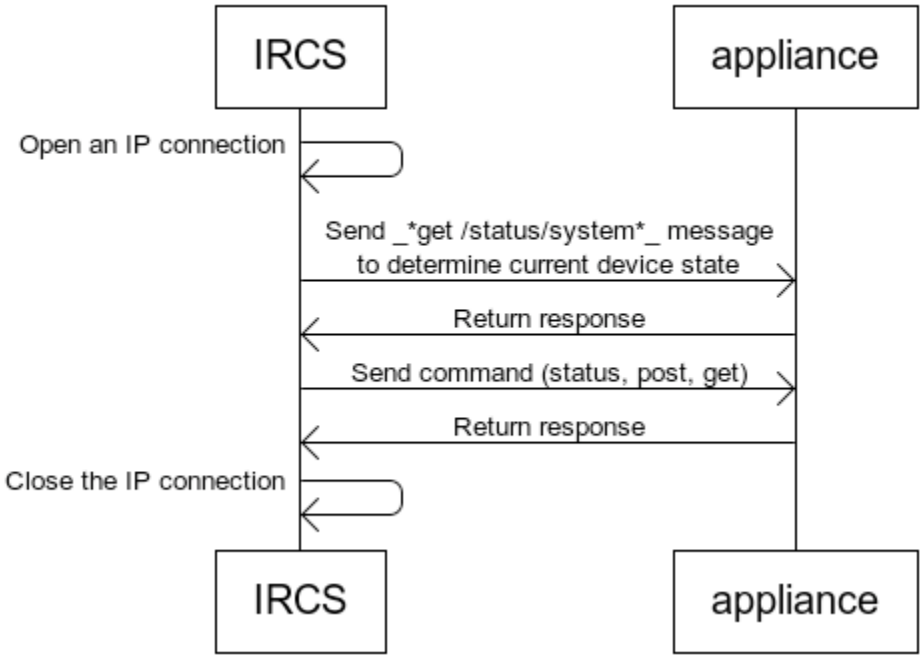
Workflow

The capture appliance provides a built-in web server, which is used to display the appliance's web UI for local status and configuration. The web UI is using the same REST API commands that the IRCS will be using to communicate with the appliance. This is important to note, because it means that *anything* you see in the web UI can also be done from an IRCS integration, via the REST API.

The communication between the capture appliance and the IRCS can be described as follows:

1. The ICRS opens an IP connection to the appliance
2. The ICRS sends a _*get/status/system command to determine the appliance's current state
3. The appliance returns a response
4. The ICRS sends a command (status, post, get) to the appliance
5. The appliance returns a response
6. The ICRS closes the connection

This communication is illustrated in the following diagram:



About Capture Device API Version 2.2

In this section:

Overview

The capture appliance API v2.2 is backward compatible with all previous versions of the API.

The v2.2 release adds the ability to query the capture appliance for local confidence monitoring attributes, including audio level and thumbnails of currently capturing video and vga. The audio attributes are being updated on ~1 second intervals, and the video and VGA attributes are being updated on ~10 second intervals, so querying the system more frequently does not provide greater granularity. This version also adds the ability to start ad hoc captures pre-associated with a section in EchoSystem Server. This type of ad hoc capture automates processing and publishing of the capture based on the course information.

Known Issues

The 2.2 version of the REST API contains the following known issues. Workarounds have been noted if available.

/Capture/New_Capture Method Backward Compatibility

All versions of the API are compliant with standard HTTP/1.1 header formats, as per the following example:

FORMAT

```
[GET | POST] <method> <format; always "HTTP/1.1">
<Authentication parameters>
[if POST w/parameters:  Content-Length = <data set length>]
(blank line)
<data parameters>
(blank line)
```

EXAMPLE

```
POST /capture/new_capture HTTP/1.1
Authorization: Basic aW5zdHJ1Y3RvcjpwYXNzd29yZA==
Content-Length: 87

duration=300&capture_profile_name=display-audio&description=%22test%20description%22
```

However, the EchoSystem v2.5 device software enforces the optional "Content-length" parameter, making it mandatory, whereas the previous device software releases allowed this value to be optional. This may cause previous integrations that were more lax in their implementation to be unable to start an ad hoc capture.

Single Connection to API

The capture appliance uses a web server that processes a single connection at a time. For most implementations, this doesn't create an issue, because the appliance opens the connection, processes the request, and immediately closes the connection. This process takes only a fraction of a second and generally assures that the appliance is available for another request (either from the same client, or a different client). As such, we do not currently support "http-keepalive" messages.

We have found a few examples of IRCS programming implementations where the code assumes that the connection is always open, and in some cases

1. will send a partial message while the code completes the process of building the remainder of the message, *or*
2. will open the connection without sending any message, and will then attempt to hold the connection open using the http-keepalive method.
   Both cases will result in an error.

In the first case, the appliance holds the port open to await the completion of the message, and will block any other client that attempts to connect to the API or the built-in webpages (which are also using the REST API). This isn't a fatal error, it's just that the second client will experience very long waits, and possibly connection timeouts, when trying to access the API or web UI.

In the second case, the appliance will simply idle timeout the connection and close the connection so that it's available for another client. The way that the IRCS handles this closed connection is system dependent. In most cases (such as AMX systems and modern Crestron systems), the IRCS will recognize the closed port and simply reopen it, without a code interaction needed. In other cases (such as inexpensive brands that treat the IP port as a really fast serial port), the IRCS will not recognize the close connection and will cause a message-send failure that must be handled in code.

The solution, in both cases, is to surround the actual send/receive communication with an "open-port / close-port" pair, effectively making explicit the state of the communication port.

Four (4) Hour Captures

The EchoSystem has been officially qualified to support captures up to four hours in duration. The EchoSystem UI currently restricts captures to this four hour limit, however, API does not currently impose this limit.

Active Ad Hoc Captures

Capture submission with the *new_capture* command, which will start a new ad hoc capture, will force a running ad hoc capture to stop immediately. This will not affect scheduled captures as they always have priority over ad hoc captures.

SSL Certificate

The Capture Appliance uses a self-signed SSL certificate for the default HTTPS connection to the REST API. The self-signed certificate may cause connection issues in your programming environment. There are three workarounds for this issue:

Option 1: Obtain a Valid Trusted Certificate (recommended)

If you are concerned that the capture appliance will be subjected to security threats, then Echo360 recommends that the SSL certificate for EchoSystem devices be overridden with a valid trusted certificate. This process is briefly outlined below:

1. Obtain a wild card certificate from a trusted root authority (Verisign, Thawte, etc) for your domain (e.g. *.some.university.edu). This must be a .pem certificate file.
2. Rename the .pem certificate to device_server.pem
3. Copy device_server.pem to the /server/etc directory on the EchoSystem Server (ESS).
4. Restart the ESS.
5. The capture appliances (all of them) will download the new certificate shortly after the ESS restarts.
6. After the certificate has been successfully installed, each device must only be contacted with the Fully Qualified Domain Name (FQDN) that was specified in the certificate. The wildcard certificate allows the use of a single certificate for all appliances within the same domain.

Option 2: Create a Certificate with a Local Certificate Authority (not recommended)

If a valid trusted certificate cannot be obtained, a local certificate authority (CA) can be used to generate the certificate. This method will require the root certificate from the local CA to be accessible by the application communicating with the capture device API. This will vary between the many programming environments and is therefore not recommended due to the additional programming complexities. This process is briefly outlined below:

1. Obtain a wild card certificate for your domain from a local CA (requires network administrator access). This must be a .pem certificate file.
2. Rename the .pem certificate to device_server.pem.
3. Copy device_server.pem to the /server/etc directory on the EchoSystem Server (ESS).
4. Restart the ESS.
5. The capture appliances (all of them) will download the new certificate shortly after the ESS restarts.
6. Configure your application to trust the the local certificate authority. This will usually require importing the root certificate into a certificate store accessible by the application.

Option 3: Configure Devices for HTTP Connections (not recommended)

Each device can also be configured to accept HTTP connections instead of HTTPS connections. This will bypass the need for any certificates. This process is briefly outlined below:

1. Browse to the Configuration > Devices tab in the ESS UI.
2. Highlight the appropriate device and click the *Edit* link.
3. Check the *Ad Hoc Control Service - HTTP Access (instead of HTTPS)?* check box.
4. Set the port for the HTTP listener in the *Ad Hoc Control Service - Port* field.
5. *Save* the configuration.
6. Repeat these steps for each device being controlled through the REST API.
7. The devices will now listen on the specified port using the HTTP protocol. Your application should be programmed accordingly.

Capture Device API v2.2 - Sample HTML Programs

In this section:

Sample HTML Programs

You might find it helpful to run these sample HTML pages, observing the way they interact with a capture appliance's REST API. You can observe the conversation with a packet sniffer application, such as Wireshark (http://www.wireshark.org/) or similar protocol analyzer.

To use these samples, open a text editor, cut and paste the samples below to your text editor, save the snippet as the appropriate .htm or .html file, and then run the HTML page in your favorite browser.

getSystemStatus.html

This example page is a very simple form to query the overall system status. You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<body>
<form action="http://10.3.10.152:8080/status/system" method="get">
<input type="submit"/>
</form>
</body>
</html>
```

getCaptureStatus.html

This is also a very simple example form, this time to query the system for the current and upcoming captures. You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<body>
<form action="http://10.3.10.152:8080/status/captures" method="get">
<input type="submit"/>
</form>
</body>
</html>
```

createNewBasicAdHocCapture.html

This example page opens a form that allows you to create a basic Ad Hoc capture. You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

CHANGE URL TO /confidence_monitor and this will submit a confidence monitor capture.

```
<html>
<body>
<form action="http://10.3.10.152:8080/capture/new_capture" method="post">
description:<input type="text" name="description"/>
duration (seconds):<input type="text" name="duration"/>
capture_profile_name:<select name="capture_profile_name">
<option value="audio">audio</option>
<option value="display-audio">display-audio</option>
<option value="display-audio-video">display-audio-video</option>
</select>
<input type="submit"/>
</form>
</body>
</html>
```

pauseCapture.html

This example pauses our currently active capture (either scheduled or ad-hoc). You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<body>
<form action="http://10.3.10.152:8080/capture/pause" method="post">
Pause: <input type="submit"/>
</form>
</body>
</html>
```

extendCapture.html

Now we're going to extend the duration of the currently active capture (either scheduled or ad-hoc). You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<form action="http://10.3.10.152:8080/capture/extend" method="post">
duration (seconds):<input type="text" name="duration"><BR>
<input type="submit" name="extend"><BR>
</form>
</html>
```

recordCapture.html

This example allows you to either resume a paused capture, or to make a scheduled capture start early. You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<body>
<form action="http://10.3.10.152:8080/capture/record" method="post">
Record Now\! <input type="submit"/>
</form>
</body>
</html>
```

stopCapture.html

Finally, this example allows you to stop the current capture, whether it is paused or actively recording. You will need to change the IP address, port, and protocol (https vs. http) to match your capture device.

```
<html>
<body>
<form action="http://10.3.10.152:8080/capture/stop" method="post">
Stop Recording <input type="submit"/>
</form>
</body>
</html>
```

Capture Device API v2.2 - Workflow Example

In this section:

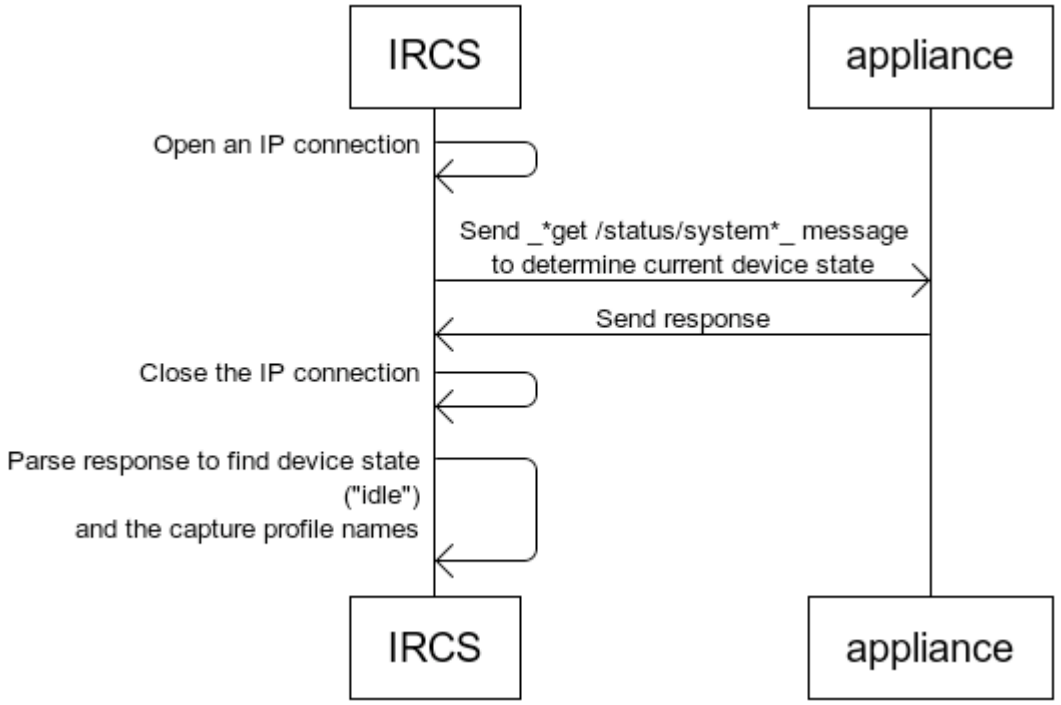
Workflow Example

There may be some confusion about the proper way to form the commands that are sent to the REST API, and the sorts of responses that you should expect. The following examples are taken from the logs of a working conversation between a very simple control application (in this case, PuTTY running a Raw connection with no encoding or formatting) and an EchoSystem capture appliance. The appliance is configured to use HTTP rather than HTTPS (so that the PuTTY client can forgo encryption demands) and the port was set to 8080 rather than the default, 8443.

In the below examples, red italics is sent by the client, the black text is response from the appliance.
\n represents a 'newline' character.

Query System Status

This first example queries the capture appliance for the system status. This call is used frequently before sending an action command (start, stop, pause, new_capture) so that you will not conflict with another action.

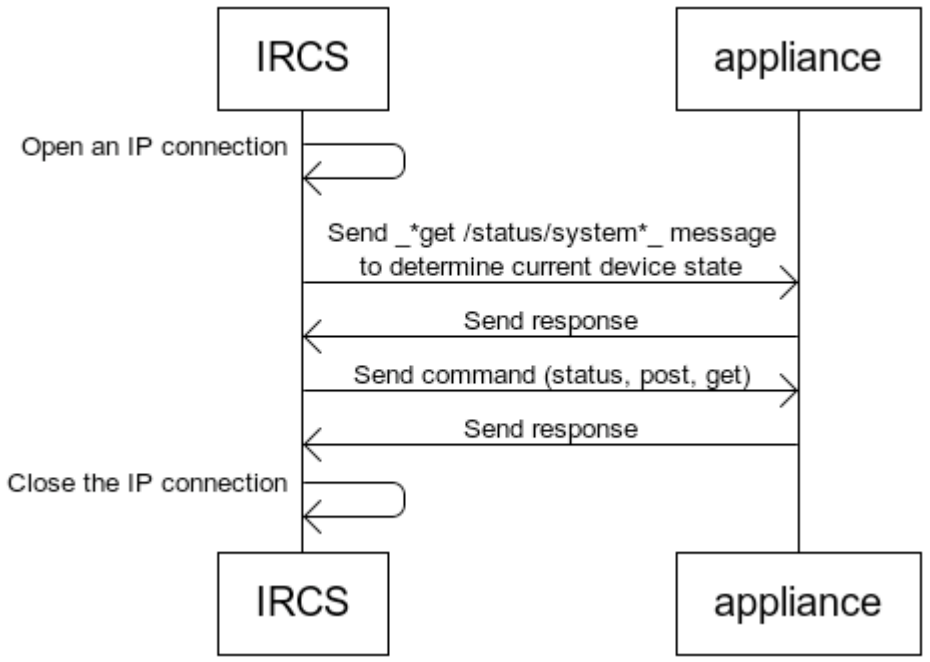


GET /status/system HTTP/
Authorization: Basic aW5zc
\n

HTTP/1.1 200 OK
Server: hardware-capture/2
Date: Thu, 14 Aug 2008 19
Content-Type: text/xml
Content-Length: 1155
Cache-Control: max-age=0
Expires: Thu, 14 Aug 2008
Connection: close
<status>
<wall-clock-time>2008-08-1
<capture-profiles>
<capture-profile>audio </ca
<capture-profile>display-au
<capture-profile>display-au
</capture-profiles>
<host-address>echo</host-
<serial-number>00-50-c2-2
<system-version>2.2.3541<
<up-since>2008-08-14T16:
<last-sync>2008-08-14T18
<content>
<state>idle</state>
<archive-space-usage />
<uploaded>0</uploaded>
<uploads-pending>0</uploa
<bytes-pending>0</bytes-p
<uploading>false</uploadir
<log>
<state>error</state>
<archive-space-usage>56.8
<uploaded>40</uploaded>
<uploads-pending>3</uploa
<bytes-pending>179889</b
<uploading>false</uploadir
<upload>
<bytes-per-second>0</byte
<filename>18.58.23.yaml</
<start-time>2008-08-14T19
</upload> </log>
<location>Dulles: Echo360-
</status>

Starting a New Capture

One of the more common workflows is to create a new capture that was not previously scheduled.

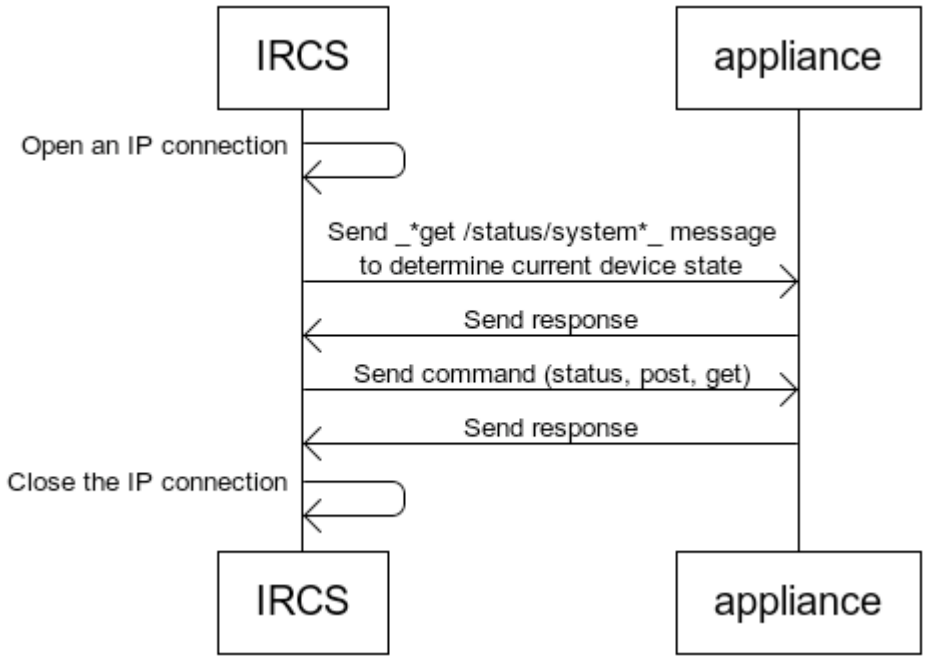


POST /capture/extend HTTP/1.1\n
Authorization: Basic
aW5zdHJ1Y3RvcjpwYXNzd29yZA==\n
\n

Other stuff...

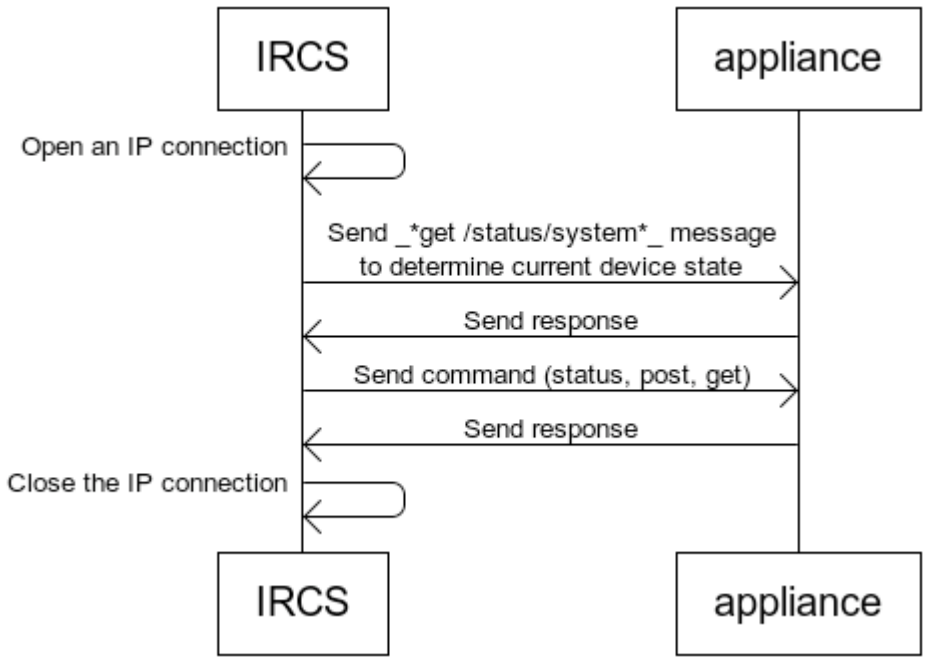
Extending a Capture

The diagrams and workflow examples below show the process for extending a capture, pausing a capture, and stopping a capture.



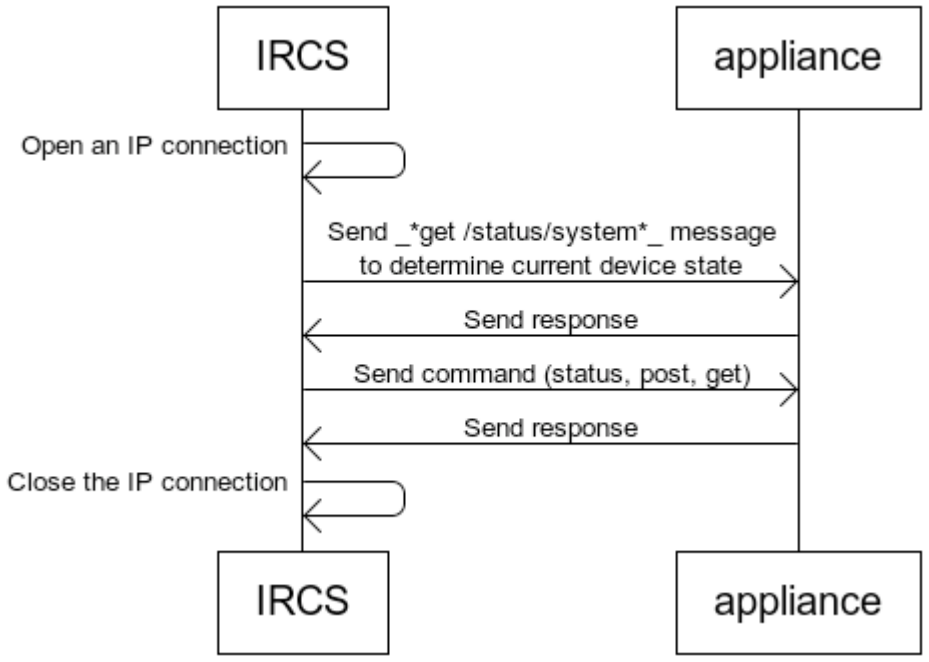
POST /capture/extend HTTP/1.1\n
Authorization: Basic
aW5zdHJ1Y3RvcjpwYXNzd29yZA==\n
\n

HTTP/1.1 200 OK
Server: hardware-capture/2.2.3541
Date: Thu, 14 Aug 2008 19:09:46 GMT
Content-Type: text/xml
Content-Length: 42
Cache-Control: max-age=0
Expires: Thu, 14 Aug 2008 19:09:46 GMT
Connection: close
<ok text="Extend by seconds recieved" />



POST /capture/pause HTTP/1.1\n
Authorization: Basic
aW5zdHJ1Y3RvcjpwYXNzd29yZA==\n
\n

HTTP/1.1 200 OK
Server: hardware-capture/2.2.3541
Date: Thu, 14 Aug 2008 19:10:48 GMT
Content-Type: text/xml
Content-Length: 40
Cache-Control: max-age=0
Expires: Thu, 14 Aug 2008 19:10:48 GMT
Connection: close
<ok text="Command (pause) submitted" />



POST /capture/stop HTTP/1.1\n
Authorization: Basic
aW5zdHJ1Y3RvcjpwYXNzd29yZA==\n
\n

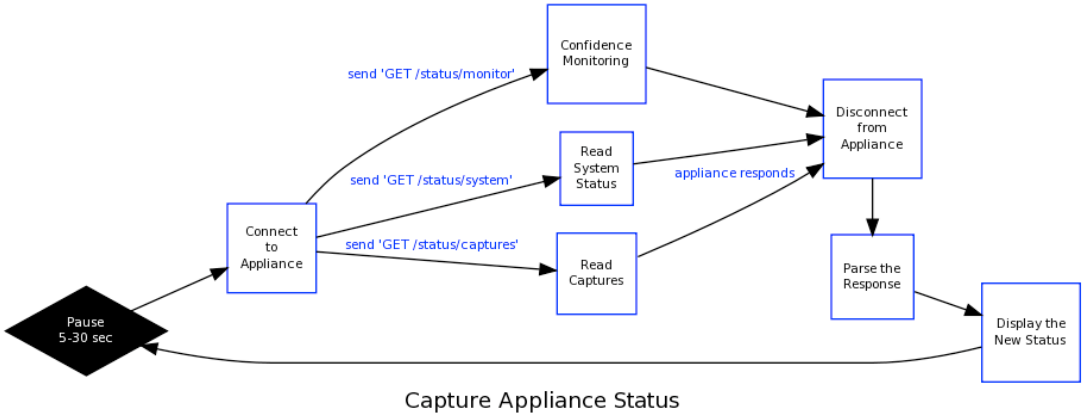
HTTP/1.1 200 OK
Server: hardware-capture/2.2.3541
Date: Thu, 14 Aug 2008 19:12:03 GMT
Content-Type: text/xml
Content-Length: 39
Cache-Control: max-age=0
Expires: Thu, 14 Aug 2008 19:12:03 GMT
Connection: close
<ok text="Command (stop) submitted" />

Typical Use Cases

In this section:

Displaying Capture Appliance Status

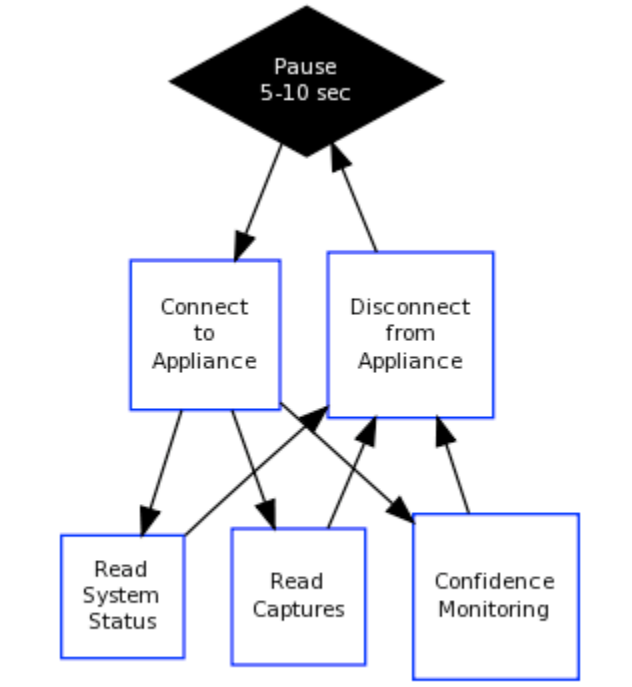
This use case shown in the flowchart below also includes the ability to perform local confidence monitoring of the audio, video, and VGA sources.



Capture Appliance Status

Working with Captures

The use case shown in the flowchart below shows the communication paths for working with captures from the appliance.



Another Capture Appliance Status

Capture Device API Version 3.0

In this section:

- Commands

Commands

The Capture API document lists API commands relevant for 3.0 capture devices. These commands can be used with:

- The EchoSystem Capture Appliance
- The EchoSystem SafeCapture HD
- Classroom Capture