AKADÉMOS

# School Integration Specification

Version 2.0

Last update: March 2025

# Table of Contents

# 01 Platform Introduction

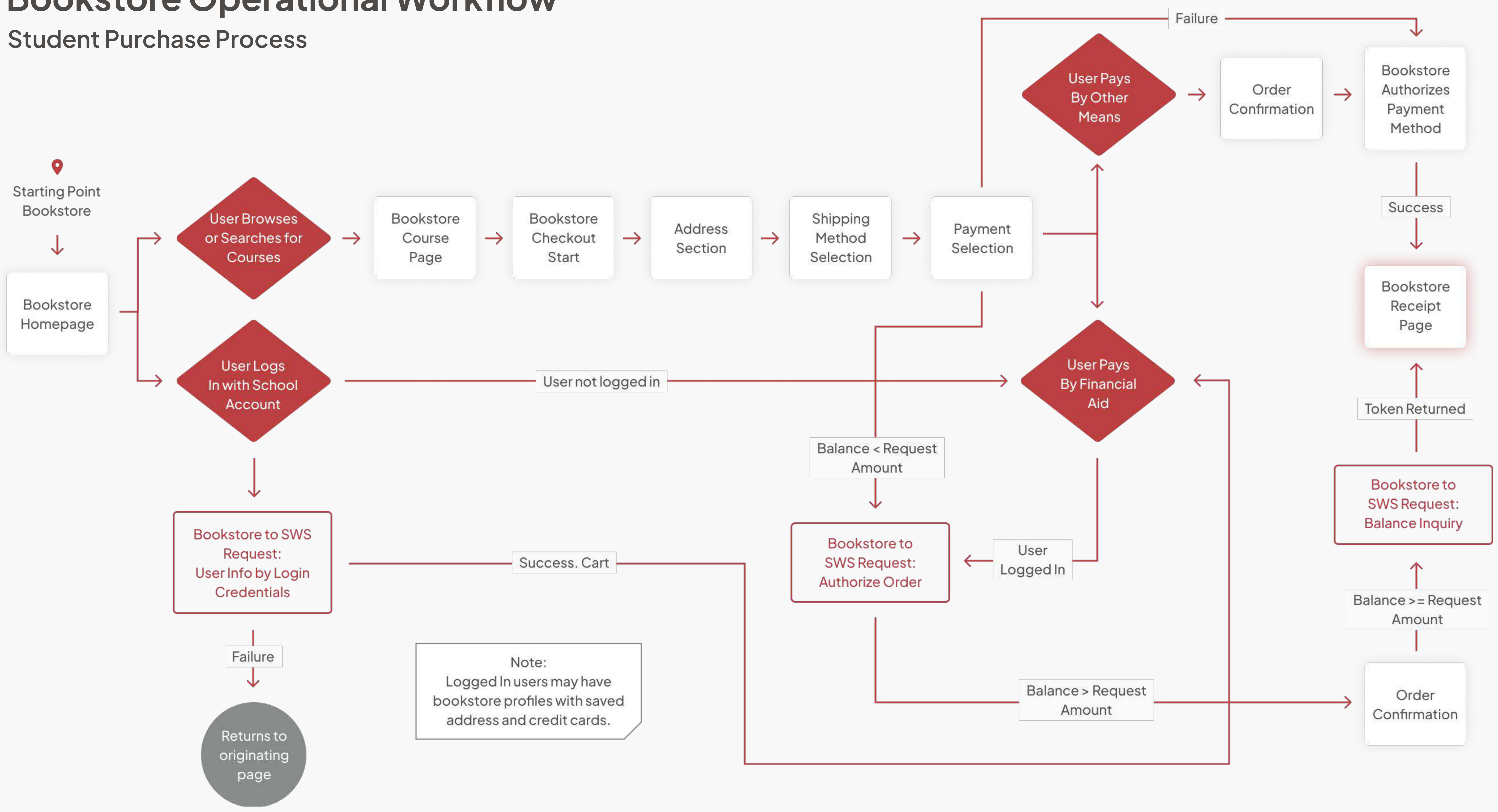
Welcome to the Akademos Integrated Bookstore! Integrating your school systems with the Akademos platform brings many benefits over the standard installation, including:

- Automated data synchronization requiring less actions from the administrators
- Expedited and more accurate student shopping experience
- Support for custom payment methods
- Single Sign-On account management for students, administrators and faculty

The Akademos platform is designed to be as flexible as possible: depending on your need, components can be interchanged and left off of a final build, and the system will scale smoothly.
This flexibility is also found in data methods and formats. In the following document, we will describe the interactions between the Akademos API systems and your School Web Service (SWS) required to create the optimum integrated platform.

# Bookstore Operational Workflow

## Student Purchase Process


Starting Point Bookstore
Bookstore Homepage
User Browses or Searches for Courses
Bookstore Course Page
Bookstore Checkout Start
Address Section
Shipping Method Selection
Payment Selection
User Pays By Other Means
Order Confirmation
Bookstore Authorizes Payment Method
Failure
Success
Bookstore Receipt Page
Token Returned
Bookstore to SWS Request: Balance Inquiry
Balance >= Request Amount
Order Confirmation
User Logs In with School Account
User not logged in
User Pays By Financial Aid
Balance < Request Amount
Bookstore to SWS Request: Authorize Order
User Logged In
Balance > Request Amount
Bookstore to SWS Request: User Info by Login Credentials
Success. Cart
Failure
Returns to originating page

Note:
Logged In users may have bookstore profiles with saved address and credit cards.

# 02 Single Sign-On (SSO)

SSO provides the ability for Akademos to correctly authorize users and direct them to the corresponding location on the bookstore website through use of credentials they already make use of.
Depending on your platform capabilities, the bookstore can authenticate users who have already signed into the school SIS system without asking for additional credentialing.

Akademos preferred protocol is SAML 2.0 but we are capable of working with Shibboleth. Any IdP that can use one of these methods can be connected to our service.

## a Authentication

Common attributes used for authentication include;

- Student ID
- Username
- Email

To set up the SSO the attribute selected to authenticate users must be the same value included in the user information request (API) or in the user file (Flat File) as the ID (see field definitions in section 3). During the integration Akademos will request test credentials to ensure the SSO is set up correclty.

## b Direct Links

To take students to a specific course and be able to look for all the adopted materials made, you can place the following url:

- [domain].textbookx.com/classes/{course_number}-{term_code}

See section 3, Course structure request and course file field definitions: course_number and term_code.

To trigger the login and allow students and faculty to access the bookstore without credentials, you can place the following ulr:

- https://[domain].textbookx.com/account/login.php?ticket=institute_[domain]

# 03 Student Information System (SIS) methods and formats

Akademos is capable of working with a number of data formats and transfer methods, including both real-time and flat-file. Note that not all system components are capable of flat-file background processing.

Akademos will require three sets of information: user, course and term data. Please see the description of each field and make sure these are label exactly as we present them here.

## User information: Field Definitions

| Property Name | Value | Character limit | Description |
|---|---|---|---|
| status_code | integer | - | Represents status of request. 0 indicates success. (API) |
| status_message | string | - | Represents specific information regarding failure. (API) |
| id | string | varchar (50) | The user ID for students, professors or admins. |
| role | string | enum ('student','professor') | The user type, student, professor, or readonly. |
| first_name | string | varchar (150) | The user's first name. |
| last_name | string | varchar (150) | The user's last name. |
| email | string | varchar (150) | The user's registered email. |
| phone_number | string | varchar (15) | The user's phone number. Only for students. |
| address_line_1 | string | varchar (255) | The user's street address. Only for students. |
| address_line_2 | string | varchar (255) | The user's street address. Only for students. |

Table continues on next page.

| Property Name | Value | Character limit | Description |
|---|---|---|---|
| city | string | varchar (100) | The user's city. Only for students. |
| state | string | varchar (100) | The user's state (valid U.S. state required, otherwise leave blank). Only for students. |
| postal_code | string | varchar (30) | The user's postal code. Only for students. |
| student_grade_level | string | enum('freshman','sophomore','junior','senior','graduate','post_graduate','unclassified') | The student grade level that represents a common description of a student's current progress within a certificate or degree program.<br>Specific institutes may differ, but the following guide may be used:<br>• Freshman: Student is in the first year of a program<br>• Sophomore: Student is in the second year of a program<br>• Junior: Student is in the third year of a program<br>• Senior: Student is in the fourth year of a program<br>• Graduate: Student is a fifth or beyond year of a graduate level program |
| student_major | string | varchar (20) | The user's current school major studies. |
| course_number | string | varchar (100) | In the case of students: all course_numbers where the student is registered. Leave empty if the user is not yet registered.<br>In the case of instructors: all course_numbers where the professor is assigned to teach. |
| term_code | string | varchar (20) | Numeric term code. When joined to course_number creates CRN. |
| term_desc | string | varchar (150) | Term description (e.g. "Fall 2014", "Summer 2015", etc.) |
| credit_amount | integer | decimal (7,2) | The dollar credit amount that is issued to user to be used in the bookstore. |
| credit_exp_date | date | yyyy-mm-dd or mm/dd/yyyy | Selected date of when credit amount is expired. Credit value changes to '0' value. |
| credit_notify | string | enum('yes','no') | If value is 'Yes' an email notification is sent to user including their credit amount and expiration date. |
| credit_start_date | date | yyyy-mm-dd or mm/dd/yyyy | This field can be blank or you can indicate the exact date you want the credits to be issued. |
| username | string | varchar (50) | This field can be used to get a parallel identifier of student and professor accounts. This field can be used to authenticate users through LMS or send information back to school. |
| special_type | string | varchar (50) | Designate students in a particular program (e.g. EA). Required for Equitable Access clients. |
| student_type | string | varchar (255) | Identifies student cohorts, such as Dual Enrollment. |

## Syllabus course data: Field definitions

| Property Name | Value | Character limit | Description |
|---|---|---|---|
| course_credits | integer | int(3) | Numbers from 1 to 255 |
| course_description | string | varchar(3000) | Text or HTML field up to 3,000 character limit |
| course_start_time | time | HHMMSS | 12 or 24-hour format |
| course_end_time | time | HHMMSS | 12 or 24-hour format |
| course_location | integer/string | char(64) | Alphanumeric (letters and/or numbers) |
| course_schedule | string | - | Allowed values: Mon, Tue, Wed, Thu, Fri, Sat, Sun |
| course_type | string | varchar(255) | Allowed values: Traditional, Blended, Online |
| course_method | string | vachar(255) | Allowed values: ONL, LEC, LAB, CLA, REC |
| lab_location | integer/string | char(64) | Alphanumeric (letters and/or numbers) |
| lab_start_time | time | HHMMSS | 12 or 24-hour format |
| lab_end_time | time | HHMMSS | 12 or 24-hour format |
| prerequisite | integer/string | - | Alphanumeric (letters and/or numbers), comma separated values allowed |
| corequisite | integer/string | - | Alphanumeric (letters and/or numbers), comma separated values allowed. |

## Term information: Field Definitions

| Property Name | Value | Character limit | Description |
|---|---|---|---|
| status_code | integer | - | Represents status of request. 0 indicates success |
| status_message | string | - | Represents specific information regarding failure. |
| term_code | string | varchar (20) | Unique term code |
| start_date | date | yyyy-mm-dd or mm/dd/yyyy | Start date of term. Format: Y-m-d |
| end_date | date | yyyy-mm-dd or mm/dd/yyyy | End date of term. Format: Y-m-d |

## Course information: Field Definitions

| Property Name | Value | Character limit | Description |
|---|---|---|---|
| status_code | integer | - | Represents status of request. 0 indicates success (API). |
| status_message | string | - | Represents specific information regarding failure (API). |
| course_number | string | varchar (100) | Unique per term. When joined to term_code creates CRN. |
| course_title | string | varchar (100) | Title of course (e.g. "Introduction to Accounting"). |
| course_name | string | varchar (60) | Abbreviated name of course (e.g. "ACCT"). |
| course_code | string | varchar (60) | Course code. (e.g. "200" as in ACCT 200). |
| course_section | string | varchar (60) | Course section (e.g. "1A" as in ACCT 200 1A). |
| course_credits | integer | int (3) | Course credits (e.g. 3) Required for Equitable Access clients. |
| course_model | string | varchar (60) | Designate courses in a particular program (e.g. EA). Required for Equitable Access clients. |
| department_code | string | varchar (20) | Department code (not visible to students). |
| department_desc | string | varchar (150) | Department description (e.g. "English", "Political Science"). |
| campus_code | string | varchar (20) | Section campus code (not visible to students). |
| campus_desc | string | varchar (150) | Section campus description (e.g. "Lansing", "Central Campus", etc. --- exclude campus_code and desc if only one campus. |
| term_code | string | varchar (20) | Numeric term code. When joined to course_number creates CRN. |
| term_desc | string | varchar (150) | Term description (e.g. "Fall 2023", "Summer 2023", etc.) |
| session_code | string | varchar (64) | Session code. |
| start_date | date | yyyy-mm-dd or mm/dd/yyyy | Date the course begins (YYYYMMDD). |
| end_date | date | yyyy-mm-dd or mm/dd/yyyy | Date the course ends (YYYYMMDD). |
| enrollment_cap | string | int (4) | Maximum number of students who may be enrolled in the course. |

## a API integration

### I Summary

**Rest**
The Akademos REST client supports multiple resource formats and schemes of data transfer between Akademos and the SWS API server.

**Resource format:**
- JSON (application/json; charset=UTF-8)
- XML (application/xml; charset=UTF-8)

**Data transfer schemes:**
- Resource as RAW body (Akademos POST requests or SWS response).
  In this case Request Authentication Code signature_method and signature_code should be passed as HTTP headers. Resources should be passed in body as RAW JSON/XML data without any encoding.
- Same as the previous item, but Akademos REST client could send signature_method and signature_code in URI instead of headers.

### II Authentication

The Akademos REST client supports the following authentication method:
- HTTPS + Static authentication key in HTTP header

- **Rest**

The Akademos REST client supports HMAC-SHA256 or HMAC-SHA1 for signature code generation.

**Getting a signing key**
Akademos software development team will provide a signing key for HMAC.

**Required data**
When using this feature, additional data should be present in every SWS API response and Akademos REST client request. Depending of data transfer schemes it could be HTTP header or URI parameter.

| | |
|---|---|
| signature_method | HMAC-SHA256 |
| signature_code | 6829e274cb147760dc5db4b5c0248699d939c7b8 |

**Creating the signature base string**
The REST URI and raw request/response body must be joined to make a single string, from which the signature will be generated.
- Convert the HTTP Method to uppercase and set the output string equal to this value.
- Append the '&' character to the output string.
- Append REST method URI to the output string.
- Append the '&' character to the output string. Skip this step if resource body is empty.

- Append response resource body string and append it to the output string. Skip this step if resource body is empty.

### Example of SWS balance response signature base string:

```
GET&/12345/balance&<?xml version="1.0" encoding="UTF-8" ?><response>
<balance>{number}</balance>
<status_code>{integer}</status_code>
<status_message>{string}</status_message>
</response>
```

## III API Components

### Terms Request

Contains data regarding active or upcoming terms.
All fields are required.

#### HTTP Request

```
GET /terms
```

#### Request Body

Do not supply a request body with this method.

#### Resource Representation (JSON)

```
"status_code": {integer},
"status_message": {string},
"terms": [
  {
   "term_code": {string},
   "start_date": {date},
   "end_date": {date}
  }
]
```

#### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
  <terms>
    <term>
      <term_code>{string}</term_code>
      <start_date>{date}</start_date>
      <end_date>{date}</end_date>
    </term>
  </terms>
</response>
```

### Course Structure Request

Akademos is capable of working with a number of data formats and transfer methods, including both real-time and flat-file. Note that not all system

components are capable of flat-file background processing.
**All fields are required.**

### Resource Representation (JSON)

```
“status_code”: {integer},
“status_message”:{string},
“courses”: [
 {
 "course_number": {string},
 “course_title”: {string},
 "course_name": {string},
 "course_code": {string},
 "course_section": {string},
 "course_credits": {integer},
  "course_model": {string} ,
 "department_code": {string},
 “department_desc”: {string},
 “campus_code”: {string},
 “campus_desc”: {string},
 "term_code": {string},
 “term_desc”: {string},
 "session_code": {string},
 "start_date": {date},
 "end_date": {date},
 "enrollment_cap": {integer}
 }
]
```

### HTTP Request

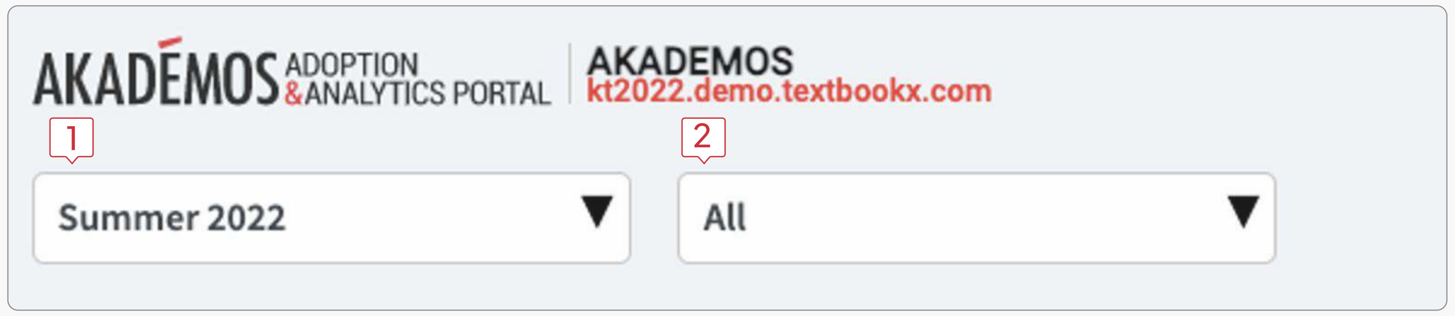
```
GET /structure
```

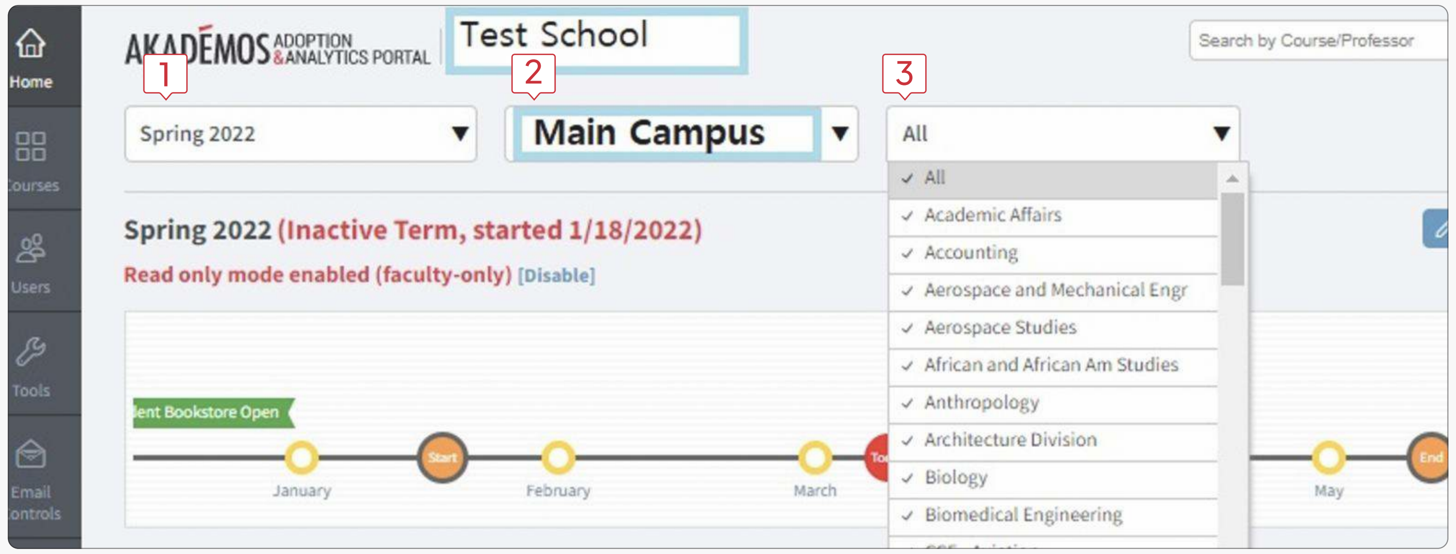
### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
  <courses>
    <course>
    <course_number>{string}</course_number>
    <course_title>{string}</course_title>
    <course_name>{string}</course_name>
    <course_code>{string}</course_code>
    <course_section>{string}</course_section>
    <course_credits>{integer}</course_credits>
    <course_model>{string}</course_model>
    <department_code>{string}</department_code>
    <department_desc>{string}</department_desc>
    <campus_code>{string}</campus_code>
    <campus_desc>{string}</campus_desc>
    <term_code>{integer}</term_code>
    <term_desc>{string}</term_desc>
    <session_code>{string}</session_code>
    <start_date>{date}</start_date>
    <end_date>{date}</end_date>
    <enrollment_cap>{integer}</enrollment_cap>
    </course>
  </courses>
</response>
```

### Course Navigation Structure Usage

For the course navigation structure you can choose a three department level or a two department level structure. The fields that will determine one of these two structure are the campus_code and campus_desc. If you decide to have a three department level you must include the campus_code and campus_desc information. If you decide to have a two department level you must leave empty the campus_code and campus_desc fields.

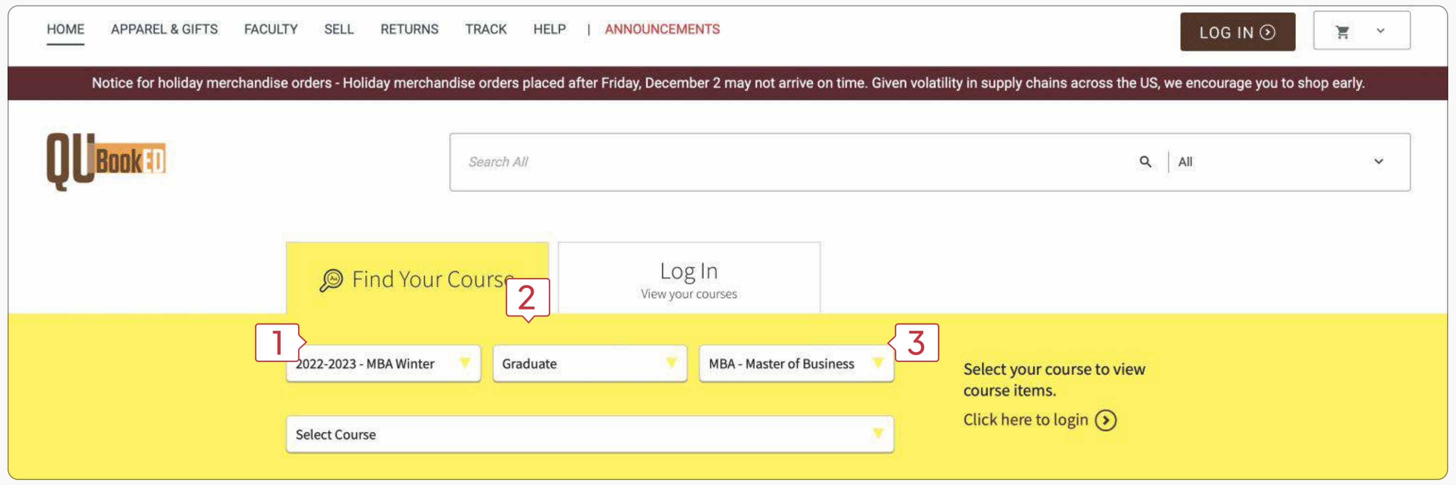

AKADÉMOS ADOPTION & ANALYTICS PORTAL
AKADEMOS
kt2022.demo.textbookx.com
1
Summer 2022
2
All



**3 levels:**

1. Term (here you will see the term_desc)
   2. Campus description (here you will see the campus_desc. It doesn't has to be the campus sub level, you can use this field to identify subterms, or different programs at your institution)

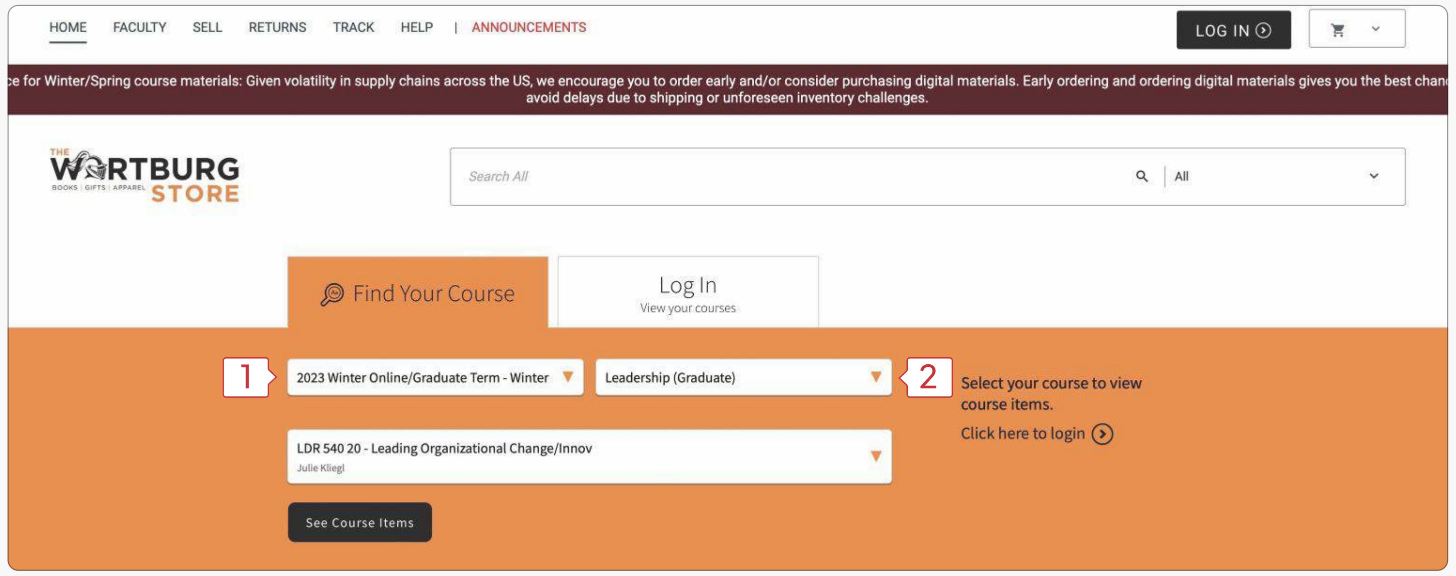
      3. Department description (here you will see the department_desc)
         Course title (here you will see the course_title)



Notice for holiday merchandise orders - Holiday merchandise orders placed after Friday, December 2 may not arrive on time. Given volatility in supply chains across the US, we encourage you to shop early.

**2 levels:**

1. Term (here you will see the term_desc)
   2. Department description (here you will see the department_desc)
      Course title (here you will see the course_title)



e for Winter/Spring course materials: Given volatility in supply chains across the US, we encourage you to order early and/or consider purchasing digital materials. Early ordering and ordering digital materials gives you the best chan avoid delays due to shipping or unforeseen inventory challenges.

## Request Body

Do not supply a request body with this method.

## • User Information Request

Contains data for both students and instructors. In the case of students an array of courses associates the student user with his or her active enrollments; in the case of instructors the array specifies courses the user teaches. User address data allows us to pre-populate required checkout fields in the case of students, and to tie in advanced adoption functionality (reminders, etc) in the case of instructors. Note that for instructors, physical address fields can be filled in with a fixed school address or left empty.
Full requests are typically made once per night, and partial requests are made upon successful authentication of the student into the system. Partial requests can only be served through a real-time method. The partial request is not mandatory, however it allows students to see their most up-to-date enrollment data. This is especially useful if students wish to add a course and immediately log into the bookstore to view the relevant booklist.
**All fields are required.**

### + Full Request

## HTTP Request

```
GET /courses
```

## Resource Representation (JSON)

```
“status_code”: {integer},
“status_message”: {string},
“users”: [
{
“id”:{string},
“username”: {string},
“role”: {string},
“first_name”: {string},
“last_name”: {string},
“email”: {string},
“phone_number”: {string},
“address_line_1”: {string},
“address_line_2”: {string},
“city”: {string},
“state”: {string},
“postal_code”: {string},
“student_major”: {string},
“student_grade_level”: {string},
“courses”: [ {
   “course_number”: {string},
   “term_code”: {string},
   “term_desc”: {string},
   “special_type”: {string}
   “student_type”: {string}
} ]
}
]
```

## Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
  <users>
  <user>
    <id>{string}</id>
    <username>{string}</username>
    <role>{string}</role>
    <first_name>{string}</first_name>
    <last_name>{string}</last_name>
    <email>{string}</email>
    <phone_number>{string}</phone_number>
    <address_line_1>{string}</address_line_1>
    <address_line_2>{string}</address_line_2>
    <city>{string}</city>
    <state>{string}</state>
    <postal_code>{string}</postal_code>
    <student_major>{string}</student_major>
    <student_grade_level>{string}</student_grade_level>
    <courses>
      <course>
        <course_number>{string}</course_number>
        <term_code>{string}</term_code>
        <term_desc>{string}</term_desc>
        <special_type>{string}</special_type>
        <student_type>{string}</student_type>
      </course>
      ...
    </courses>
  </user>
</users>
...
</response>
```

## + Partial Request

### HTTP Request

```
GET/{user_id}/courses?role={user_role}
```

Role values: student, professor

### Resource Representation (JSON)

```
"status_code": {integer},
"status_message": {string} ,
"user": {
  "id": {string} ,
  "username": {string},
  "role": {string},
  "first_name": {string},
  "last_name": {string},
  "email": {string},
  "phone_number": {string},
  "address_line_1": {string},
  "address_line_2": {string},
  "city": {string},
  "state": {string},
  "postal_code": {string},
  "student_major": {string},
  "student_grade_level": {string},
  "courses": [{
    "course_number": {string},
    "term_code": {string},
    "term_desc": {string},
    "special_type": {string}
    "student_type": {string}
  }]
}
```

### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
  <user>
    <id>{string}</id>
    <username>{string}</username>
    <role>{string}</role>
    <first_name>{string}</first_name>
    <last_name>{string}</last_name>
    <email>{string}</email>
    <phone_number>{string}</phone_number>
    <address_line_1>{string}</address_line_1>
    <address_line_2>{string}</address_line_2>
    <city>{string}</city>
    <state>{string}</state>
    <postal_code>{string}</postal_code>
    <student_major>{string}</student_major>
    <student_grade_level>{string}</student_grade_level>
    <courses>
      <course>
        <course_number>{string}</course_number>
        <term_code>{string}</term_code>
        <term_desc>{string}</term_desc>
        <special_type>{string}</special_type>
        <student_type>{string}</student_type>
      </course>
      ...
    </courses>
  </user>
</response>
```

### Request Body

Do not supply a request body with this method.

## b Flat file via SFTP integration

### I Access

To access the SFTP Akademos will provide an username and password to the corresponding IT contact at your institution. Akademos also support a public key authentication access. To set up the public key you can contact akademos.integrations@vitalsource.com and send your public key and our IT team will install it.

To set up the SFTP please use the following information:

Server: ftp.akademos.com

SFTP Port = 2122

**FTPS:** To use FTPS, you have to use a compatible client that supports TLS and use FTP port 21.

Akademos will create two directories: TEST and PROD, each directory will have an user, course and term folders . To start with the integration and make sure that the data structure is consistent your institution IT team will have to upload the files in the TEST directory. Once Akademos completes the integration development, we will request your IT team to start uploading the files to the PROD directory.

To ensure  the data will be updated, Akademos recommends to upload at least one set of files per day. Akademos will schedule a synchronization process at a time of your institution convenience.

## II File names

Akademos can support different file formats such as csv, xml and json.

```
user_YYYYMMDDHHMMSS
course_YYYYMMDDHHMMSS
term_YYYYMMDDHHMMSS
```

## III Flat files components

### • User file

| id | role | first_name | last_name | email | phone_number | address_line1 | city | state | postal_code | student_major | student_grade_level | course_number | term_code | term_desc | username | special_type |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S01010 | student | S1First | S1Last | student1@akd.edu | 5559871234 | 147 Any St | Metropolis | KS | 66002 | Civ. Eng. | Graduate | 1605 | 2023FA | Fall 2023 | username1 | EA |
| S01010 | student | S1First | S1Last | student1@akd.edu | 5559871234 | 147 Any St | Metropolis | KS | 66002 | Civ. Eng. | Graduate | 1709 | 2023FA | Fall 2023 | username1 | EA |
| S01010 | student | S1First | S1Last | student1@akd.edu | 5559871234 | 147 Any St | Metropolis | KS | 66002 | Civ. Eng. | Graduate | 1908 | 2023FA | Fall 2023 | username1 | EA |
| S01010 | student | S1First | S1Last | student1@akd.edu | 5559871234 | 147 Any St | Metropolis | KS | 66002 | Civ. Eng. | Graduate | 1019 | 2023FA | Fall 2023 | username1 | EA |
| P24356 | professor | P2First | S2Last | professor2@akd.edu | | | | | | | | 1605 | 2023FA | Fall 2023 | username2 | |
| P24356 | professor | P2First | S2Last | professor2@akd.edu | | | | | | | | 1092 | 2023FA | Fall 2023 | username2 | |

## • Course file

| course_number | course_title | course_name | course_code | course_section | course_credits | course_model | department_code | department_desc | campus_code | campus_desc | term_code | term_desc | session_code | start_date | end_date | enrollment_cap |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1605 | ADV ST: NYC EXPER PERFORMANCE | THTR | 465 | 1 | 3 | EA | THTR | THEATRE-ARTS | WC | West Campus | 2023FA | Fall 2023 | 8W | 8/26/2023 | 12/15/2023 | 6 |
| 1908 | GEN PSYCH;SOC & CLIN PROCESSE | PSYCH | 101 | OL01 | 3 | EA | PSYCH | PSYCHOLOGY | WC | West Campus | 2023FA | Fall 2023 | 8W | 8/26/2023 | 12/15/2023 | 5 |
| 1019 | MANAGEMENT | BUS | 200 | OL01 | 3 | EA | BUS | BUSINESS | EC | East Campus | 2023FA | Fall 2023 | 16W | 8/26/2023 | 12/15/2023 | 12 |
| 1020 | ADVERTISING I | COMM | 232 | OL01 | 4 |  | COMM | COMMUNICATION-ARTS | EC | East Campus | 2023FA | Fall 2023 | 16W | 8/26/2023 | 12/15/2023 | 20 |

## • Term file

| term_code | start_date | end_date |
|---|---|---|
| 2023SP | 1/9/2023 | 5/7/2023 |
| 2023SU | 5/20/2023 | 8/15/2023 |
| 2023FA | 8/26/2023 | 12/15/2023 |

← Back to Table of Contents

# 04 Payment Integrations

## a API – Order Request

The order request API set allows the bookstore to initiate financial transactions with your aid, bursar or other debiting account system. This option will allow students to make their purchases with the financial instrument they may prefer to utilize. Orders may be split between order request methods and traditional, private methods such as student credit card.

Akademos and the institution developers will need to agree upon security protocols (basic authentication protocols, signatures) before providing the initial end points. Akademos will require the staging API endpoints for the testing phase and once everything has been validated Akademos will request the production API endpoints before release.

### I Balance Inquiry

A method which returns the authenticated user's current available funds.

#### HTTP Request

```
GET /{user_id}/balance
```

#### Resource Representation (JSON)

```
{
  "balance": {number},
  "status_code": {integer},
  "status_message": {string}
}
```

#### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <balance>{number}</balance>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
</response>
```

#### Request Body

Do not supply a request body with this method.

#### Field Definitions

| Property Name | Value | Description |
|---|---|---|
| balance | number | The user's current account balance. |
| status_code | integer | Represents status of request. 0 indicates success |
| status_message | string | Represents specific information regarding failure |

## II Hold

Initially when a student submits their order, the Hold request should reserve the funds from the student's account for at least 14 days, at which time the hold can be reversed back to the student's account automatically. Holds will be converted to charges (made permanent) when the items are confirmed as shipping, in the Charge request. Hold requests are typically done immediately following a Balance Inquiry, to avoid declining balance failures.

### HTTP Request

```
POST /{user_id}/hold
```

### Resource Representation (JSON)

```
{
  "id": {string},
  "status_code": {integer},
  "status_message": {string}
}
```

### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <id>{string}</id>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
</response>
```

### Field Definitions

| Property Name | Value | Description |
|---|---|---|
| id | string | Identifier for this hold request. |
| status_code | integer | Represents status of request. 0 indicates success. 5 indicates insufficient funds. |
| status_message | string | Represents specific information regarding failure. |

### Request Body (JSON)

```
{
  "amount": {number}
}
```

### Request Body (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <amount>{number}</amount>
</request>
```

## III Charge

The bookstore will initiate a Charge on previously-submitted Holds, which indicates that the order is being shipped and the reserved funds can be committed in the school's transaction system.
Charges are made once per order, and will never be more than the original Hold, although they may be less depending on the state of the items within the

order. Any amount between the original Hold and the subsequent Charge should be reversed back to the student's account.
Should an order be cancelled entirely, the Bookstore will send a Charge of "0" to indicate that the entire Hold amount should be reversed to the student's account.

### HTTP Request

```
POST /{user_id}/payment
```

### Resource Representation (JSON)

```
{
  "id": {string},
  "status_code": {integer},
  "status_message": {string}
}
```

### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <id>{string}</id>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
</response>
```

### Field Definitions

| Property Name | Value | Description |
|---|---|---|
| id | string | Identifier for this hold request. |
| status_code | integer | Represents status of request. 0 indicates success. 5 indicates insufficient funds. |
| status_message | string | Represents specific information regarding failure. |

### Request Body (JSON)

```
{
  "hold_id": {integer},
  "aka_order_id": {integer},
  "amount": {number},
  "order_items": [
    {
      "aka_item_id": {integer},
      "title": {string},
      "quantity": {integer},
      "amount": {number}
    }
  ]
}
```

## Request Body (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<request>
  <hold_id>{integer}</hold_id>
  <aka_order_id>{integer}</aka_order_id>
  <amount>{number}</amount>
  <order_items>
    <item>
      <aka_item_id>{integer}</aka_item_id>
      <title>{string}</title>
      <quantity>{integer}</quantity>
      <amount>{number}</amount>
    </item>
  </order_items>
</request>
```

## Field Definitions

| Property Name | Value | Description |
|---|---|---|
| hold_id | string | Identifier of the previous hold request. |
| aka_order_id | integer | Akademos Order ID (one per order) |
| amount | number | Requested payment amount. This must be less than or equal to* the amount request in the hold. |
| order_items | array | Contains the following rows: |
| aka_item_id | integer | Specifies a product in the bookstore system. Will be "0" when identifying shipping cost. |
| title | string | The title of the item being purchased. |
| quantity | integer | The quantity confirmed as shipping to the customer. |
| amount | number | The unit price of the item being purchased.** |
| crn | integer | A combination of course_number and term_code to specify a unique course (Optional).*** |
| term | string | Specific term_code related to the crn (Optional).*** |

* Payment request amount MAY BE LESS than Hold Amount due to canceled items. School should automatically retire the difference between Hold Amount and Payment Request Amount (Akademos will charge each order only once).
** order_items[amount] is on a unit basis, so order_items[amount] * order_items[quantity] = total price per item. Note that the sum of these subtotaled amounts from item array MAY BE MORE than payment request amount -- this is due to secondary payments like credit cards not being charged to the school's transaction system.
*** crn and term are optional fields, available for inclusion as per your institution's specific requirements.

## IV Refund

Refunds are issued in order to credit student accounts based on successful Charge requests. These may be done to accommodate customer returns or failed shipments. An order may be subject to more than one Refund request, however aggregate refund amounts will never exceed the original Charge amount.

### HTTP Request

```
POST /{user_id}/refund
```

## Resource Representation (JSON)

```
{
"id": {string},
"status_code": {integer},
"status_message": {string}
}
```

## Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <id>{string}</id>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
</response>
```

## Field Definitions

| Property Name | Value | Description |
|---|---|---|
| id | string | Identifier for this transaction. |
| status_code | integer | Represents status of request. 0 indicates success. |
| status_message | string | Represents specific information regarding failure. |

## Request Body (JSON)

```
{
  "payment_id": {string},
  "amount": {number},
  "refund_items": [
    { "aka_item_id": {integer} }
  } ]
}
```

## Request Body (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<request>
  <payment_id>{string}</payment_id>
  <amount>{number}</amount>
  <refund_items>
    <aka_item_id>{integer}</aka_item_id>
  </refund_items>
</request>
```

## Field Definitions

| Property Name | Value | Description |
|---|---|---|
| payment_id | string | Identifier from the payment request. |
| amount | number | Requested refund amount.* |
| refund_items | array | Array of items returned.** |

* Refund amounts will be expressed in positive numbers
** Note that refunds can be non-item specific, or can be for items but not for the full charge amount. Akademos recommends either leaving off the refund_items array entirely to avoid

confusion, or indicating only the book and not using that information to predict or display refund amount in school system.

### v Error Codes

- **General Codes**
  - code: 0
    message: success
  - code: 1
    message: unauthorized
  - code: 2
    message: user data not found
  - code: 3
    message: user not found
  - code: 4
    message: transaction failed
  - code: 5
    message: insufficient funds

- **Hold Codes**
  - code: 10
    message: funds not available

- **Payment Codes**
  - code: 20
    message: corresponding hold transaction not found
  - code: 21
    message: hold amount does not match payment amount

## b Automated Voucher Processing

With this option your institution can create an automated process by setting up a script to upload a template file to Akademo's SFTP.
This method is recommended if your institution has a large population that will received the credits.

Once the vouchers are issued student's will receive a notification email with a code, balance and expiration information.
Akademos is also capable of setting up specific restriction for the usage of these vouchers.

**Field definitions**

| Property Name | Description |
|---|---|
| your_initials | The initials of the person making the voucher. |
| student_name | The student's real name. |
| dollar_amount | The total amount for the textbook & shipping or Financial Aid amount. No $ sign needed. |
| voucher_number | This is a 7 digit alphanumeric code that you create. It's a best practice to come up with a standard convention for these. We recommend adding the school or program initials for the 1st few digits, then add any numbers/letters that identify the student, and then a term identifier. For example: FA1234F22 might be FA (financial aid) for student ID 1234 for the Fall 22 term. For example: FA1234F22 might be FA (financial aid) for student ID 1234 for the Fall 22 term. |
| expiration_date | Add a date for when you would like this voucher to expire. We suggest one or two weeks after the first day of class. |
| vendor_ID | Leave blank unless you have been given a Vendor ID. |
| email_address | An email will be sent to the email entered in this field, with instructions on how to use the voucher. If you use the same email address that is in our system (tied to the SIS system) the funds will automatically appear in the student's bookstore account. |
| Student ID | The student ID provided in the user data set. |
| tax_exempt | Not subject to taxation. |

### Voucher template

| Your Initials | Studen's Name | Voucher Number | Dollar Amount | Student's Email Address | Notify Student | Expiration Date | Student ID | tax_exempt |
|---|---|---|---|---|---|---|---|---|
| AK | Franklin Smith | 2123411402 | 100 | fsmith@school.com | yes | 11-01-2023 | 12345 | 1 |
| AK | Amanda Jones | 7902913123 | 150 | ajones@school.edu | yes | 11-01-2023 | 12346 | 1 |
| AK | Doug Wheatly | 1981413432 | 250 | dwheatly@school.edu | yes | 08-15-2023 | 12347 | 1 |
| AK | Karen Watts | 1311235134 | 500 | kwatts@schooledu | yes | 08-15-2023 | 12348 | 0 |
| AK | Arnold Douglas | 1231613454 | 700 | adouglas@school.edu | yes | 08-15-2023 | 12349 | 0 |

## c Bookstore Charge Program (BCP)

With this solution student's can be assigned a credit amount. To create these credits you can add four extra fields to the user information request if you are transmitting your data through an API or the user file if you are using the flat file method.

**How it works?**
Akademos will issue this balance only once per term. You will need to update the expiration date for every term. For example if you add a balance of $500 with 'credit_start_date' of 10/26/23 and 'credit_exp_date' of 1/26/2024 for Fall 2023, we will issue the $500 only once even if the students is enrolled in more than one course. As this file will be sent every day, we have a logic that won't allow additional amounts to be added more than once.
If only some specific students will receive the book vouchers you'll need to add the 'credit_amount', 'credit_exp_date', 'credit_notify', 'credit_start_date' data only for those students. This means that for the students that won't receive the book vouchers you may leave the fields mentioned blanks/empty.

### Field definitions

| Property Name | Value | Description |
|---|---|---|
| credit_amount | integer | The dollar credit amount that is issued to users to be used in the bookstore. |
| credit_exp_date | date | Selected date of when credit amount is expired. Credit value changes to '0' value. |
| credit_notify | string | Values: 'yes' or 'no'. If value is 'Yes' an email notification is sent to user including their credit amount and expiration date. |
| credit_start_date | date | This field can be left blank or you can indicate the exact date you want the credits to be issued. |

### User information request with additional credit block

### Resource Representation (JSON)

```
“status_code”: {integer},
“status_message”: {string} ,
“user”: {
  “id”: {string} ,
  “username”: {string} ,
  “role”: {string} ,
  “first_name”: {string} ,
  “last_name”: {string} ,
  “email”: {string} ,
  “phone”: {string} ,
  “address_line_1”: {string} ,
  “address_line_2”: {string} ,
  “city”: {string} ,
  “state”: {string} ,
  “postal_code”: {string} ,
  “student_major”: {string} ,
  “student_grade_level”: {string} ,
  “courses”: [ {
    "course_number": {string} ,
    "term_code": {string} ,
    “term_desc”: {string}
    “special_type”: {string} ,
  } ]
  “credits”: [ {
    "term_code": {string} ,
    “credit_amount”: {integer}
    “credit_exp_date”: {date}
    “credit_start_date”: {date}
    “credit_notify”: {string}
} ]
 }
```

### Resource Representation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
  <status_code>{integer}</status_code>
  <status_message>{string}</status_message>
  <users>
  <user>
    <id>{string}</id>
    <username>{string}</username>
    <role>{string}</role>
    <first_name>{string}</first_name>
    <last_name>{string}</last_name>
    <email>{string}</email>
    <phone>{string}</phone>
    <address_line_1>{string}</address_line_1>
    <address_line_2>{string}</address_line_2>
    <city>{string}</city>
    <state>{string}</state>
    <postal_code>{string}</postal_code>
    <student_major>{string}</student_major>
    <student_grade_level>{string}</student_grade_level>
    <courses>
      <course>
        <course_number>{string}</course_number>
        <term_code>{string}</term_code>
        <term_desc>{string}</term_desc>
        <special_type>{string}</special_type>
      </course>
    </courses>
    <credits>
      <credit>
        <term_code>{string}</term_code>
        <credit_amount>{integer}</credit_amount>
        <credit_exp_date>{date}</credit_exp_date>
```

```
          <credit_notify>{string}</credit_notify>
          <credit_start_date>{date}</credit_start_date>
        <credit>
        ...
      </credits>
    </user>
  </users>
  ...
  </response>
```

## d Payment Partners

Akademos has existing integrations with several payment partners to allow students to use their financial aid, scholarships or school funds in the bookstore.

For all of the following integrations during development, Akademos requires a test user (only student ID) with a balance greater than $0. The student ID must be the same ID provided in the user data. If it's an ID different from what is provided in the user data, the ID must be included in the username field within the user data.

### I CBORD

CBORD is a cloud-based campus card solution.

To proceed with the integration your institution must set up the Odyssey server and provide to Akademos the following information:

- Server IP
- Port
- Sequence_limit
- Code_map
- Location
- Operator
- Media_entry

### II Trimdata (FA ~link)

Trimdata is an API solution.

To proceed with the integration Akademos will need the following information:

- Store ID (store number)
- AR codes
- Register numbers

### III Watchman Payment System

To proceed with the integration more details will be shared on a technical discovery call.

## IV TransactCampus

To proceed with the integration more details will be shared on a technical discovery call.

## V TouchNet

TouchNet is an API solution.

To proceed with the integration Akademos will need the following information:

- OperatorID
- PIN (operator password)
- TerminalID
- TerminalType

# 05 Akademos Data Exports via SFTP

## a Transaction report

When using one of the payment integration solutions such as the Voucher tool/ upload or BCP or if your institution is using Inclusive Access (IA), Akademos can generate a daily report to reflect all the transactions made by the students. This report can be placed in the SFTP under the voucher_return directory or can be sent to a specific email at your institution. The file format can be a csv or a txt file.
Akademos can provide a default sample file, however, the report can be customized upon your institution requirements.

### Field definitions

| Property Name | Description |
|---|---|
| Transaction Date | The date when the charge or refund was made. |
| Full Name | Student's first and last name that comes from the user file. |
| Account ID | It can be the same value sent as ID or username in the user file. |
| Voucher Code | Unique identifier. |
| Charge Amount | If it shows as a positive value it represents the charge amount. If it's a negative value it represents a refund. |
| Vendor ID | 4-digit institute ID or voucher agency ID (provided by Akademos). |

### Transaction report template

| Transaction Date | Full Name | Account ID | Voucher Code | Amount | Vendor ID |
|---|---|---|---|---|---|
| 12/14/23 | Student1 | 799007432 | 799007432-FM22 | 483.9 | 2542 |
| 12/14/23 | Student2 | 799007433 | 799007433-FM23 | 483.9 | 2542 |
| 12/14/23 | Student3 | 799007434 | 799007434-FM24 | -483.9 | 2542 |

## 06 LMS

Depending on the LMS your institution uses, Akademos will provide an specific installation guide for your LMS platform.

We have solutions for all major LMS including LTI tools for:

- Canvas
- Moodle
- Brightspace (D2L)
- Blackboard

## Need help? Contact us

**Technical Support**
akademos.integrations@vitalsource.com

**Brett Horton**
Sr. Product Manager
203-852-3938
brett.horton@vitalsource.com