Axis Public API

Axis needs a public API so that customers can integrate Axis functionality with other applications, most importantly customers that use an LMS to manage their learning want to be automatically synchronize data between the LMS and Axis.

The Axis API will allow customers to work with two primary objects: events and seats. Training events and seats are objects that the LMS also needs to manage (though they are often called courses and registrations) so it is important they be kept in sync. In addition certain LMS users may need to be kept in sync with Axis, mainly LMS users who will be Axis event owners and instructors. The Axis API will not allow customers to work with Axis-specific objects like templates and lab plans. The definition of such objects is specific to Axis and it would not make sense to allow customers to build a UI for managing them in their LMS. They should be managed through the Axis UI.

Although the Axis API will allow customers to manage events and seats (and to some extent users) it be focused on tasks that make sense to automate. These include tasks like:

- Creating and canceling events
- Adding and removing seats for an event
- Assigning a user to a seat
- Changing the instructor or dates/schedule of an event

These tasks are all part of the normal workflow of managing events and seats. On the other hand, the following tasks are more exceptions to the normal workflow. These kinds of one-time tasks are best done using the Axis UI.

- Changing the owner of an event
- Changing the template of an event
- Changing the schedule for an individual seat in an event
- Changing the email address for an instructor

It doesn't make sense for a customer to rebuild the Axis UI in their LMS for performing these rare one-time actions.

Express Gateway

API users will not talk directly to Axis. Doing so make it too easy for a badly written application or rouge programmer to have a negative impact on Axis and all the other Axis users. Instead we will make use of an API management solution which will sit between Axis and the API users (LMS applications). This API management solution will provide several important features including the ability to authenticate API

users, track metrics around API usage, rate limit usage, cache responses, and load balance requests. Although there are many API management solutions, including the API gateway provided by Amazon, we have decided to use an open source server called Express Gateway. There are several key reasons why we think this is a good option for us. First it is a mature application with strong user base. The programming language it uses is Javascript is one all our developers know. It uses simple json config files to configure the frontend and backend behavior of the API. Using such files, as opposed to a GUI interface, allows us to check the API config into our code repository just like the rest of our code.

Using the API

This section describes the usage and behavior of the API. The Axis API will follow REST principals, so the protocol used to communicate with the API will be HTTP. The base endpoint URL will be:

http://axis-api.readytech.com/v1

Including the version in the URL allows us to simultaneously support multiple versions of the API.

Authentication

The express gateway supports all the different authentication methods, including OATH2, but for simplicity I think we can start with basic auth. Customers who want to use the API will have to configure a user in Axis with API credentials. As a best practice they should configure a new user for this purpose, rather than use an existing/real user. The API credentials will be different from the user's normal username and password. This is necessary to ensure that regular users cannot make API calls.

JSON

Request and response data will be formatted in JSON.

HTTP Methods

Consistent with REST principals the API will align HTTP methods with API actions. POST will be used for creating objects. GET is for retrieving one or more objects. PUT is for updating objects. DELETE is for deleting objects. We will also support certain actions, like notify and cancel, and searches. Since these actions require additional data such as the fields you want to search on we will use the POST method for these actions.

HTTP Response codes and errors

The API will also make use of HTTP response codes to indicate whether the request was correctly processed or not. For example in most cases a 200 response code means the request was process while codes 4xx and 5xx indicate that something went wrong. When an error occurs the response body should include a json formatted message indicating more specifically what the error is, such as:

{ "status": "404", "message": "Requested record does not exist or the user cannot access it" }

Lookups

When posting data to the API you often need to specify a specific object such as a user who will be the instructor or owner of the event or the template the event should use. One approach the API could take in these situations is to require the customer to provide the ID/UUID of the object in the request, however, this ends up being a burden because before the customer can create an event they have make API calls to get all the IDs they need to make a request. For example, if you want to create an event you might have to do the following:

1. Query API to get ID of user who will be owner
2. Query API to get ID of user who will be instructor
3. Query API to get ID of template the event will use
4. Query API to get ID of the region the event will be assigned to
5. Make the API call to create the event

This is a pain. Of course the customer could query all the information one time and then store our ID in their LMS but not every LMS is flexible enough to provide a place to store our IDs and keeping the data up to date is difficult. A better approach is to provide an API that allows the customer to do lookups for data as part of the main request. A lookup uses one or more properties besides the primary key to identify a record. For example, instead of identifying the event owner by their Axis UUID the customer could identify them by email or even name. Here is how this would work in practice. Imagine I want to create a new event and specify the owner and template, I could specify these using their id.

”owner”: “yyyy-yyyy-yyyy-yyyy”,
”template”: “zzzz-zzzz-zzzz-zzzz”,
….

Alternatively I can use the lookup syntax and tell the API to find the user that matches the properties. To trigger a lookup you just use the json object syntax instead of a single value:

”owner”: { “email” : “peter@gmail.com” },
”template”: { “name”: “Introduction to SQL” }
….

As long as each lookup finds exactly one matching record the API request will succeed. This flexibility will result in much less code on the customer side and an easier integration.

From an implementation perspective we can implement the lookup functionality in the API gateway so it works like this:

1. Customer calls the API to create an event using a number of lookups
2. The API gateway parses the lookups and makes backend queries to Axis to get the ID for each property that uses a lookup

3. The API gateway substitutes the ID into the original API request and sends the request on to Axis.

One advantage of this approach is that it offloads so work from Axis. Instead of Axis receiving a request that requires complex lookups and substitutions, it can receive a request that uniquely identifies each object by ID. If we build some caching logic into the gateway some lookups may be done without talking to Axis at all.

ExternalId

LMSes, like most applications, generate unique identifiers for each object it stores. For example, every user in an LMS will have an ID as will every course template and scheduled event. For many customers developing the integration will be easier and more robust if Axis can support these external IDs. For example when the customer creates an event in Axis and associate their ID with it it will be easier for them to identify the event later if they want to change or cancel it. So we will add an externalId property to the following objects:

- Event
- Seat
- User
- Template

When they create an event, seat, or user using the API they can set the externalId property and then make lookup requests and identify objects using the externalId. In addition we should also add the externalId field to the Template object in Axis. Axis templates generally correspond to courses in an LMS and each course usually has some unique identifier so allowing customers to associate this identifier with the matching Axis template is useful. Although we will not (initially) have an API endpoint for creating or managing templates we can make this property available on the template edit form in Axis. Then they can use that property in lookups as shown above.

“template”: { “externalId”: “1001” },
”owner”: { “externalId” : “YYSODUSS },
…

We cannot treat the externalId as key, like uuid, because it is not required and we have no way to ensure uniqueness. That means customers have to use the lookup syntax to use it.

Seat Roles

In the Axis application we use descriptive role names, like instructor and student. Later when we support sales demos and trials we will have sales person and prospects and for certification exams we will have proctors and candidates. While these descriptive roles in the UI make events easier to understand they complicate the API. Which roles are available depend on the template type. For example instructor and student roles only apply to ILT events. In the UI this works because the user first selects the template and

then the UI can display the appropriate roles. However, for the API the template and role specifications must be provided in a single request. We could put the burden on API users to know which roles go with which event types but I think it is better to have the API use generic role names and let Axis save the appropriate name, based on the template type. So far the best generic names I have come up with are Leaders and Attendees. Here is how they should be used. When creating an event with the API the Leaders property will be specified such as:

“leaders” : “xxxx-xxxx-xxxx-xxxx”

Multi-value properties

There are a number of properties in the API which can take multi-values. For example, the leaders property of an event can specify a single leader or multiple leaders. For example to specify a single leader using their UUID use the following syntax:

“leaders”: “xxxx-xxxx-xxxx-xxxx”

Each event can also have multiple leaders and they can specified using the list syntax:

“leaders”: [“xxxx-xxxx-xxxx-xxxx”, “yyyy-yyyy-yyyy-yyyy”]

Also the lookup syntax can be used for some or all of the items in the list:

“leaders”: [{ “email”, “test@readytech.com” }, { “firstName”: “Steve”, “lastName”: “Smith” }, “zzzz-zzzz-zzzz-zzzz”]

Another example of a multi-value property is the attendees property. It can specify a simple integer indicating the number of attendees.

“attendees”: “10”

Or it can specific a list of attendees.

```
"attendees": [
    { “firstName”: “Steve”, “lastName”: ”Jones” },
    { “firstName”: “Marc”, “lastName”: ”Jacobs”, “email”: “mjacobs@gmail.com” },
    { “firstName”: “Carol”, “lastName”: ”Andrews”, “email”: “carol@yahoo.com” },
    { “firstName”: “Kevin”, “lastName”: ”Clint”, “externalId”: “123000” }
]
```

Event endpoints

Customers can create, update, search, and cancel events using the API. An event is a scheduled instance of template. The goal of an LMS - Axis integration is to keep the scheduled class data automatically in sync between Axis and the LMS. For example if an event is canceled in the LMS perhaps due to low enrollment it is automatically canceled in Axis to avoid the cost of spinning up the labs.

Creating an event

In general, the training schedule in the LMS should be synchronized with the event schedule in Axis. However, customers can decide when to have their LMS create the Axis event. For public instructor-led training it might make sense to create the event in Axis when a course is published (rather than when the event is initially created in the LMS). For customers onsites it might make sense to create the event when the course is confirmed or the contract is signed. For a self-paced event it probably makes sense to create it in Axis as soon as the student buys the course. Alternatively, a customer might decide to create all events in Axis some arbitrary number of days before they start, such as 3 days. It does not matter from an Axis perspective when the event is created as long as it is done early enough to successfully deploy the class (which might require an image copy).

To create an event, customers will POST to the /event endpoint and include the event details in the payload of the request as JSON. The following properties can be posted:

Property	Values	Notes
template	Key=uuid	Required, Lookups=name,externalId
owner	Key=uuid	Defaults to API User, Lookups=externalId, email, firstName, lastName
leaders	Key=uuid, or list. Leader is instructor for ILT, Sales Rep for sales demo, proctor for exams	Required for ILT templates, Lookups=externalId, email, firstName, lastName can specify a list
startDate	YYYY-MM-DD	Required
endDate	YYYY-MM-DD	Not required if duration set on template
providerRegion	Key=id	Lookups=name Not required if default set on template
attendees	Integer or List that identifies the non-leader attendees.	Defaults to 0, max 999
timezone	Europe/London	Not required if set on template
name	Varchar(512)	Defaults to template name

description	Text	
externalId	Varchar(32)	
schedule	List of items that specify day, startTime, endTime	Specified days must be inside the start/end window

Here is a curl example. Assume the ILT template

```
curl https://axis-api.readytech.com/v1/event
-H "Content-Type: application/json"
-k -u (login):(password)
-X POST
-d '{
"template": { “name”: "Training 123" },
"startDate": "2018-10-13",
"timezone": "Europe/Paris",
"leaders": { "email": "jsmith@gmail.com" },
"attendees": "10”,
"schedule": [
    { "day": "2018-10-13", "startTime": "09:00", "endTime": "17:00"},
    { "day": "2018-10-14", "startTime": "09:00", "endTime": "17:00"}
] }'
```

Instead of specifying a simple attendee count, users can be assigned using an attendee list.

…

```
"leaders": { "email": "jsmith@gmail.com" },
"attendees": [
    { “firstName”: “Steve”, “lastName”: ”Jones” },
    { “firstName”: “Marc”, “lastName”: ”Jacobs”, “email”: “mjacobs@gmail.com” },
    { “firstName”: “Carol”, “lastName”: ”Andrews”, “email”: “carol@yahoo.com” },
    { “firstName”: “Kevin”, “lastName”: ”Clint”, “externalId”: “123000” }
],
"schedule": [
    { "day": "2018-10-13", "startTime": "09:00", "endTime": "17:00"},
    { "day": "2018-10-14", "startTime": "09:00", "endTime": "17:00"}
] }'
```

No specific field in the attendee list is required but at least one must be specified. Options are firstName, lastName, email, externalId. The externalId is intended to be the LMS ID of the attendee.

#here is an SPP example using all defaults from template

curl https://axis-api.readytech.com/v1/event
-H "Content-Type: application/json"
-k -u (login):(password)
-X POST
-d '{
"template": “xxxx-xxxx-xxxx-xxxxx”,
"startDate": "2018-10-13", “firstName”: “Steve”, “lastName”: “Smith”,
”email”: “xxxx@gmail.com” }’

Implementation Notes

When the api management server receives an event creation request it should process the request in the following way:

- Validate authorization and return 401 if authorization header is missing or invalid
- Validate the URL/method combination return 405 if the method is not supported or does match URL (for example PUT for /events)
- Validate that the request has correct format/syntax (ie json and well formed query params). If not return 400.
- Do validation property level validation. Are required properties provided? Are there unknown properties? Are property values properly formed? For example are dates in YYYY-MM-DD format? If any validation fails return 400. Note the API server should not do data level validation (is the uuid valid or is start date before end date). Axis will do data validation.
- Do lookup processing by making lookup queries to Axis and substituting uuids back into the request. If one or more lookups fails to find 1 matching record return 404.
- The API server should send the request to Axis. If Axis fails to response API should return 500.
- If Axis does not create the event, API should return 400
- If the event is successfully created return the created event in the response along with response code 201

Successful Event Creation

Anytime an event is successfully created using the API will the response code 200 and the created event will be returned in the response data (in the same format it will be return for GET actions. Below is an example:

{

```
"uuid": "xxxxx-xxxx-xxx-xxxx-xxxx",
“externalId”: null,
"templateId": "xxxx-xxxxx-xxxxx-xxxxx-xxxxx",
“type”: “ILT”,
"name": "Introduction to SQL - Boston",
"ownerId": "yyyy-yyyyy-yyyy-yyyyy-yyyyy",
“ownerFirstName”: “Mary”,
“ownerLastName”: “Smith”,f
“ownerEmail”: “msmith@yahoo.com”,
"leader": [ { "id": "zzzzz-zzzz-zzzz-zzzz-zzzz", "firstName": "Steve", “lastName”: “Jones”, "email":
"steve@gmail.com" }, { "id": "wwww-wwww-wwww-wwww", "firstName": "Peter", “lastName”:
“Carson”, "email": "peter@gmail.com" } ],
"timezone": "Europe/Amsterdam",
"schedule": [
   { "day": "2018-10-10", "startTime": "09:00", "endTime": "17:00"},
   { "day": "2018-10-11", "startTime": "09:00", "endTime": "17:00"},
   { "day": "2018-10-12", "startTime": "09:00", "endTime": "17:00"}
],
"description": null
"seatCount": "10",
“seats”: [
   { “seat”: “1”, “accessCode”: “XXXXXXXX01”, “role”: “INSTRUCTOR”, “assigneeFirstName”:
“Steven”, “assigneeLastName”: “Smith”, “assigneeEmail”: “ssmith@gmail.com”, “activatedTs”: null,
“activatedFirstName”: null, “activatedLastName”: null },
   { “seat”: “2”, “accessCode”: “XXXXXXXX02”, “role”: “STUDENT”, “assigneeFirstName”: “Peter”,
“assigneeLastName”: “Jones”, “assigneeEmail”: “pjones@gmail.com”, “activatedTs”: null,
“activatedFirstName”: null, “activatedLastName”: null },
   …
]
}
```


Getting Events

Single Event using UUID

The easiest and most efficient way to retrieve an event is to use the UUID. In that case you use the GET method with the following endpoint:

GET /event/xxxx-xxxx-xxxx-xxxx-xxxx

For example:

```
curl https://axis-api.readytech.com/v1/event/390d9-300ss3-s999s-399s0
    -H "Content-Type: application/json"
```

-k -u (login):(password)
-X GET

If the specified UUID exists (and the api user has the permission to access it) the API will return response code 200 and the event data in the following format:


```
{
    "uuid": "xxxxx-xxxx-xxx-xxxx-xxxx",
    “externalId”: null,
    "templateId": "xxxx-xxxxx-xxxxx-xxxxx-xxxxx",
    “type”: “ILT”,
    "name": "Introduction to SQL - Boston",
    "ownerId": "yyyy-yyyyy-yyyy-yyyyy-yyyyy",
    “ownerFirstName”: “Mary”,
    “ownerLastName”: “Smith”,f
    “ownerEmail”: “msmith@yahoo.com”,
    "leader": [ { "id": "zzzzz-zzzz-zzzz-zzzz-zzzz", "firstName": "Steve", “lastName”: “Jones”, "email": "steve@gmail.com" }, { "id": "wwww-wwww-wwww-wwww", "firstName": "Peter", “lastName”: “Carson”, "email": "peter@gmail.com" } ],
    "timezone": "Europe/Amsterdam",
    "schedule": [
       { "day": "2018-10-10", "startTime": "09:00", "endTime": "17:00"},
       { "day": "2018-10-11", "startTime": "09:00", "endTime": "17:00"},
       { "day": "2018-10-12", "startTime": "09:00", "endTime": "17:00"}
    ],
    "description": null
    "seatCount": "10",
    “seats”: [
       { “seat”: “1”, “accessCode”: “XXXXXXXX01”, “role”: “INSTRUCTOR”, “assigneeFirstName”: “Steven”, “assigneeLastName”: “Smith”, “assigneeEmail”: “ssmith@gmail.com”, “activatedTs”: null, “activatedFirstName”: null, “activatedLastName”: null },
       { “seat”: “2”, “accessCode”: “XXXXXXXX02”, “role”: “STUDENT”, “assigneeFirstName”: “Peter”, “assigneeLastName”: “Jones”, “assigneeEmail”: “pjones@gmail.com”, “activatedTs”: null, “activatedFirstName”: null, “activatedLastName”: null },
       …
    ]
}
```


If the specified event does not exists the API will return response 404.

Getting a list of events

Customers can also return a list of events using the GET method and /events endpoint. For example:

```
curl https://axis-api.readytech.com/v1/events
    -H "Content-Type: application/json"
    -k -u (login):(password)
    -X GET
```

By default, this request will return the current (not expired, not cancelled) events defined in the entity of the API user. There are several optional URL params that can be used to change this default behavior:

scope – current (default), expired, cancelled, all
entities – can specify the uuid of one or more entities to get the events for
subentities=true – includes all sub entities

```
curl https://axis-api.readytech.com/v1/events
    -H "Content-Type: application/json"
    -k -u (login):(password)
    -X GET
    --data-urlencode “scope=all” –data-urlencode “subentities=true”
```

Note that using the –data-urlencode param is preferred over configuring the endpoint directly because it is less likely to result in encoding errors. However, this also works:

```
curl https://axis-api.readytech.com/v1/events?scope=all&subentities=true
```

IMPORTANT: As will all API commands the visibility/permissions of the API user determines what data can be returned. For example if the API user only has access to data in a specific entity he cannot use the API to return data from another entity.

Searching Events

It is also possible to search for events using the POST method. In this case the payload of the request specifies the search parameters. The following params are available.

template
timezone
owner
instructor
startDate (accepted format yyyy-mm-dd)
endDate (accepted format yyyy-mm-dd)
providerRegion

For the date params we should match based on the time zone of the event, not UTC.

The scope, entities, and subentities parameters also available for searches. Without them the search will be limited to current events in the same entity as the API user.

The following example searches for events that have a specific owner, instructor and start on a particular

date. When multiple criteria are specified all must be true for matching records (AND, not OR).

```
curl https://axis-api.readytech.com/v1/events
    -H "Content-Type: application/json"
    -k -u (login):(password)
    -X POST
    -d '{ "ownerId": "xxxx-xxxx-xxxx-xxxxx", "instructorId": "yyyy-yyyy-yyyy-yyyy", "startDate":
"2018-04-22" }'
```

The following example searches for events that ended on a specific date in the past. Note that the scope parameter must be used in this case or 404 will be returned because by default only current events are searched.

```
curl https://axis-api.readytech.com/v1/events
    -H "Content-Type: application/json"
    -k -u (login):(password)
    -X GET
    -d '{ "endDate": "2017-12-15" }'
    --data-urlencode “scope=all”
```

Using operators (this does not need to be in the first release)

By default search parameters use exact matching. For example, endDate=2017-12-15 will only match events that ended on that date (based on the event timezone). However, the following operators are also available. They are used by adding a suffix to the parameter name:

_lt less than
_le less than or equal to
_gt greater than
_ge greater than or equal to

For example, to search for all events that started in the month of May you can use the following:

```
curl https://axis-api.readytech.com/v1/events
    -H "Content-Type: application/json"
    -k -u (login):(password)
    -X GET
    -d '{ "startDate_ge": "2017-05-01", “startDate_le”: “2017-05-31” }'
    --data-urlencode “scope=all”
```

Event Actions

There are several event level actions that can be done via the API. There are:

- Canceling a whole event

- Sending the event notification to the instructors/owner

Actions should be done using the POST action because in some cases additional data will need to be submitted. Also it makes it less likely that actions are done by accident.

#cancel
curl https://axis-api.readytech.com/v1/event/{uuid}/cancel
-H "Content-Type: application/json"
-k -u (login):(password)
-X POST

For notify we just need a simple solution now because we don't have support creating custom templates. So we just need to know which roles the standard notification should be sent to. If they only want to notify specific students they should use the seat action.

#notify
curl https://axis-api.readytech.com/v1/event/{uuid}/notify
-H "Content-Type: application/json"
-k -u (login):(password)
-X POST
-d '{ "owner": "true", "leaders": "true", "attendees": "true" }'

Updating events

PUT /event/{uuid} - updating existing event such as changing schedule

The following params are available for updating via the API.

startDate
endDate
description
externalId
schedule
providerRegion

There are several restrictions on the values of that can be specified for the schedule property. First all the scheduled items must be between 00:00:01 on the startDate and 11:59:59 on the end date. It would not make sense to allow them to schedule sessions outside the window where people can access the portal. Also only future scheduled items can be changed. If a class runs Mon-Friday and they try to change the schedule on Wed they can only change the schedule for Thurs/Friday and the end time for Wednesday session if it has not ended yet.

POST /event/{uuid}/{action} - perform bulk action on event, like Cancel or Notify or start now

Seat endpoints

Customers need to be able to fully manage their event seats using the API.

GET /event/{uuid}/seats - get the seats for an event

POST /event/{uuid}/seat - add seat to event

PUT /seat/{uuid} - update a seat

DELETE /seat/{uuid} - delete seat

POST /seat/{uuid}/{action} - perform some action on the seat such as Cancel or Notify