Praesidium APIs

This book contains the official documentation for all APIs which Praesidium makes available to its clients

- Event Publishing Specification
- User Management REST API

Event Publishing Specification

This document provides the build specification to clients who wish to receive learning event notifications programmatically.

Changelog

2023-03-15	Added several refX fields to event_specific_detail
2023-01-20	Clarified meaning of UUID in eventContext Corrected typo in learning_path_completed sample payload
2022-12-20	Added LEARNING_PATH_COMPLETED event type
2022-12-13	Corrected payload property names from camel case to snake case
2022-01-12	Added first_name and last_name fields to COURSE_COMPLETED payload in event_specific_detail to aid with the resolution of matching errors

Overview

In general, a client will reach out to Praesidium to identify relevant events and an endpoint to which to publish them in near real-time. Authentication credentials (to be used with HTTP Basic Authentication) may be provided securely to Praesidium as well. Praesidium will then set up the event subscription for the client in question. From the time of setup onward, any relevant events that happen will be communicated in near real-time to the endpoint specified by the client.

Event Payload

The event payload will take the form of a JSON Object as follows:

```
{
"version": <STRING>,
    "event_type": <STRING>,
"event_timestamp": <STRING>,
"event_context": <JSON>,
"event_specific_detail": <JSON>
```

```
}
```

where
version - the version string (value: "1.0")
event_type - what kind of event is being reported. Event type will determine the structure of event_context (see below)
event_timestamp - a UTC timestamp in ISO format (i.e. YYYY-MM-DD HH24:MM:SS)
event_context - a JSON object with a specific structure depending upon event_type
event_specific_detail - a JSON object with additional contextual information that may differ depending on event_type

Accepted Return Values

In keeping with general RESTful principles, HTTP Status codes should be used. A return value of 400 should be used if the event payload violates the specification above or is otherwise malformed. A return value of 200, 201 or 202 should be returned to indicate that the event record was successfully received.

Event Types

COURSE_COMPLETED

The event context structure will consist of 2 fields inside the JSON object

1. uuid - This field will contain the UUID of the user as returned by user creation via the REST API
2. user – This field will contain the email address needed to identify the user who completed the course
3. course – This field will contain a JSON object with 2 fields:
    1. id: the course sku for the course completed
    2. name: the course name for the course completed

The event_specific_detail field now contains an object user_detail with the following fields:

- first_name
- last_name
- clientExternalId - this field will hold the value from the Academy user record that the client has provided uniquely identifying this user in their system. (Note: this field may be null if the client is not using this feature.
- ref3 - this corresponds to the ref3 element in the Academy user record
- ref4 - this corresponds to the ref4 element in the Academy user record
- ref5 - this corresponds to the ref5 element in the Academy user record
- ref7 - this corresponds to the ref7 element in the Academy user record
- ref8 - this corresponds to the ref8 element in the Academy user record
- ref9 - this corresponds to the ref9 element in the Academy user record

Example Payload


```
{
    "version": "1.0",
    "event_type": "COURSE_COMPLETED",
    "event_timestamp": "2018-03-01 17:45:37",
    "event_context": {
        "uuid": "aaaaaaaa-bbbb-cccc-dddd-ffffffffffff",
        "user": "email@gmail.com",
        "course": {
            "id": "CON20938ES",
            "name": "Duty to Report: Mandated Reporter"
        }
    },
    "event_specific_detail": {
        "user_detail": {
            "first_name": "Tester",
            "last_name": "Testerman",
            "clientExternalId": "1234569",
            "ref3": "arbitrary text",
            "ref4": "arbitrary text2",
            "ref5": "arbitrary text3",
            "ref7": "arbitrary text4",
            "ref8": "arbitrary text5",
            "ref9": "arbitrary text6",
        }
    }
}
```


LEARNING_PATH_COMPLETED

The event context structure will consist of 2 fields inside the JSON object

1. uuid - This field will contain the UUID of the user as returned by user creation via the REST API
2. user – This field will contain the email address needed to identify the user who completed the course
3. learning_path – This field will contain a JSON object with 2 fields:
    1. id: the learning path sku for the learning path completed
    2. name: the learning path name for the learning path completed

The event_specific_detail field now contains an object user_detail with the following fields:

- first_name
- last_name
- clientExternalId - this field will hold the value from the Academy user record that the client has provided uniquely identifying this user in their system. (Note: this field may be null if the client is not using this feature.
- ref3 - this corresponds to the ref3 element in the Academy user record
- ref4 - this corresponds to the ref4 element in the Academy user record
- ref5 - this corresponds to the ref5 element in the Academy user record
- ref7 - this corresponds to the ref7 element in the Academy user record
- ref8 - this corresponds to the ref8 element in the Academy user record
- ref9 - this corresponds to the ref9 element in the Academy user record

Example Payload

```
{
"version": "1.0",
    "event_type": "LEARNING_PATH_COMPLETED",
"event_timestamp": "2018-03-01 17:45:37",
"event_context": {
        "uuid": "aaaaaaaa-bbbb-cccc-dddd-ffffffffffff",
        "user": "email@gmail.com",
        "learning_path": {
            "id": "CONLP10023EN",
            "name": "Duty to Report: Mandated Reporter"
        }
    },
"event_specific_detail": {
        "user_detail": {
            "first_name": "Tester",
            "last_name": "Testerman",
            "clientExternalId": "1234569",
            "ref3": "arbitrary text",
            "ref4": "arbitrary text2",
            "ref5": "arbitrary text3",
            "ref7": "arbitrary text4",
            "ref8": "arbitrary text5",
            "ref9": "arbitrary text6",

        }
    }
```

```
}
```

Testing

COURSE_COMPLETED Test Endpoint

This endpoint will allow a client to manually trigger the mechanism which sends course completion events back to client systems. The client can specify an endpoint to which to send the test event payload. The client will send a payload to the Praesidium endpoint and a success response will be sent to the endpoint that the client specified.

Endpoint: https://test.praesidiumacademy.com/portal/event_pub_webhooks/client_course_action

Method: POST

Sample Payload (note the "T" embedded in the "date" field format)

```
{
    "client_id": "35F5C5A985D111EB857A0A3ECA36592D",
    "user_guid": "584adf35-85d1-11eb-857a-0a3eca36592d",
    "email": "lcarl@notreallythere.com",
    "location_guid": "5f383262-85d1-11eb-857a-0a3eca36592d",
    "courseSku": "TCCE1001",
    "date": "2021-03-11T12:01:03",
    "url": "https://test.client.eventpublication/endpoint"
}
```

> Note that the url parameter is optional. When present, the current client event publishing endpoint will be changed to the url value in the payload and will be the endpoint used until it is changed again.

Sample Response (HTTP 200)

```
{
    "message": "success"
}
```

User Management REST API

This document details the RESTful API methods available to manage users for Praesidium clients.

Changelog

2025-11-04	• Added new endpoint for determining if a user with a given email exists
2024-08-18	• Added note about ignoring duplicate requests

2023-10-30	• Addition of content removal capabilities to the API
	  ◦ New Endpoint for removal of content from User
	• Addition of content re-enrollment capabilities to the API
	  ◦ New Endpoint for re-enrollment content in User
	• Updated the "Create a new user" notes to explain changes to the functionality of the HTTP 400 errors that may occur.
	• For "Create a new user", added a note about Academy requiring unique emails

2023-01-18	• Added examples with and without content specification • Updated general object structure to specify some fields as optional/ignored • Noted lack of content return from GET • Noted which UUIDs will be provided by Praesidium and the general format in which they will be provided

2022-10-18	• Added content enrollment to API
	  ◦ User Creation & Update impacted
	  ◦ New endpoint for enrollment of User in content
	  ◦ added content into structure of User object
	• Clarification of JWT expiration timeframe

2021-05-18	• Added instructions on how to use refresh token • changed field name "program-type" in user object to "program_type" for consistency

2021-02-26	• Added structure of User Object • Endpoint URLs updated • Sample requests updated to match structure and to use properly formatted UUIDs • Clarified return value of creation endpoint • Fixed typo in authentication success response • Role details and descriptions added

Overview

Authorization will be handled by Oauth2 using JWT. There are methods provided for only one resource: Users. All URLs will be specified without hostname, as only the hostname need change to identify the test environment versus production.

Methods are available to perform the following operations:

1. Retrieve a User Record (GET)
2. Create a new User Record (POST)
3. Update a User Record (PUT)
4. Deactivate a user record (PUT) - remove system access
5. Add content to a user record (PUT)
6. Determine a if a user with a given email exists in Praesidium Academy

> **Unless otherwise specified, all services require the JWT token to be passed in the Authorization: Bearer header**

> **In addition to all other potential error messages listed below, any endpoint may return HTTP 429 Too Many Requests. If this code is returned, a Retry-After header will be attached to the message to indicate how long to wait before retrying the request.**

> **Any duplicate requests to change data (e.g. Create, Update, Deactivate, Enroll) within 30 seconds will be ignored.**

Authentication

> JWT expiration is 900 seconds (15 minutes)

URL	/portal/authenticate/login_api
Method	POST

Data Format	<pre>{
	"grant_type": "client_credentials",
	"client_id": "<Praesidium provided client ID>",
	"client_secret": "<Praesidium provided client secret>",
	"scope": ["<scope1>", "<scope2>", ..., "<scopeN>"]
	}</pre>
	The only valid scope value is: "prae.client.api.user"

Success Response	**HTTP Status Code: 200**
	<pre>{
	"access_token": "<JWT>",
	"token_type": "bearer",
	"expires_in": 900,
	"refresh_token": "<JWT>"
	}</pre>
	The Refresh token can be used to generate a new access token (if it has not expired) by passing it in the Authorization: Bearer header and using an HTTP GET to retrieve /portal/session/refresh_jwt

Failure Response	Authentication Error
	HTTP Status Code: 401 Authorization Required
	`{ "error": "invalid_client", "error_description": "[string]", "expires_uri": "[reserved for future use]" }`
	
	Input Validation Error
	HTTP Status Code: 400 Bad Request
	`{ "error": "invalid_request", "error_description": "[string]", "expires_uri": "[reserved for future use]" }`
	
	Invalid Scope Error
	HTTP Status Code: 400 Bad Request
	`{ "error": "invalid_client", "error_description": "[string]", "expires_uri": "[reserved for future use]" }`

Example	`// JQuery`
	`$.ajax({`
	`  url: "/portal/authenticate/login_api",`
	`  dataType: "json",`
	`  data: {`
	`  "grant_type": "client_credentials",`
	`  "client_id": "xyz_client",`
	`  "client_secret": "itsasecret",`
	`  "scope": [`
	`  "prae.client.api.user"`
	`  ]`
	`  },`
	`  type: "POST",`
	`  success: function( r ) {`
	`  console.log( r );`
	`  }`
	`});`

Resource: User

Structure of the User Object

All data fields in the user object will fall into one of five categories:

1. Principal – this category is for personal information that is primarily used to identify the user
2. System – this category is for system information, such as roles, status (active or inactive)
3. Person – this category is for other personal information that is not used to identify the user
4. Context – this category is for information describing the contexts that apply to the user
5. Attributes – this category is for information describing the user that can be used to determine which course assignments are applied, and for reporting purposes

So, in general a user object (for the purpose of this API is defined by the following format:

```
{
  "principal": {
```

```
      "UUID": "<string>",
      "first_name": "<string>",
      "last_name": "<string>",
      "email": "<email>",
      "client_external_id": "<string>"
    },
  "system": {
    "status": "<(active | inactive)>",
    "role": "<role-value>"
  },
  "person": {
  },
  "context": {
    "client": "<UUID>",
    "locations": [
      "<UUID1>",
      "<UUID2>",
      ...,
      "<UUIDN>"
    ],
    "content": [
      {
        "uuid": "<UUID>",
        "type": "<content_type-value>"
      },
      ...
    ]
  },
  "attributes": {
      "program_type": "<program_type-value>",
      "position": "<position-value>"
  }
}
```

Currently, the only valid content_type-values are "course" and "learning path"

The only valid role-values are:

- **Learner:** The user in question can only take courses, and has no administrative access
- **Administrator**: The user in question can edit user info for non-administrative user

- **Administrator - View Only**: The user in question has administrative access, but cannot edit any information

General Notes about the fields included in the user object relevant to individual REST operations

> When using a POST to create a user, the UUID field at the top level is optional.

> The "content" object under the "context" object is optional when passing a user object to POST for creation or PUT for general update

> In addition, the "content" object will never be specified in a return value from an API call for performance reasons

> The UUIDs under "client", "locations" and "content" will be provided by Praesidium (csv file will contain UUID, name for client and locations, and will additionally contain content_type and SKU for courses/learning paths)

Retrieve a User Record

URL	/portal/lms_api/v2/user/{UUID}
Method	GET
Path Parameters	Required **UUID** - UUID for the desired user
Success Response	**HTTP Status Code: 200** User object (structure as described above)

Failure Response	Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example	<pre>// JQuery
	$.ajax({
	 url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce2287baaa",
	 type: "GET",
	 success: function(r) {
	 console.log(r);
	 }
	});</pre>

Create a New User

Upon creation, the user will be automatically enrolled in any content included in the payload

Email addresses must be unique across all Academy.

If an attempt is made to create a user with an email that already exists in Academy, the technicalMessage property of the HTTP 400 error will now contain the id of the user with the existing email IF that user falls under the requesting client's hierarchy.

URL	/portal/lms_api/v2/user
Method	POST
Data Parameters	Required User object (structured as above). **Any value in the UUID field will be ignored**.
Success Response	**HTTP Status Code: 201** User object (structure as described above). **The UUID field will contain the UUID of the newly created user**

Failure Response	Input Validation Error
	HTTP Status Code: 400 Bad Request
	`{`
	`"status": [integer],`
	`"code": "[string]",`
	`"message": "[string]",`
	`"technicalMessage": "[string]",`
	`"infoURI": "[reserved for future use]"`
	`}`
	
	Internal Server Error
	HTTP Status Code: 500 Internal Server Error
	`{`
	`"status": [integer],`
	`"code": "[string]",`
	`"message": "[string]",`
	`"technicalMessage": "[string]",`
	`"infoURI": "[reserved for future use]"`
	`}`
	
	status - integer HTTP status
	code - internal error code (if available, otherwise HTTP Status)
	message - Human readable error messages, separated by new lines
	technicalMessage - Human readable error message, separated by new lines, containing technical detail

Example

```
// JQuery (without content)
$.ajax({
 url: "/portal/lms_api/v2/user",
 type: "POST",
 dataType: "json",
 data: {
   "principal": {
     "first_name": "John",
     "last_name": "Smith",
     "email": "jsmith@example.com",
     "client_external_id": "123456958"
   },
   "system": {
     "status": "active",
     "role": "Administrator - View Only"
   },
   "person": {},
   "context": {
     "client": "aaaaaaaa-bbbb-cccc-dddd-
5ce2287baaa",
     "locations": [
       "aaaaaaaa-bbbb-cccc-dddd-
5ce1207baaa"
     ]
   },
   "attributes": {
     "position": "director (camp)",
     "program_type": "aquatics"
   }
 },
 success: function( r ) {
   console.log( r );
 }
});
// JQuery (with content)
$.ajax({
url: "/portal/lms_api/v2/user",
type: "POST",
dataType: "json",
data: {
"principal": {
"first_name": "John",
"last_name": "Smith",
```

Update an Existing User

Note: upon completion of other updates, the user will be automatically enrolled in any content included in the payload

URL	/portal/lms_api/v2/user/{UUID}
Method	PUT
Path Parameters	Required **UUID** UUID for the user to be updated
Data Parameters	Required User object (structured as above).
Success Response	**HTTP Status Code: 200** User object (structure as described above).

Failure Response	Authorization Error **HTTP Status Code:** 403 `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example

```
// JQuery (without content)
$.ajax({
 url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce9942baaa",
 type: "PUT",
 dataType: "json",
 data: {
   "principal": {
     "first_name": "John",
     "last_name": "Smith",
     "email": "jsmith@example.com",
     "client_external_id": "123456958"
   },
   "system": {
     "status": "active",
     "role": "Administrator - View Only"
   },
   "person": {},
   "context": {
     "client": "aaaaaaaa-bbbb-cccc-dddd-5ce2287baaa",
     "locations": [
       "aaaaaaaa-bbbb-cccc-dddd-5ce1207baaa"
     ]
   },
   "attributes": {
     "position": "director (camp)",
     "program_type": "aquatics"
   }
 },
 success: function( r ) {
   console.log( r );
 }
});
// JQuery (with content)
$.ajax({
url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce9942baaa",
type: "PUT",
dataType: "json",
data: {
"principal": {
```

Deactivate an Existing User (Remove system access)

URL	/portal/lms_api/v2/user/{UUID}/deactivate
Method	PUT
Path Parameters	Required **UUID** - UUID for the desired user
Success Response	**HTTP Status Code: 200** User object (structure as described above)

Failure Response	Authorization Error **HTTP Status Code:** 403 `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example	`// JQuery`
	`$.ajax({`
	`url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce2287baaa/deactivate",`
	`type: "PUT",`
	`success: function( r ) {`
	`console.log( r );`
	`}`
	`});`

Add Content to an Existing User

URL	/portal/lms_api/v2/user/{UUID}/enrollContent
Method	PUT
Path Parameters	Required **UUID** UUID for the user to be updated

Data Parameters	Required
	Content Listing (structured as below).
	
	`{`
	`"content": [`
	`{`
	`"uuid": "[string]",`
	`"type": "[string]"`
	`},`
	`...,`
	`]`
	`}`
	
	uuid - course UUID (Praesidium provided)
	type - only the values "course" and "learning path" are available

Success Response	**HTTP Status Code: 200** User object (structure as described above).

Failure Response	Authorization Error **HTTP Status Code:** 403 `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example	<pre>// JQuery
	$.ajax({
	  url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce9942baaa/enrollContent",
	  type: "PUT",
	  dataType: "json",
	  data: {
	    "content": [
	      {
	        "uuid": "aaaaaaaa-bbbb-cccc-dddd-5ce1248baaa",
	        "type": "course"
	      }
	    ]
	  },
	  success: function(r) {
	    console.log(r);
	  }
	});</pre>

Remove Content from an Existing User

WARNING: Removing a Learning Path from a user will **NOT** remove the corresponding courses

URL	/portal/lms_api/v2/user/{UUID}/removeContent
Method	PUT
Path Parameters	Required **UUID** UUID for the user to be updated

Data Parameters	Required
	Content Listing (structured as below).
	
	`{`
	`"content": [`
	`{`
	`"uuid": "[string]",`
	`"type": "[string]"`
	`},`
	`...,`
	`]`
	`}`
	
	uuid - course UUID (Praesidium provided)
	type - only the values "course" and "learning path" are available

Success Response	**HTTP Status Code: 200** User object (structure as described above).

Failure Response	Authorization Error **HTTP Status Code:** 403 `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example	`// JQuery`
	`$.ajax({`
	`url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce9942baaa/removeContent",`
	`type: "PUT",`
	`dataType: "json",`
	`data: {`
	`"content": [`
	`{`
	`"uuid": "aaaaaaaa-bbbb-cccc-dddd-5ce1248baaa",`
	`"type": "course"`
	`}`
	`]`
	`},`
	`success: function( r ) {`
	`console.log( r );`
	`}`
	`});`

Re-Enroll Content in an Existing User

Note: User's content that is re-enrolled will be set to a not-started status in Academy. Any prior completion certificates will be unchanged.

WARNING: Re-Enrolling a user in a Learning Path will **NOT** re-enroll that user in the corresponding courses

URL	/portal/lms_api/v2/user/{UUID}/reEnrollContent
Method	PUT
Path Parameters	Required **UUID** UUID for the user to be updated

Data Parameters	Required
	Content Listing (structured as below).
	
	`{`
	`"content": [`
	`{`
	`"uuid": "[string]",`
	`"type": "[string]"`
	`},`
	`...,`
	`]`
	`}`
	
	uuid - course UUID (Praesidium provided)
	type - only the values "course" and "learning path" are available

Success Response	**HTTP Status Code: 200** User object (structure as described above).

Failure Response	Authorization Error **HTTP Status Code:** 403 `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Input Validation Error **HTTP Status Code:** 400 Bad Request `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Not Found Error **HTTP Status Code:** 404 Not Found `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` Internal Server Error **HTTP Status Code:** 500 Internal Server Error `{ "status": [integer], "code": "[string]", "message": "[string]", "technicalMessage": "[string]", "infoURI": "[reserved for future use]" }` **status** - integer HTTP status **code** - internal error code (if available, otherwise HTTP Status) **message** - Human readable error messages, separated by new lines **technicalMessage** - Human readable error message, separated by new lines, containing technical detail

Example	`// JQuery`
	`$.ajax({`
	`url: "/portal/lms_api/v2/user/aaaaaaaa-bbbb-cccc-dddd-5ce9942baaa/reEnrollContent",`
	`type: "PUT",`
	`dataType: "json",`
	`data: {`
	`"content": [`
	`{`
	`"uuid": "aaaaaaaa-bbbb-cccc-dddd-5ce1248baaa",`
	`"type": "course"`
	`}`
	`]`
	`},`
	`success: function( r ) {`
	`console.log( r );`
	`}`
	`});`

User Exists

URL	/portal/lms_api/v2/user/{email}/exists
Method	GET
Path Parameters	Required **email** email to be checked against valid user emails for Academy system

Success Response	When a user exists with that email in a sublicense that you have access to:
	
	HTTP Status Code: 200
	User object (structure as described above).
	
	--- 	
	
	When a user exists with that email, but you do not have access to the sublicense(s) associated with that user:
	
	HTTP Status Code: 200
	
	`{`
	`"status": 200,`
	`"code": "HTTP/1.1 200 OK",`
	`"message": "user with given email found",`
	`"technicalMessage": "",`
	`"infoURI": ""`
	`}`

Failure Response	User Not Found
	HTTP Status Code: 404 Not Found
	
	`{`
	`"status": 404,`
	`"code": "HTTP/1.1 404 Not Found",`
	`"message": "404 Not Found",`
	`"technicalMessage": "User not found",`
	`"infoURI": ""`
	`}`

Example	// JQuery
	$.ajax({
	url:
	"/portal/lms_api/v2/user/notreallyanemail@example.com/exists",
	type: "GET",
	success: function(r) {
	console.log(r);
	}
	});