IMS GLOBAL Learning Consortium

1EdTech OneRoster®: Specification

1EdTech Final Release
Version 1.1

Date Issued: 29th April, 2022

Latest version: http://www.imsglobal.org/lis/

IPR and Distribution Notices

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

1EdTech takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on 1EdTech's procedures with respect to rights in 1EdTech specifications can be found at the 1EdTech Intellectual Property Rights web page: http://www.imsglobal.org/ipr/imsipr_policyFinal.pdf.

Use of this specification to develop products or services is governed by the license with 1EdTech found on the 1EdTech website: http://www.imsglobal.org/speclicense.html.

Permission is granted to all parties to use excerpts from this document as needed in producing requests for proposals.

The limited permissions granted above are perpetual and will not be revoked by 1EdTech or its successors or assigns.

Public contributions, comments and questions can be posted here: http://www.imsglobal.org/forums/ims-glc-public-forums-and-resources/learning-information-services-oneroster-public-forum.

Document Name: 1EdTech OneRoster® v1.1 Specification Document Release 2.0.1

Revision: 29th April, 2022

Table of Contents

1. Introduction

1.1. OneRoster Overview

1.2. Context

1.3. Structure of this Document

1.4. References

1.5. Nomenclature

2. Specific Requirements for LIS in K12

3. RESTful Service Model

3.1. Endpoints

3.2. Service State Interactions

3.2.1. Service Consumer Driven Events (Pull Model)

3.2.2. Service Provider Driven Events (Push Model)

3.3. API Root URL and Version

3.4. Binding of Resource Data

3.4.1. Pagination

3.4.2. Sorting

3.4.3. Filtering

3.4.4. Field Selection

3.5. Error Handling

3.6. Security [R19]

3.6.1. Core Security

3.6.2. OAuth 2 Bearer Token Authorization

3.7. Serialization Format [R9]

3.8. Extensions [R36]

4. Data Model

4.1. The Base Class [R35, R40]

4.1.1. Extensions

4.2. AcademicSession [R1, R12, R25]

4.3. Class [R16, R21, R23, R30, R38, R39]

4.4. Course [R15, R22, R24, R29]

4.5. Demographic Data [R42]

4.6. Enrollments [R43]

4.7. Line Items [R6, R7, R20]

4.8. Line Item Categories [R3, R4]

4.9. Org [R21, R40]

4.10. Resources [R46, R47, R48]

4.11. Results [R5, R27, R28]

4.12. Users, Students, Teachers [R2, R14, R17, R31, R32, R33, R38, R41, R42]

4.13. Enumerated Vocabularies

4.13.1. ClassType

4.13.2. Gender

4.13.3. Importance

4.13.4. OrgType

4.13.5. RoleType

4.13.6. ScoreStatus

4.13.7. SessionType

4.13.8. StatusType

4.14. Common Data Models

4.14.1. GUID

4.14.2. GUIDRef

4.15. Base Data-types

5. JSON Binding

5.1. AcademicSessions

5.2. Class

5.3. Course

5.4. Demographics

5.5. Enrollments

5.6. LineItem

5.7. LineItem Categories

5.8. Org

5.9. Resource

5.10. Result

5.11. Teachers and Students (Users)

5.12. Returning An Array of Objects

5.13. Metadata

5.14. Error Payloads

Appendix A - Recommended Vocabularies

Appendix B - The Complete Data Model

About This Document

List of Contributors

Revision History

1. Introduction

1.1. OneRoster Overview

Learning Information Services (LIS) is a standard that is maintained by 1EdTech [LIS, 13]. The standard addresses the exchange of student data (about people, courses, enrollments and grades) between different educational systems. LIS has very wide adoption in Higher Education (HE). OneRoster is a reworking of that

specification to support the specific needs of K-12. OneRoster is a considerably simplified LIS service and data model. The binding for OneRoster is either a pure data format using Comma Separated Value (CSV) files or as a service with data exchange between service providers and service consumers using a RESTful approach carrying Java Script Object Notation (JSON)-encoded data.

1.2. Context

1EdTech Members representing the interests of the K12 sector completed an evaluation of LIS. They found LIS to broadly meet their needs when it came to student enrollment scenarios, but desired extra functionality when it came to the transport of grades and grade-books.

The group wanted to use functionality of LIS, but to make it simpler to use. In LIS, there are eleven documents that describe the six LIS services and the information models that underpins them. More documents describe the XML schemas, WSDL files, conformance tests and more. Most of these documents are generated from the computer UML models of LIS, and then augmented with highly technical commentary. The LIS working group created the LIS that limited implementers to building a subset of the services and operations. The core profile means that developers need only consider the replace and delete operations for the person, group, membership and course section services. Conformance to the core profile can be tested, and all of the LIS conformant products to date are to the core profile. The LIS group also created the FINAL GRADE profile of LIS, and this is a statement of the operations that developers need to build in order to move final grades for course sections between systems. Conformance to this profile can be tested.

The OneRoster working group published the 1EdTech OneRoster V1.0 Specification with REST and CSV binding in July 2015. This document is the 1EdTech OneRoster V1.1 Specification that contains the REST-based binding. The equivalent CSV binding is published in [OneRoster, 20a]. The technical changes introduced in V1.1 are:

- Support for describing resources associated with a Course has been introduced to the data model;
- Support for describing resources associated with a Class has been introduced to the data model;
- The 'periods' attribute has been added to the Class data model;
- The 'schoolYear' attribute has been added to the AcademicSession data model;
- The 'userEnabled', 'grades', 'password' and 'middleName' attributes have been added to the User data model;
- The 'userId' structure has been amended to include the type of identifier to allow multiple entries and renamed 'userIds';
- The 'beginDate' and 'endDate' attributes have been added to the Enrollment data model;
- The 'periods' attribute has been added to the Class data model;
- The 'subjectCodes' attribute has been added to the Class and Course data models;
- In the Result data model the 'date' attribute as been renamed 'scoreDate' and the 'statusOfResult' renamed to scoreStatus';
- The 'resultValueMin' and 'resultValueMax' attributes have be added to the LineItem data model;

- New operations of 'deleteCategory()' and putCategory()' have been added;
- New operations of 'getAllResults()', 'getResult()', deleteResult()' and putResult()' have been added;
- New operations of 'getAllLineItems()', 'getLineItem()', deleteLineItem()' and putLineItem()' have been added;
- New operations of 'getAllResources()' and 'getResource()' have been added;
- New operations of 'getClassesForUser()', 'getResourcesForCourse()' and 'getResourcesForClass()' have been added;
- Required usage of SHA2 (256) with the OAuth 1.0a for request signing;
- The security model has been changed to REQUIRE the use of OAuth 2 Bearer Tokens and Transport Layer Security (TLS). Support for TLS 1.2 or TLS 1.3 is REQUIRED and use of SSL is now PROHIBITED.

There have also been a number of editorial refinements, clarifications and corrections made in this document. The other document changes are:

a) The 'Conformance Testing' has been moved to the 'OneRoster 1.1 Conformance and Certification' document [OneRoster, 20c];

b) The 'Best Practices' section has been moved to the 'OneRoster 1.1 Best Practices and Implementation Guide' [OneRoster, 20b].

The final change is that the endpoints have been grouped into one of three modes:

- Rostering - to enable the management of academicSessions, classes, courses, demographics, enrollments, gradingPeriods, orgs, schools, students, teachers, terms and Users;
- Resources - to enable the management of resources;
- Gradebook - to enable management of the lineItems, results and categories.

NOTE: New features are denoted by yellow shading and features that have been modified are denoted by blue shading.

1.3. Structure of this Document

The structure of the rest of this document is:

2. SPECIFIC REQUIREMENTS FOR LIS IN K12	The set of requirements to be supported by the OneRoster specification;
3. RESTFUL SERVICE MODEL	The definition of the service calls and the mapping of these to the equivalent HTTP verbs and endpoints;
4. DATA MODEL	The definition of the data that can be exchanged using the REST/SOAP/CSV bindings;
5. JSON BINDING	The representation of the data model as JSON data carried in the REST calls;

APPENDIX A - RECOMMENDED VOCABULARIES	Consolidation of the set of vocabularies that are defined within the data model classes;
APPENDIX B - THE COMPLETE DATA MODEL	A visualization of the full data model.

1.4. References

[ISO 8601]	*ISO8601:2004 Data elements and interchange formats - Information interchange - Representation of dates and times*, ISO, International Standards Organization (ISO), 2000, p.33.
[LIS, 13]	*1EdTech Learning Information Services*, C.Smythe 1EdTech, July 2013.
[LTI, 10]	*1EdTech Learning Tools Interoperability (LTI)* v1.0, S.Vickers, 1EdTech, May 2010.
[OneRoster, 15a]	*OneRoster Specification v1.0*, P.Nicholls, 1EdTech, July 2015.
[OneRoster, 15b]	*OneRoster Specification: CSV Tables v1.0*, P.Nicholls, 1EdTech, July 2015.
[OneRoster, 20a]	*OneRoster Specification: CSV Tables v1.1.1 Final Release*, P.Nicholls and C.Smythe, 1EdTech, December 2020.
[OneRoster, 20b]	*OneRoster v1.1 Best Practices and Implementation Guide*, C.Smythe and P.Nicholls, Version 1.1 Final Release, 1EdTech, December 2020.
[OneRoster, 20c]	*OneRoster v1.1 Conformance and Certification*, L.Mattson and C.Smythe, Version 1.1 Final Release, 1EdTech, December 2020.
[RFC 2617]	HTTP Authentication: Basic and Digest Access Authentication, J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen and L. Stewart, IETF RFC 2617, IETF, 1999. https://tools.ietf.org/pdf/rfc2617.pdf
[RFC5849]	*OAuth Version 1.0* IETF, April 2010.
[RFC 6749]	The OAuth 2.0 Authorization Framework, D.Hardt, IETF RFC 6749, IETF, 2012. https://tools.ietf.org/pdf/rfc6749.pdf
[RFC 6750]	The OAuth 2.0 Authorization Framework: Bearer Token Usage, D.Hardt, IETF RFC 6750, IETF, 2018. https://tools.ietf.org/pdf/rfc6750.pdf
[SecurityFramework]	1EdTech Security Framework 1.0, C.Smythe, N.Mills, C.Vervoort and M.McKell, 1EdTech, 2018. https://www.imsglobal.org/spec/security/v1p0/
[UNICODE, 16]	*UNICODE Collation Algorithm Version 9.0*, M.Davis, K.Whistler and M.Scheer, Unicode® Technical Standard #10, May 2016.

1.5. Nomenclature

API	Application Programming Interface
CEDS	Common Education Data Standards

CMS	Course Management Service
CSV	Comma Separated Values
GMS	Group Management Service
GUID	Globally Unique Identifier
HE	Higher Education
HMAC-SHA	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
IETF	Internet Engineering Task Force
ISO	International Standards Organization
JSON	Java Script Object Notation
JSON-LD	Java Script Object Notation Linked Data
LIS	Learning Information Services
LTI	Learning Tools Interoperability
MMS	Membership Management Service
NCES	National Centre for Education Statistics
OMS	Outcomes Management Service

PII	Personally Identifiable Information
PMS	Person Management Service
REST	Representation State Transfer
RFC	Request For Comment
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transaction Layer Security
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
UUID	Universal Unique Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	Exchange Markup Language

2.Specific Requirements for LIS in K12

The following are the requirements for the LIS profile for K12.

This table of requirements has been updated since the previous version of this document. New requirements from the previous version are shown under the "NEW REQUIREMENTS" line (shaded in 'yellow').

ID#	Requirement
1	Support for a term object, with fields: ID, Title, Start Date, End Date.
2	Support for a username field in Person.
3	Support for a "Category" in relation to a line item.
4	Category fields: ID, Title.
5	Support for a Grade Comment in Result.
6	Support for an "Assignment" in relation to a Line Item.
7	Assignment Fields: ID, Title, Description, Due Date, Category [see 3], Grading Scale [result value].
8	RESTful Binding (HTTP Verbs, 'normal' RESTful URL patterns.
9	JSON or JSON-LD Binding.
10	Solution also possible in SOAP / WSDL.
11	Decide if there are multiple data types per file / message, or just one data type per file/message.
12	Simple Data Types: Course, Class, Teacher, Student, Term.
14	Simple Data Type: Student = Student ID and Student Name.
15	Simple Data Type: Course = Course ID and Course Name.
16	Simple Data Type: Class = Class ID and Class Name.
17	Simple Data Type: Teacher = Teacher ID and Teacher Name.
18	A school has multiple courses; a course has multiple classes; users take multiple classes and classes are made up of multiple users. Teachers teach in many Schools, Learners Learn in many Schools, a school has many teachers and a school has many learners.
19	There must be a notion of data and transactions security (authorization and encryption), which uses two-legged OAuth 1.0a, as LTI does.
20	Assignment Field: Date Assigned.
21	Define a structure for multiple levels or organization (school, district, state, country).

ID#	Requirement
22	Add a notion of "grade" - e.g. Elementary, Middle, High.
23	Class Field: Grade.
24	Course Field: Grade.
25	Add a notion of a Grading Period, which is a unit of time, in which a lineItem has been assessed.
26	Assignment Field: Grading Period.
27	Modify the vocabulary for Result Status: Not Submitted, Submitted, Partially Graded, Fully Graded, Exempt.
28	Remove Result Value from Result (Grade).
29	Class Field: Subject(s).
30	Course Field: Subject(s).
31	User Field: Student ID#, Teacher ID# - a human readable identifier for users.
32	User Field: Contact Information - Email, SMS, Phone
33	User Field: Parent Information, User Type.
34	Add API entry point to get all classes for student.
35	All classes: Add a version id, or a date, so that requesters can access just the records that have changed, Add a status ("active or tobedeleted").
36	Add support for an extensions mechanism, allowing new fields to be passed in the JSON.
37	Add endpoints: classes for teacher, students for class in school, teachers for class in school.
38	Make explicit the relationships between objects.
39	Add term information to class.
40	Add a mechanism for adding metadata to classes.
41	LTI User ID field to be renamed (07.04.2015).
42	Add Demographics to User.
43	Break out enrollments into own object.

	NEW REQUIREMENTS for V1.1.
44	Add support for the 'getClassesForUser' operation.
45	Extend the Course data model with support for assigning 'resources' to courses.
46	Add support for 'getResourcesForCourse' operation.
47	Extend the Class data model with support for assigning 'resources' to classes.
48	Add support for 'get' a description of a resource.
49	Add support for 'getResourcesForClass' operation.
50	Add the 'grade' data field to the User class in the data model.
51	Add the 'beginDate' and 'endDate' data fields to the Enrolment class in the data model.
52	Restructure the 'userId' field in the User class to support the type if identifier.
53	Add the 'password' data field to the User class in the data model.
54	Add the 'middleName' data field to the User class in the data model.
55	Add the 'periods' data field to the Class class in the data model.
56	Add the 'schoolYear' field to the AcademicSession data model.
57	Add the 'subjectCodes' data field to the Class and Course classes in the data model.
58	Add to the security model support for the use OAuth 2 Bearer Tokens with SHA-2 and TLS.
59	Add optional usage of SHA-2 with the OAuth 1.0a for request signing.
60	Add support for 'get', 'delete' and 'put' operations for LineItem objects.
61	Add support for 'get', 'delete' and 'put' operations for Result objects.
62	Add support for 'delete' and 'put' operations for Category objects.

Table 2.1 - Requirements for LIS in K12.

3. RESTful Service Model

This section describes the RESTful binding of the data model. This RESTful binding addresses requirements [R8, R11, R19, R34, R37, R44, R45, R46, R48, 50, 51, 62 and R63].

3.1. Endpoints

In OR 1.1 the available endpoints have been collected in three groups:

- Rostering - to enable the management of academicSessions, classes, courses, demographics, enrollments, gradingPeriods, orgs, schools, students, teachers, terms and Users;
- Resources - to enable the management of resources;

- Gradebook - to enable management of the lineItems, results and categories.

Tables 3.1a, 3.1b and 3.1c show the permitted HTTP verbs for each endpoint/resource type. See the OneRoster Conformance and Certification document [OneRoster, 17c] for details on the endpoints that MUST be supported.

Table 3.1a - HTTP Endpoints for Rostering.

Service Call	Endpoint	HTTP Verb	Action
getAllAcademicSessions	/academicSessions	GET	Return collection of all academic sessions.
getAcademicSession	/academicSessions/{id}	GET	Return specific Academic Session.
getAllClasses	/classes	GET	Return collection of classes.
getClass	/classes/{id}	GET	Return specific class.
getAllCourses	/courses	GET	Return collection of courses.
getCourse	/courses/{id}	GET	Return specific course.
getAllGradingPeriods	/gradingPeriods	GET	Return collection of grading periods. A Grading Period is an instance of an AcademicSession.
getGradingPeriod	/gradingPeriods/{id}	GET	Return specific Grading Period. A Grading Period is an instance of an AcademicSession.
getAllDemographics	/demographics	GET	Return collection of demographics.
getDemographics	/demographics/{id}	GET	Return specific demographics.

Service Call	Endpoint	HTTP Verb	Action
getAllEnrollments	/enrollments	GET	Return collection of all enrollments.
getEnrollment	/enrollments/{id}	GET	Return specific enrollment.
getAllOrgs	/orgs	GET	Return collection of Orgs.
getOrg	/orgs/{id}	GET	Return Specific Org.
getAllSchools	/schools	GET	Return collection of schools. A School is an instance of an Org.
getSchool	/schools/{id}	GET	Return specific school. A School is an instance of an Org.
getAllStudents	/students	GET	Return collection of students. A Student is an instance of a User.
getStudent	/students/{id}	GET	Return specific student. A Student is an instance of a User.
getAllTeachers	/teachers	GET	Return collection of teachers. A Teacher is an instance of a User.
getTeacher	/teachers/{id}	GET	Return specific teacher.
getAllTerms	/terms	GET	Return collection of terms. A Term

Service Call	Endpoint	HTTP Verb	Action
			is an instance of an AcademicSession.
getTerm	/terms/{id}	GET	Return specific term.
getAllUsers	/users	GET	Return collection of users
getUser	/users/{id}	GET	Return specific user
getCoursesForSchool	/schools/{id}/courses	GET	Return the collection of courses taught by this school.
getEnrollmentsForClassInSchool	/schools/{school_id}/classes/{class_id}/enrollments	GET	Return the collection of all enrollments into this class.
getStudentsForClassInSchool	/schools/{school_id}/classes/{class_id}/students	GET	Return the collection of students taking this class in this school.
getTeachersForClassInSchool	/schools/{school_id}/classes/{class_id}/teachers	GET	Return the collection of teachers taking this class in this school.
getEnrollmentsForSchool	/schools/{school_id}/enrollments	GET	Return the collection of all enrollments for this school.
getStudentsForSchool	/schools/{school_id}/students	GET	Return the collection of students attending this school.

Service Call	Endpoint	HTTP Verb	Action
getTeachersForSchool	/schools/{school_id}/teachers	GET	Return the collection of teachers teaching at this school.
getTermsForSchool	/schools/{school_id}/terms	GET	Return the collection of terms that are used by this school.
getClassesForTerm	/terms/{term_id}/classes	GET	Return the collection of classes that are taught in this term.
getGradingPeriodsForTerm	/terms/{term_id}/gradingPeriods	GET	Return the collection of Grading Periods that are part of this term.
getClassesForCourse	/courses/{course_id}/classes	GET	Return the collection of classes that are teaching this course.
getClassesForStudent	/students/{student_id}/classes	GET	Return the collection of classes that this student is taking.
getClassesForTeacher	/teachers/{teacher_id}/classes	GET	Return the collection of classes that this teacher is teaching.
getClassesForSchool	/schools/{school_id}/classes	GET	Return the collection of classes taught by this school.

Service Call	Endpoint	HTTP Verb	Action
getClassesForUser	/users/{user_id}/classes	GET	Return the collection of classes attended by this user.
getStudentsForClass	/classes/{class_id}/students	GET	Return the collection of students that are taking this class.
getTeachersForClass	/classes/{class_id}/teachers	GET	Return the collection of teachers that are teaching this class.

Table 3.1b - HTTP Endpoints for Resources.

Service Call	Endpoint	HTTP Verb	Action
getAllResources	/resources	GET	Return collection of resources.
getResource	/resources/{id}	GET	Return specific resource.
getResourcesForCourse	/courses/{course_id}/resources	GET	Return the collection of resources associated to this course.
getResourcesForClass	/classes/{class_id}/resources	GET	Return the collection of resources associated to this class.

Table 3.1c - HTTP Endpoints for Gradebooks.

Service Call	Endpoint	HTTP Verb	Action
getAllCategories	/categories	GET	Return collection of grading categories.
getCategory	/categories/{id}	GET	Return specific grading category.
deleteCategory	/categories/{id}	DELETE	Enable the associated SourcedId to be marked as deleted. An immediate read will result in 404 code.
putCategory	/categories/{id}	PUT	To create a new Category record or to replace one that already exists.
getAllLineItems	/lineItems	GET	Return collection of lineItems.
getLineItem	/lineItems/{id}	GET	Return specific lineItem.
deleteLineItem	/lineItems/{id}	DELETE	Enable the associated SourcedId to be marked as deleted. An immediate read will result in 404 code.
putLineItem	/lineItems/{id}	PUT	To create a new LineItem record or to replace one

Service Call	Endpoint	HTTP Verb	Action
			that already exists.
getAllResults	/results	GET	Return collection of results.
getResult	/results/{id}	GET	Return specific result.
deleteResult	/results/{id}	DELETE	Enable the associated SourcedId to be marked as deleted. An immediate read will result in 404 code.
putResult	/results/{id}	PUT	To create a new Result record or to replace one that already exists.
getResultsForClass	/classes/{class_id}/results	GET	Return the collection of results (assessed grades) for this class.
getLineItemsForClass	/classes/{class_id}/lineItems	GET	Return the collection of line items (Columns) in the gradebook for this class.
getResultsForLineItemForClass	/classes/{class_id}/lineItems/{li_id}/results	GET	Return the collection of results (assessed grades), for

Service Call	Endpoint	HTTP Verb	Action
			this specific lineItem (Column) for this class.
getResultsForStudentForClass	/classes/{class_id}/students/{student_id}/results	GET	Return the collection of results (assessed grades), for this specific student, attending this class.

In the case of a 'DELETE' it is not a requirement that the record is hard deleted. From the perspective of this interoperability specification the key requirement is that once a delete has been issued, any immediate 'GET' request will result in a 404 status code being returned.

3.2. Service State Interactions

3.2.1. Service Consumer Driven Events (Pull Model)

The state diagram for the 'Pull Model" based data exchange is shown in Figure 3.1. In this model the service consumer must read the data from the service provider.

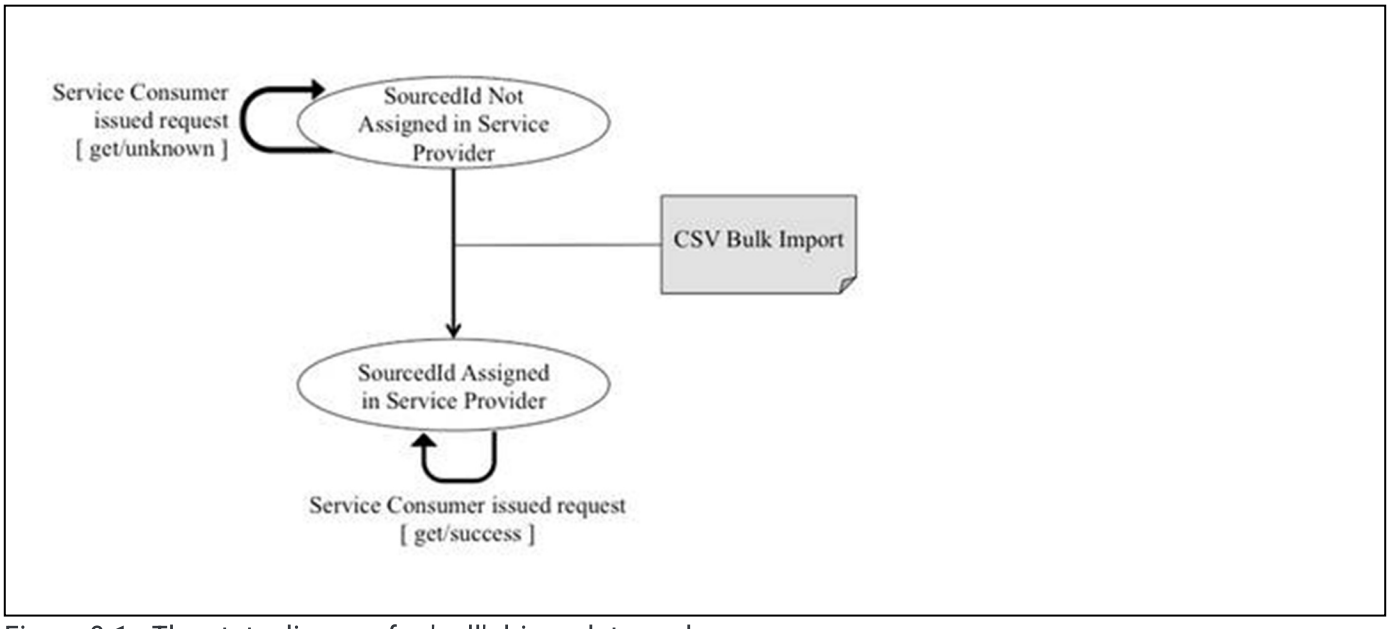


Figure 3.1 - The state diagram for 'pull' driven data exchange.

The state diagram shows that if a read request (from the service consumer) is issued on a 'sourcedId' before it has been assigned in the service provider then a failure/unknown error will occur. Once the 'sourcedId' has been assigned (for example using the OneRoster CSV Bulk Import) then a read request should be successful and the request object returned in the service payload.

For OR 1.1 the push capability is ONLY available for the Rostering, Resources and Gradebook services.

3.2.2. Service Provider Driven Events (Push Model)

The state diagram for the 'Push Model" based data exchange is shown in Figure 3.2. In this model the service provider must write the data into the service consumer.

The state diagram shows that the push process can only begin once a 'sourcedId' has been assigned within the service provider (for example using the OneRoster CSV Bulk Import). The service provider can now create a new record in a service consumer by issuing the write request (HTTP Put request). If this is the first time this 'sourcedId' has been allocated in the service consumer a 'create success' response will be returned. If this is a repeated write request then the response will be a 'replace success'. The service provider can 'delete' the record.

For OR 1.1 the push capability is ONLY available for the Gradebook service.

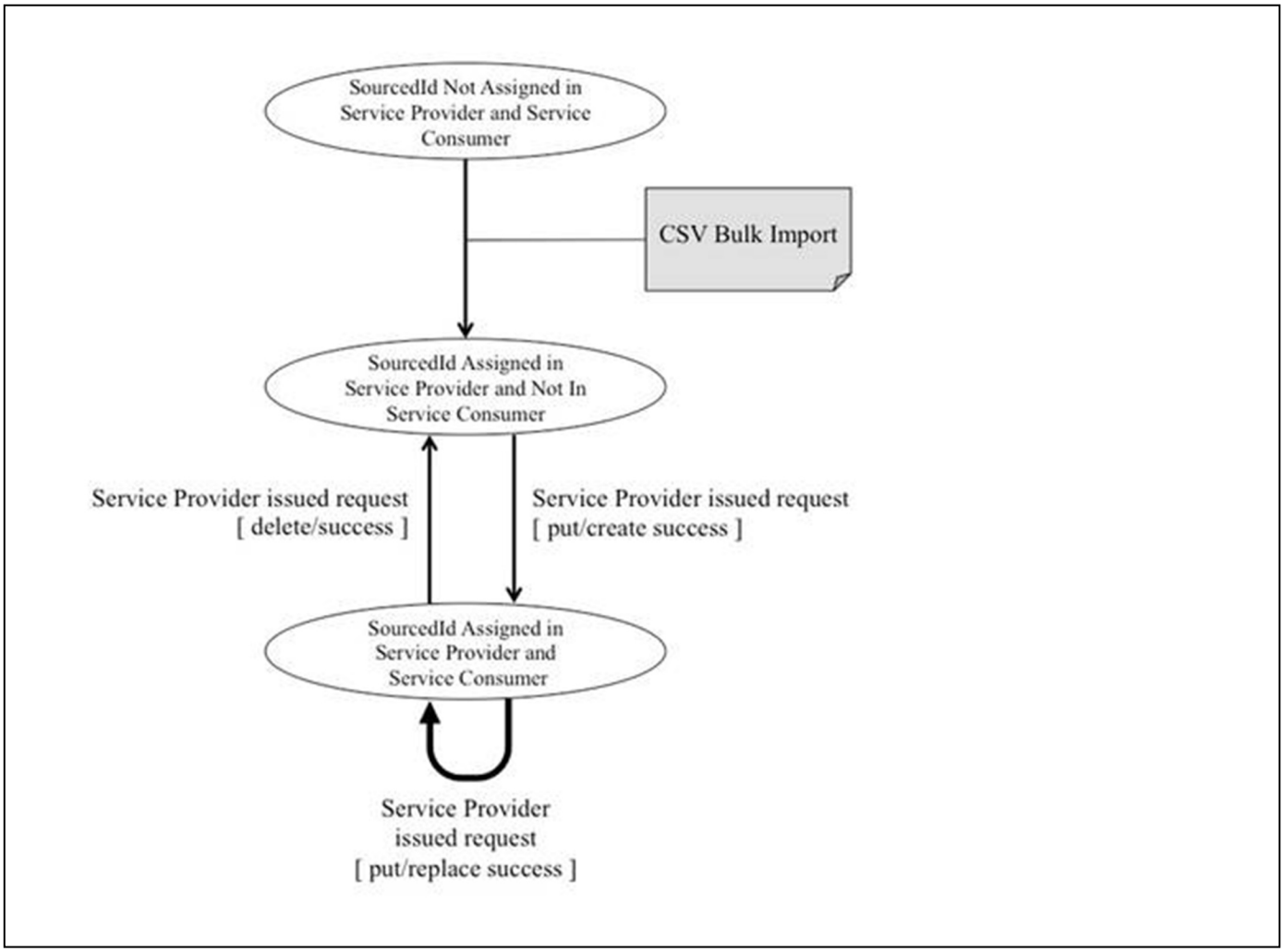


Figure 3.2 - The state diagram for 'push' driven data exchange.

3.3. API Root URL and Version

The API Root URL MUST be /ims/oneroster.

To allow further versions of the specification to exist in a controlled manner, the new version number MUST be '/v1p1'.

Example: https://imsglobal.org/ims/oneroster/v1p1/*

Responses to Requests which are sent to the root URL MUST include an HTML page that contains:

- A list of URLs to the endpoints supported under the root URL;
- A link to the developer documentation (for example, the online version of the specification, a link to online API documentation).

3.4. Binding of Resource Data

JSON IS the binding form to represent the resource data. Section 4 shows the data models that are required to be bound.

Parsers MUST be capable of parsing the JSON data (see Section 5) shown in the UML and example bindings. Parsers MAY ignore any other JSON fields that they encounter, UNLESS those fields are in the extension space (see later).

3.4.1. Pagination

It is expected that Student Information Systems may well contain massive amounts of data, and as such there is a real danger of data overload if entire collections are requested. To avoid this, implementations MUST adopt a pagination mechanism.

Pagination is controlled via two parameters that are appended to the request.

- Limit - the number of results to return
- Offset - the index of the first record to return. (zero indexed)

The default value for limit MUST be 100.

The default value for offset MUST be 0.

Example: return the first 10 resources in a collection of students:

GET https://imsglobal.org/ims/oneroster/v1p1/students?limit=10

Example: return the second 10 resources in a collection of students:

GET https://imsglobal.org/ims/oneroster/v1p1/students?limit=10&offset=10

It is RECOMMENDED that implementations pass the total resource count in collection back to the requester. This MUST be provided in the custom HTTP header: X-Total-Count.

It is RECOMMENDED that implementers pass back next, previous, first and last links in the HTTP Link Header.

Example: 503 student resources exist in the collection. Pagination is on the second page, in units of 10.

Link:

<https://imsglobal.org/ims/oneroster/v1p1/students?limit=10&offset=20>; rel="next",

<https://imsglobal.org/ims/oneroster/v1p1/students?limit=3&offset=500>; rel="last",

<https://imsglobal.org/ims/oneroster/v1p1/students?limit=10&offset=0>; rel="first",

<https://imsglobal.org/ims/oneroster/v1p1/students?limit=10&offset=0>; rel="prev"

NOTE: Pagination must be supported for ALL endpoints that return a collection.

3.4.2. Sorting

It MUST be possible for collections to be returned in a sorted order. It MUST be possible to sort the collection based on any single data element in the core description of the resource. Sort requests MUST make use of the reserved word "sort" (?sort= *data_field*), and optionally the reserved word orderBy for which:

- *data_field* MUST be used in the request to ask for the collection to be sorted on data field. The form of ordering is implementation dependent if the 'orderBy' attribute is not used;
- *orderBy* MAY be used in the request to ask for the collection to be ordered ascending (*asc*) or descending (*desc*).

Example: To ask for a list of students sorted into ascending familyName order:

GET https://imsglobal.org/ims/oneroster/v1p1/students?sort=familyName&orderBy=asc

Sorting should conform to the use of the Unicode Collation Algorithm [UNICODE, 16] when using the relevant comparisons.

If the consumer requests that the data is to be sorted by a non-existent field, the data is returned in the service provider's default sort order and the server must provide the associated transaction status code information of:

- CodeMajor value is 'success';
- Severity value is 'warning';
- CodeMinor value is 'invalid_sort_field';
- Description should contain the supplied unknown field.

NOTE: Sorting must be supported for ALL endpoints that return a collection.

3.4.3. Filtering

It MUST be possible to filter collections for elements matching a certain criteria.

It MUST be possible to filter collections based on any data element in the core description of the resource.

Filter requests MUST consist of the form:-

?filter=<data field><predicate><value>

or

?filter=<data field><predicate><value><logical><data field><predicate><value>

The data fields that can be used are those present in the class definition being filtered. So for example in "courses", it MUST be possible to filter on: 'sourcedId', 'status', 'dateLastModified', 'title', 'grades', 'subjects', etc.

Predicates MUST be chosen from the following predicates in Table 3.2:

Table 3.2 - List of predicates used for filtering.

Predicate	Representation
Equals	=
Not Equal	!=
Greater Than	>
Greater Than or Equal	>=
Lesser Than	<
Lesser Than or Equal	<=
Contains	~

Values MUST be enclosed within single quotes and they **must** be handled as case insensitive. When the response is returned it is the receiving system that should consider whether or not case-sensitivity is important.

<logical> parameters allow more complex queries to be created. For version 1.1, it is RECOMMENDED that logical operations are limited to " AND " and " OR " (note the surrounding white space at each side) and that there is only one such operator used in any filter i.e. a single 'AND' or a single 'OR' in the filter. A single white space must occur before and after the parameter.

To query on the properties of nested objects, (for example, with metadata properties), a dot-notation approach MUST be used. The format for this is:

<NestedObject>.<Property>

Note then when querying on metadata, the property is loosely typed.

Example: To find a student with an Identifier of ND5848416:

> https://imsglobal.org/ims/oneroster/v1p1/students?filter=identifier='ND5848416'
>
> encoded: https://imsglobal.org/ims/oneroster/v1p1/students?filter=identifier%3D%27ND5848416%27

Filter queries MUST be URL encoded.

Example: To ask for a list of students with the familyName Jones:

> Query: familyName='jones'
>
> GET https://imsglobal.org/ims/oneroster/v1p1/students?filter=familyName%3D%27jones%27
>
> To ask for a list of students whose familyName is jones and who were last modified after the 1st of January 2015:

Query: familyName='jones' AND dateLastModified>'2015-01-01'

GET https://imsglobal.org/ims/oneroster/v1p1/students?filter=familyName%3D%27jones%27%20AND%20dateLastModified%3E%272015%3D01-01%27

To ask for a list of all classes taught by teacher 123 which were last modified after the 1st of January 2015:

Query: dateLastModified>'2015-01-01'

GET https://imsglobal.org/ims/oneroster/v1p1/teachers/123/classes?filter=dateLastModified%3E%272015%3D01-01%27

When filtering on objects that are arrays the application of the filter depends on the nature of the comparison. So in the case of filtering on the 'subjects' field when the value of the field is "subject1,subject2,subject3" the following filters would return:

- ?filter="subject=subject1" - record not returned;
- ?filter="subject=subject1,subject2" - record not returned;
- ?filter="subject=subject1,subject2,subject3" - record returned;
- ?filter="subject~subject1" - record returned;
- ?filter="subject~subject1,subject2" - record returned;
- ?filter="subject~subject1,subject2,subject3" - record returned.

This means filtering using the '=' has 'AND' semantics and for '~" has 'OR' semantics.

Filtering rules should conform to the use of the Unicode Collation Algorithm [UNICODE, 16] when using the relevant comparisons.

If the consumer requests that data be filtered by a non-existent field, NO data is returned and the server must provide the associated transaction status code information of:

- CodeMajor value is 'failure';
- Severity value is 'error';
- CodeMinor value is 'invalid_filter_field';
- StatusCode value is the corresponding HTTP response code;
- Description should contain the supplied unknown field.

NOTE:

1. **Filtering must be supported for ALL endpoints that return a collection;**
2. **Filtering MUST be supported using any of the data fields that are REQUIRED;**
3. **Filtering MUST be supported using any of the OPTIONAL data fields that are supported by the Service Provider.**

3.4.4. Field Selection

It MUST be possible for requesters to select the range of fields to be returned. By default, all mandatory and optional fields from the core description of the resource MUST be returned. If any fields are specified in the request then the implementation MUST return those fields AND ONLY those fields i.e. the multiplicity rules for an element are overridden. Any Field or fields from the Full Data Model MAY be requested. Any field or fields from the Data Model MAY be requested.

Field selection request MUST make use of the reserved word *fields*. The value of fields is a comma delimited list of the fields to return.

Example: To ask for a list of students retuning only the given name and family name:

GET https://imsglobal.org/ims/oneroster/v1p1/students?fields=givenName,familyName

If the consumer requests that data be selected using non-existent field, ALL data for the record is returned and the server must provide the associated transaction status code information of:

- CodeMajor value is 'success';
- Severity value is 'warning';
- CodeMinor value is 'invalid_selection_field';
- StatusCode value is the corresponding HTTP response code;
- Description should contain the supplied unknown field.

If the consumer requests that data be selected using a blank field the request will be treated as an invalid request. The server must provide the associated transaction status code information of:

- CodeMajor value is 'failure';
- Severity value is 'error';
- CodeMinor value is 'invalid_blank_selection_field';
- StatusCode value is the corresponding HTTP response code.

NOTE: Field Selection must be supported for ALL read endpoints.

3.5. Error Handling

Implementations MUST be able to report the existence of errors that arise when processing the request. Error reporting MUST make use of the following HTTP response codes listed in Table 3.3.

Table 3.3 - HTTP response codes.

Error Code	Description	Notes
200	OK - It was possible to read the collection / resource.	This code MUST also be used to indicate that no resources have been returned e.g, when a filter

Error Code	Description	Notes
		rule has been applied to a request for a collection of records.
201	OK - New resource has been created.	Introduced in version 1.1.
204	OK - The resource was deleted successfully.	Introduced in version 1.1.
400	Bad Request - the Request was invalid and cannot be served.	
401	Unauthorized - the Request requires authorization.	
403	Forbidden - to indicate that the server can be reached and process the request but refuses to take any further action.	Introduced in version 1.1.
404	Not Found - there is no resource behind the URI.	This code MUST NOT be used to denote that no resources have been returned when a collection has been requested e.g. for a filtered request on a collection.
422	Entity cannot be processed - used where the server cannot validate an incoming entity.	Reserved for version 1.*.
429	The server is receiving too many requests. Retry at a later time.	Introduced in version 1.1.
500	Internal Server Error.	Avoid - use only if there is catastrophic error and there is not a more appropriate code.

When a request is made for a collection e.g. getAllTerms() and there are no records to be returned (such a scenario is most likely to occur when a filter query parameter is used) then a HTTP code of 200 MUST be returned (a HTTP code of 404 MUST NOT be returned). A HTTP code of 404 MUST be returned when a request is made for a related collection e.g. getTeachersForSchool(), and where the related object cannot be located (i.e. the School in the given example).

It is RECOMMENDED that implementations also provide more information about errors to requesters in the form of a dedicated error payload. As more than one error can potentially occur, the payload should be ready

to report an array of potential errors:

The error payload follows the prevailing LIS practice of specifying a CodeMajor, Severity, CodeMinor and description.

The LIS specifications list a great number of error payloads. For this specification, a subset is used:

- CodeMajor enumeration: { success | failure };
- Severity enumeration: { status | error | warning };
- CodeMinor enumeration: { full success | unknown object | invalid data | unauthorized | invalid_sort_field | invalid_filter_field | invalid_selection_field };
- Description: A text string providing a human readable description of the error that happened.

Table 3.4 shows how these errors can be applied to the HTTP response codes.

Table 3.4 - Error Payloads from LIS.

Error Code	CodeMajor	Severity	Code Minor
200	success	status	full success
	success	warning	invalid_sort_field
	success	warning	invalid_ selection _field
400	failure	error	invalid data
	failure	error	invalid_filter_field
	failure	error	invalid_ blank_selection _field
401	failure	error	unauthorized
403	failure	error	forbidden
404	failure	error	unknown object
429	failure	error	server_busy
500	failure	error	

It is RECOMMENDED that for successful requests, no error payload is returned; the HTTP status code should be enough.

3.6. Security [R19]

OneRoster 1.1 requires the use OAuth 2 Bearer Tokens as defined in RFC 6750 for authorization with Transaction Layer Security (TLS) for message encryption. The use of OAuth 1.0a message signing (as used in OneRoster 1.0 and permitted in earlier releases of OneRoster 1.1) has been deprecated and removed from the specification documentation.

3.6.1. Transport Security

As the service will be exposing personal data related to students and their grades, it is important that only authorized users have access to that data. Further, data exchanges should be encrypted to ensure that packet sniffing cannot be used to read the data in transit.

- All Requests and Responses MUST be sent using Transaction Layer Security (TLS). Exchange of the signed certificates for endpoints between clients and servers is beyond the scope of this specification. Implementers of clients and servers are advised to look at the various 3rd party certificate signing services in order to obtain signed certificates. Support for TLS 1.2 is REQUIRED and use of SSL is now PROHIBITED.

3.6.2. OAuth 2 Bearer Token Authorization

OneRoster 1.1 REQUIRES the use of OAuth 2.0 Bearer Tokens (Client Credentials) with the access token obtained using the mechanism described in [RFC 6749] (https://tools.ietf.org/html/rfc6749#section-4.4). The use of OAuth 2.0 for 1EdTech web services is defined in the document 1EdTech Security Framework 1.0 (https://www.imsglobal.org/spec/security/v1p0/).

Authorization will use the OAuth 2.0 'Client Credentials Grant' mechanism. In this mechanism the client can request an access token using only its client credentials (using the consumer key and secret information currently required in OneRoster's OAuth 1.0a usage) when the client is requesting access to the protected resources under its control, or those of another resource owner that have been previously arranged with the authorization server. In this approach the client issues a client authentication request and receives in response an access token. Issuing of an access token is defined in Section 5 of [RFC 6749]. The access token request MUST include the set of 'scopes' being requested and the response MUST include the 'scopes' being authorized.

The request for an access token takes the form of (this uses TLS):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&scope=scopename1%20scopename2%20scopenamex
```

The authorization encoding is produced using the consumer key and secret.

Success results in the granting of the access token with a response of:

```
HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

    "access_token" : "2YotnFZFEjr1zCsicMWpAA",

    "token_type" : "bearer",
```

```
    "expires_in" : 3600,
    "scope" : "scopename1 scopename2 scopenamex"
}
```

The recommended default value for the 'expires_in' is 3600s

The client utilizes the access token to authenticate with the resource using the HTTP "Authorization" request header field [RFC 2617] with an authentication scheme defined by the specification of the access token type used, such as [RFC 6750]. An example of the use of the bearer token is:

```
GET /resource/1 HTTP/1.1
Host: provider.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

NOTE: This exchange assumes that TLS is used to secure the links.

The set of scopes available in OneRoster 1.1 are:

- **https://purl.imsglobal.org/spec/or/v1p1/scope/roster-core.readonly** - enable access to the getAcademicSession, getClass, getCourse, getEnrollment, getGradingPeriod, getOrg, getSchool, getStudent, getTeacher, getUser, getAllAcademicSessions, getAllClasses, getAllCourses, getAllEnrollments, getAllGradingPeriods, getAllOrgs, getAllSchools, getAllStudents, getAllTeachers and getAllUsers rostering endpoints;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/roster.readonly** - enable access to ALL of the rostering endpoints EXCEPT getDemographic and getDemographics;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/roster-demographics.readonly** - enable access to the getDemographics and getAllDemographics rostering endpoints;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/resource.readonly** - enable access to the getResource, getAllResources, ghetResourcesForClass and getResourcesForCourse resources endpoints;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/gradebook.readonly** - enable access to the getCategory, getAllCategories, getLineItem, getAllLineItems, getResult, getAllResults, getLineItemsForClass, getResultsForClass, getResultsForLineItemForClass and getResultsForStudentForClass gradebook endpoints;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/gradebook.createput** - enable access to the putCategory, putLineItem and putResult gradebook endpoints;
- **https://purl.imsglobal.org/spec/or/v1p1/scope/gradebook.delete** - enable access to the deleteCategory, deleteLineItem and deleteResult gradebook endpoints.

3.7. Serialization Format [R9]

For version 1.0/1.1, JSON data MUST be supported as the binding (this is the only binding for which there is conformance certification). Clients MAY ask for other bindings, but implementers are not obliged to provide them, and a 4XX response is valid. It is RECOMMENDED therefore to use the HTTP header field; Accept, with a value of "application/json".

Implementers MUST use the HTTP header field: Content-Type, with a value of "application/json", to inform requesters that results will be returned in JSON.

All properties that have a multiplicity of many MUST be sent as a JSON array even when there is only one value to be sent.

NULL and EMPTY fields MUST NOT occur within a JSON payload (note this is NOT dependent on the multiplicity of the field). In many cases, a NULL/EMPTY value is a data-type violation and would be declared as such when using JSON Schema Validation. For example, the following payload for a 'getCourse()' request is invalid:

```
{
    "course" : {
        "sourcedId": "<sourcedId of the course>",
        "status" : "active",
        "dateLastModified" : "2012-04-23T18:25:43.511Z",
        "title" : "Organic Chemistry",
        "courseCode" :,
        "org" : {
            "href": "<href of the org related to this course>",
            "sourcedId": "<sourcedId of the org related to this course>",
            "type" : "org"
        }
    }
}
```

The 'courseCode' value is NULL. Instead the payload MUST become:-

```
{
    "course" : {
        "sourcedId": "<sourcedId of the course>",
        "status" : "active",
        "dateLastModified" : "2012-04-23T18:25:43.511Z",
        "title" : "Organic Chemistry",
        "org" : {
            "href": "<href of the org related to this course>",
            "sourcedId": "<sourcedId of the org related to this course>",
            "type" : "org"
        }
    }
}
```

By extension this means that the following payload is also PROHIBITED:-

```
{
    "course" : {
        "sourcedId": "<sourcedId of the course>",
        "status" : "active",
        "dateLastModified" : "2012-04-23T18:25:43.511Z",
        "title" : "Organic Chemistry",
        "org" : {}
    }
}
```

The 'org' value is NULL. The following payload for a 'getAcademicSession()' call is also PROHIBITED:-

```
{
    “academicSession”: {
        “sourcedId” : “<sourcedid of this academicSession (term)>”,
        “status” : “active”,
        “dateLastModified” : “2012-04-23T18:25:43.511Z”,
        “title” : “Summer”,
        “startDate” : “2012-04-24T00:00:00.000Z”,
        “endDate” : “2012-06-30T00:00:00.000Z”,
        “type” : “term”,
        “parent” : {
            “href” : “<href of the parent for this academic session>”,
            “sourcedId” : “<sourcedId of the  parent for this session>”,
            “type” : “academicSession”,
        },
        “schoolyear” : “2015”,
        “children” : []
    }
}
```


The ‘children’ value is NULL.

3.8. Extensions [R36]

It is recognized that implementers may wish to extend the specification. The preferred mechanism for doing this is for implementers to use an extension space within the OneRoster data model, and then set their parsers to read those extension attributes (see Subsection 4.1 for the definition of this Metadata extension structure).

4. Data Model

The logical data model is shown in Figure 4.1, with the following scenario: A school teaches over a number of terms. A school teaches a number of courses, employs a number of teachers, and educates a number of students.

A class is an instance of a course that is taught in a particular term. A course can consist of a number of classes. A class is assessed via a number of line items (columns in a gradebook), and a line item can be categorized. Students taking a class are assessed by grading; a lineItem will have zero or more results, but usually only one result per student. A more detailed representation is given in Appendix B. Classes and courses can identify the set of resources that are required in the delivery of the learning experience.

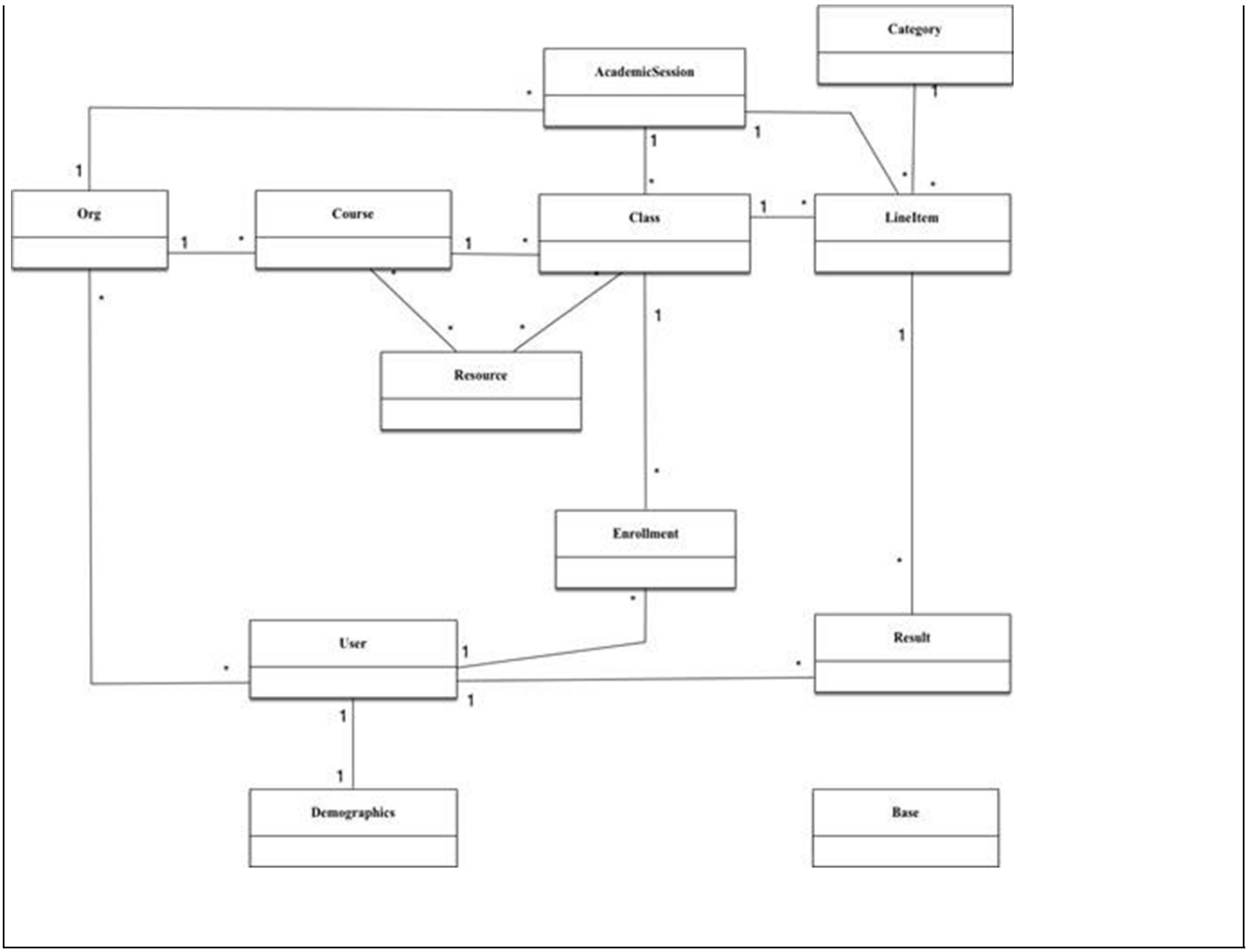


Figure 4.1 - Logical Data Model.

Notes:

a) To avoid confusion, if no context is given, then the word "Grade" means the education level of a class or course (e.g. "1st Grade, 9th Grade etc.). Result is the word used to mean educational achievement (e.g. a Grade A, or 78%). The assessment of assignments (represented by a line item) would yield results;

b) The support of ALL string-based data-types requires that a maximum length of at least 255 must be supported by implementations. If string lengths of greater than 255 are used then systems may truncate the string without failing conformance.

4.1. The Base Class [R35, R40]

All classes in the REST binding descend from the base class. This class defines that all objects must share some common properties:

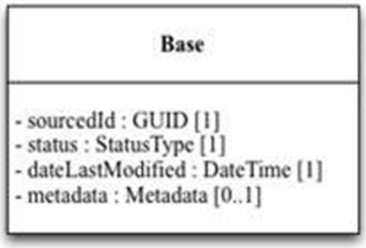


Figure 4.2 - Base Class with Status Enumeration.

- SourcedId - all Objects MUST be identified by a Source Identifier. This is a GUID[1] System ID for an object. This is the GUID that SYSTEMS will refer to when making API calls, or when needing to identify an object. It is RECOMMENDED that systems are able to map whichever local ids (e.g. database key fields) they use to SourcedId. The sourcedId of an object is considered an addressable property of an entity and as such will not be treated as Personally Identifiable Information (PII) by certified products. Therefore, as a part of certification, vendors will be required to declare that they will notify customers via documentation or other formal and documented agreement that sourcedIds should never contain PII in general, but particularly users. This means that if a customer includes a student name in an enrollment.sourcedId, it will not fall to any certified product to protect the enrollment.sourcedId as PII, or even the userSourcedId field in the enrollment record;
- Status - all objects MUST BE either "active" or "tobedeleted". Something which is flagged "tobedeleted" is to be considered safe to delete. Systems can delete records that are flagged as such if they wish, but they are not under any compulsion to do so. In v1.1 the enumeration value of 'inactive' was removed and so for backwards compatibility all such marked objects should be interpreted as 'tobedeleted';
- DateLastModified - all objects MUST be annotated with the dateTime upon which they were last modified. This enables requesters to query for just the latest objects. DateTimes MUST be expressed in W3C profile of ISO 8601, MUST have a resolution of milliseconds and MUST contain the UTC timezone;
- Metadata - all objects CAN be extended using the Metadata class. This specification is silent on what implementers may consider to be appropriate extensions.

Table 4.1 - Data Elements for Base Class

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	For example: "boarding" : "false"

Dates:

DateTimes MUST be expressed using ISO 8601 format: http://tools.ietf.org/html/rfc3339. This is JavaScript's prevailing data format:

YYYY-MM-DDTHH:MM:SS.mmmZ

An example of which:

2012-04-23T18:25:43.511Z

The Z denotes the UTC timezone.

Dates MUST be expressed using ISO 8601 format: http://tools.ietf.org/html/rfc3339. This is JavaScript's prevailing data format:

YYYY-MM-DD

An example of which:

2012-04-23

4.1.1. Extensions

All data model extensions must be contained within the Metadata class. These extensions take the form of name/value pairs. The name is the label of the extension field, and the value is the value of the extension. For example, if wanting to show the extension of field "classification", with value "private" that was added/provided by "ims", the name/value pair is: "ims.classification":"private" (see subsection 5.13 for an example in JSON).

It is RECOMMENDED that where extensions are used, whenever possible the name/value pairs are based upon vocabulary controlled files. This eliminates a proliferation of free text equivalencies from entering the data (e.g. "Florida" vs "FL", vs "Florida, USA". In such cases either the attribute, or the value (or both) MUST be a URI that references the attribute and/or value from an appropriate vocabulary file. For example:

"http://www.nbrs.org" : "FL"

4.2. AcademicSession [R1, R12, R25]

AcademicSession represent durations of time. Typically they are used to describe terms, grading periods, and other durations e.g. school years. Term is used to describe a period of time during which learning will take place. Other words for term could be in common use around the world e.g. Semester. The important thing is that Term is a unit of time, often many weeks long, into which classes are scheduled. Grading Period is used to represent another unit of time, that within which line items are assessed. A term may have many grading periods, a grading period belongs to a single term. A class may be assessed over several grade periods (represented by a line item being connected to a grading period). The parent / child attributes of academic sessions allow terms to be connected to their grading periods and vice-versa.

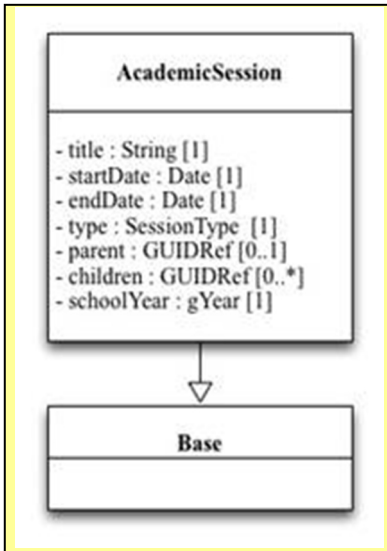


Figure 4.3 - Proposed Term Data Model

Table 4.2 - Data Elements for Academic Session.

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	
title	1	For example: Spring Term
startDate	1	For example: 2012-01-01
endDate	1	For example: 2012-04-30
type	1	See subsection 4.13.7 for the enumeration list.
parent	0..1	Link to parent AcademicSession i.e. an AcademicSession 'sourcedId'.
children	0..*	Links to children AcademicSession i.e. an AcademicSession 'sourcedId'.
schoolYear	1	The school year for the academic session. This year should include the school year end e.g. 2014.

4.3. Class [R16, R21, R23, R30, R38, R39]

A class is an instance of a course, onto which students and teachers are enrolled. A class is typically held within a term.

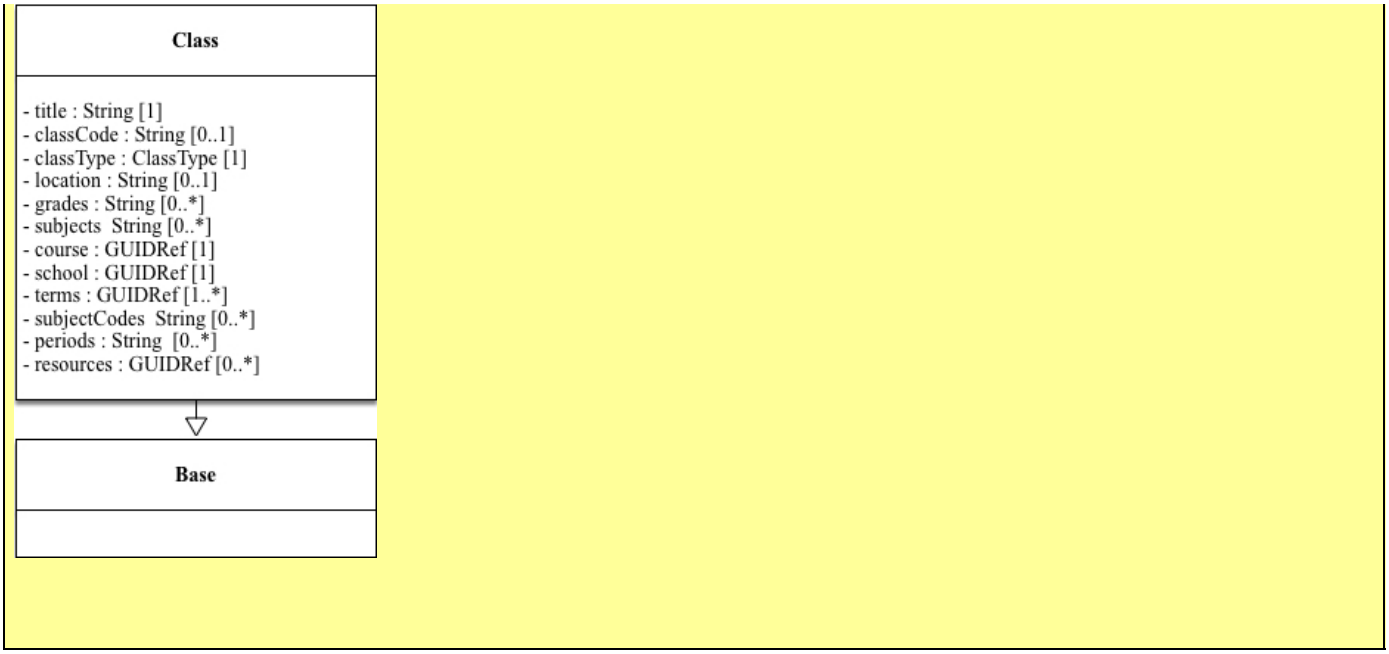


Figure 4.4 - Class Data Model

Table 4.3 - Data Elements for Classes

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	
title	1	For example: Basic Chemistry
classCode	0..1	For example: Chem101-Mr Rogers
classType	1	See subsection 4.13.1 for the enumeration list.
location	0..1	For example: "room 19"
grades	0..*	Grade(s) for which the class is attended. The permitted vocabulary is from CEDS (Version 5): https://ceds.ed.gov/ and the 'Entry Grade Level' element https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=7100. Example: 09 or an array of 09,10 and 11.
subjects	0..*	Subject name(s). Example: "chemistry"
course	1	Link to course i.e. the Course 'sourcedId'.

school	1	Link to school i.e. the School 'sourcedId'.
terms	1..*	Links to terms or semesters (academicSession) i.e. the set of 'sourcedIds' for the terms within the associated school year.
subjectCodes	0..*	This is a machine-readable set of codes and the number should match the associated 'subjects' attribute. For systems deployed in the USA this vocabulary SHOULD be a School Courses for the Exchange of Data (SCED) code: http://nces.ed.gov/forum/SCED.asp.
periods	0..*	The time slots in the day that the class will be given. This MUST always be sent as a JSON array. Examples: • "periods" : ["1"]; • "periods" : ["1", "2", "4"];
resources	0..*	Link to resources i.e. the Resource 'sourcedId'.

4.4. Course [R15, R22, R24, R29]

A Course is a course of study that, typically, has a shared curriculum although it may be taught to different students by different teachers. It is likely that several classes of a single course may be taught in a term. For example, a school runs Grade 9 English in the spring term. There are four classes, each with a different 30 students, taught by 4 different teachers. However the curriculum for each of those four classes is the same - the course curriculum.

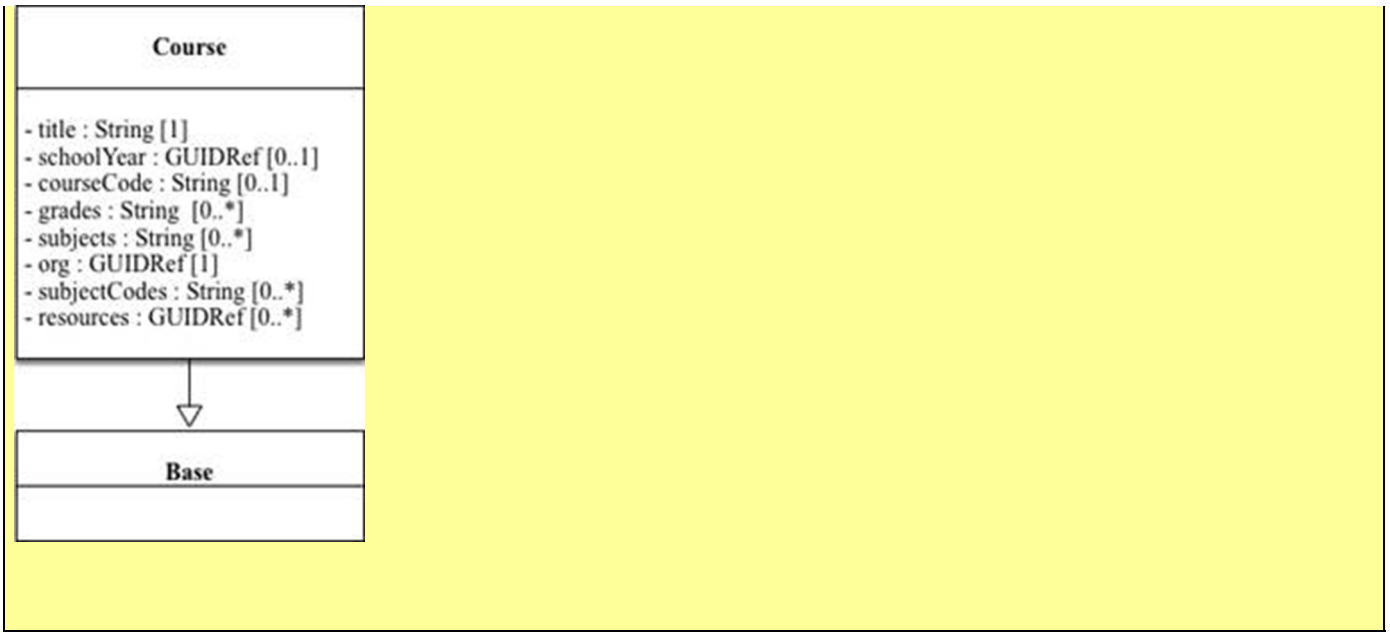


Figure 4.5 - Course Data Model.

Table 4.4 - Data Elements for Courses

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	2012-04-23T18:25:43.511Z
metadata	0..1	
title	1	For example: Basic Chemistry
schoolYear	0..1	Link to academicSessioni.e. the AcademicSession 'sourcedId'.
courseCode	0..1	For example: CHEM101
grades	0..*	Grade(s) for which the class is attended. The permitted vocabulary is from CEDS (Version 5): https://ceds.ed.gov/ and the 'Entry Grade Level' element https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=7100. Example: 09 or an array of 09,10 and 11.
subjects	0..*	This is a human readable string. Example: "chemistry".
org	1	Link to org i.e. the 'sourcedId' of the org.
subjectCodes	0..*	This is a machine-readable set of codes and the number should match the associated 'subjects' attribute.

		For systems deployed in the USA this vocabulary SHOULD be a School Courses for the Exchange of Data (SCED) code: http://nces.ed.gov/forum/SCED.asp.
resources	0..*	Link to resources if applicable i.e. the 'sourcedIds'.

4.5. Demographic Data [R42]

Demographics information is taken from the Common Educational Data Standards from the US government. (http://ceds.ed.gov). Demographics are OPTIONAL.

Note that demographics data is held in its own service, and that access to this service is considered privileged. Not all consumer keys will be able to request demographics data.

Demographic Data is modeled in LIS, but the sort of demographic data required by K12 is very different to that modeled in LIS. For this reason, new structures have been created.

The 'sourcedId' of the demographics MUST be the same as the 'sourcedId' of the user to which it refers.

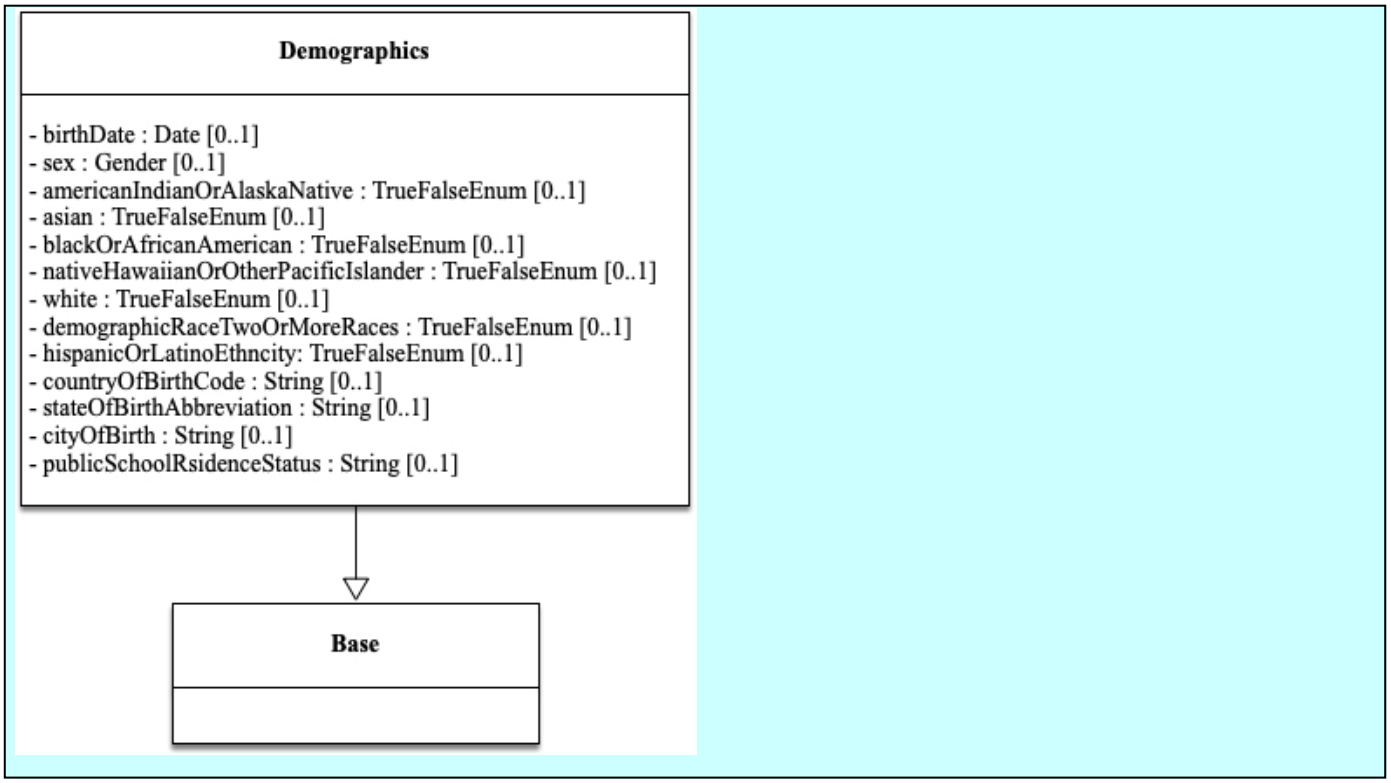


Figure 4.6 Data Elements for Demographics

Table 4.5 - Data Elements for Demographics

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	2012-04-23T18:25:43.511Z
metadata	0..1	
birthDate	0..1	For example: 1908-04-01.
sex	0..1	See subsection 4.13.2 for the enumeration list.
americanIndianOrAlaskaNative	0..1	Enumeration. Permitted values: ("true" | "false")
asian	0..1	Enumeration. Permitted values: ("true" | "false")
blackOrAfricanAmerican	0..1	Enumeration. Permitted values: ("true" | "false")
nativeHawaiianOrOtherPacificIslander	0..1	Enumeration. Permitted values:("true" | "false")
white	0..1	Enumeration. Permitted values: ("true" | "false")
demographicRaceTwoOrMoreRaces	0..1	Enumeration. Permitted values: ("true" | "false")
hispanicOrLatinoEthnicity	0..1	Enumeration. Permitted values: ("true" | "false")
countryOfBirthCode	0..1	Vocabulary - https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20002
stateOfBirthAbbreviation	0..1	Vocabulary - https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20837
cityOfBirth	0..1	String
publicSchoolResidenceStatus	0..1	Vocabulary - https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20863

4.6. Enrollments [R43]

An enrollment is the name given to an individual taking part in a course or class. In the vast majority of cases, users will be students learning in a class, or teachers teaching the class. Other roles are also possible.

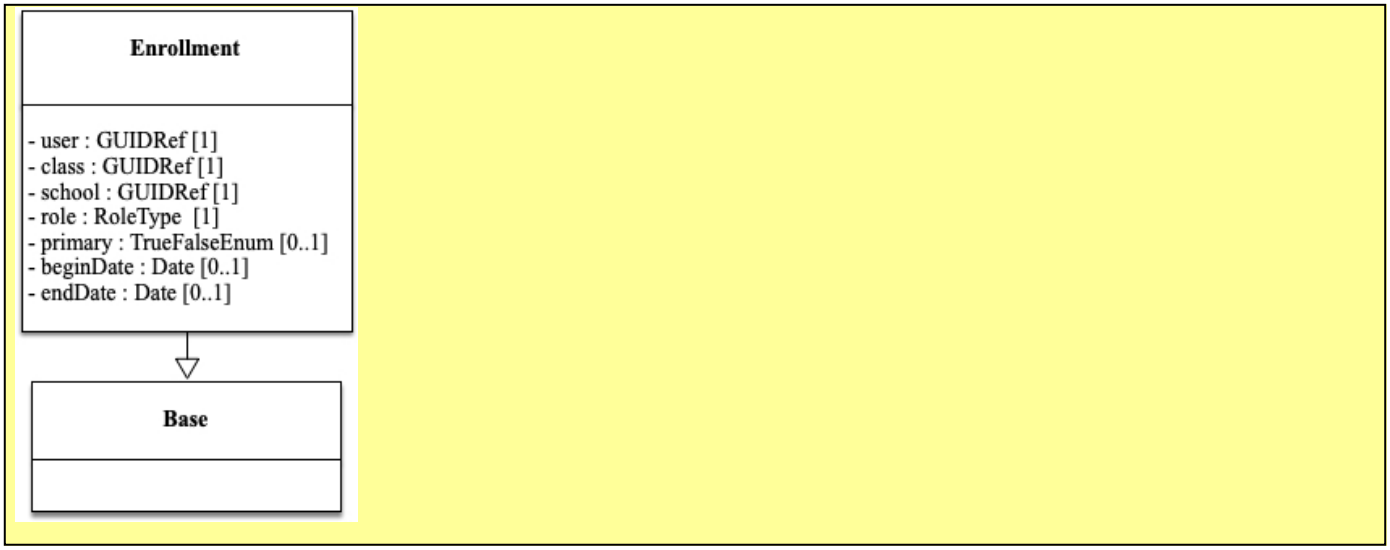


Figure 4.7 - Enrollment Data model

Table 4.6 - Data Elements for Enrollments.

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	
user	1	Link to the enrolled User i.e. a User 'sourcedId'
class	1	Link to the class on which the user is enrolled i.e. a Class 'sourcedId'
school	1	Link to the school at which the class is being provided i.e. an Org 'sourcedId'
role	1	See subsection 4.13.5 for the enumeration list. The ONLY permitted values are: { administrator | proctor | student | teacher}.
primary	0..1	Enumeration. Permitted values: ("true" | "false"). Applicable only to teachers. Only one teacher should be designated as the primary teacher for a class in the period defined by the begin/end dates.
beginDate	0..1	The start date for the enrollment (inclusive). This date must be within the period of the associated Academic Session for the class (Term/Semester/SchoolYear). Example: 2012-04-23
endDate	0..1	The end date for the enrollment (exclusive). This date must be within the period of the associated Academic

		Session for the class (Term/Semester/SchoolYear). Example: 2013-03-31

4.7. Line Items [R6, R7, R20]

Line Items are assignments in LIS, they make up the column headings in a gradebook, and many students will each complete lineItems as they are assessed. It is proposed that the Line Item object take a subset of the elements used in LIS, and adds a few more which are relevant to K12.

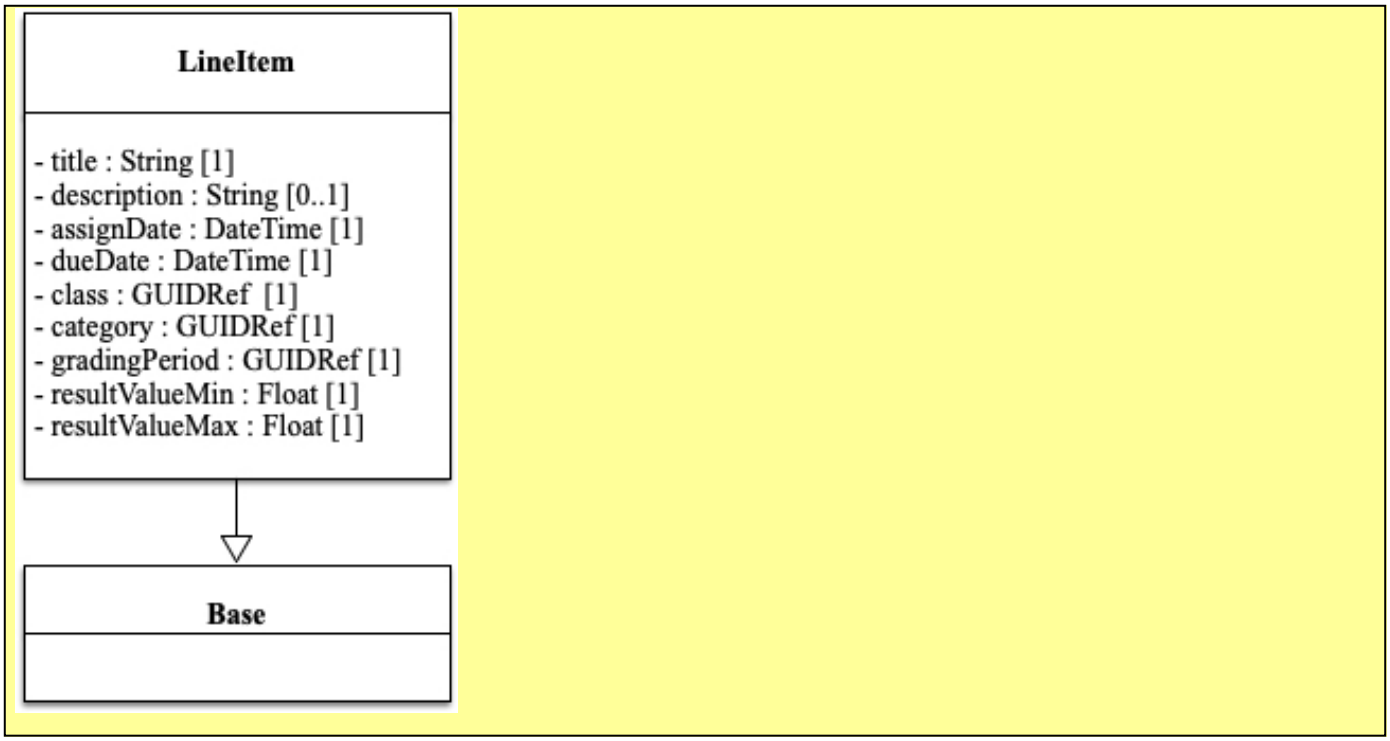


Figure 4.8 - LineItem (assignment) Data Model

Table 4.7 - Data Elements for LineItems.

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	2012-04-23T18:25:43.511Z
metadata	0..1	
title	1	For example: Maths Test 1
description	0..1	For example: Simple addition test
assignDate	1	For example: 2012-01-01T18:25:43.511Z
dueDate	1	For example: 2012-01-05T18:25:43.511Z

class	1	Link to class i.e. the class 'sourcedId'.
category	1	Link to item category i.e. the Line Item Category 'sourcedId'.
gradingPeriod	1	Link to grading period i.e. the AcademicSession 'sourcedId'
resultValueMin	1	A floating point number defining (inclusive) the minimum value for the result. For example: 0.0.
resultValueMax	1	A floating point number defining (inclusive) the maximum value for the result. For example: 10.0

4.8. Line Item Categories [R3, R4]

A Category is the name given to a grouping of line items. (Line Items being equivalent to assignments which students will complete). Examples of categories include "homework", "quizzes" or "essays". It is proposed that the Category object be defined as shown in Figure 4.9/Table 4.8.

Figure 4.9 - Category Data Model

Table 4.8 - Data Elements for Categories

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	

title	1	For example: Homework

4.9. Org [R21, R40]

ORG is defined here as a structure for holding organizational information. An ORG might be a school, or it might be a local, statewide, or national entity. ORGs will typically have a parent ORG (up to the national level), and children, allowing a hierarchy to be established.

School is defined here as the place where the learning happens. Most commonly this is the data that describes a bricks and mortar building, or, in the case of a virtual school, the virtual school organization. For enrollment and result reporting purposes, little information about this organization is required. Later versions of the specification could add further information, such as an address, for example. A common example of a local organization is a school district.

Note that although School is a type of org, the default entry point for requests in most places will be a school. The API provides many school based entry points, whilst still allowing for more generic reading of ORGs, for those applications that need to.

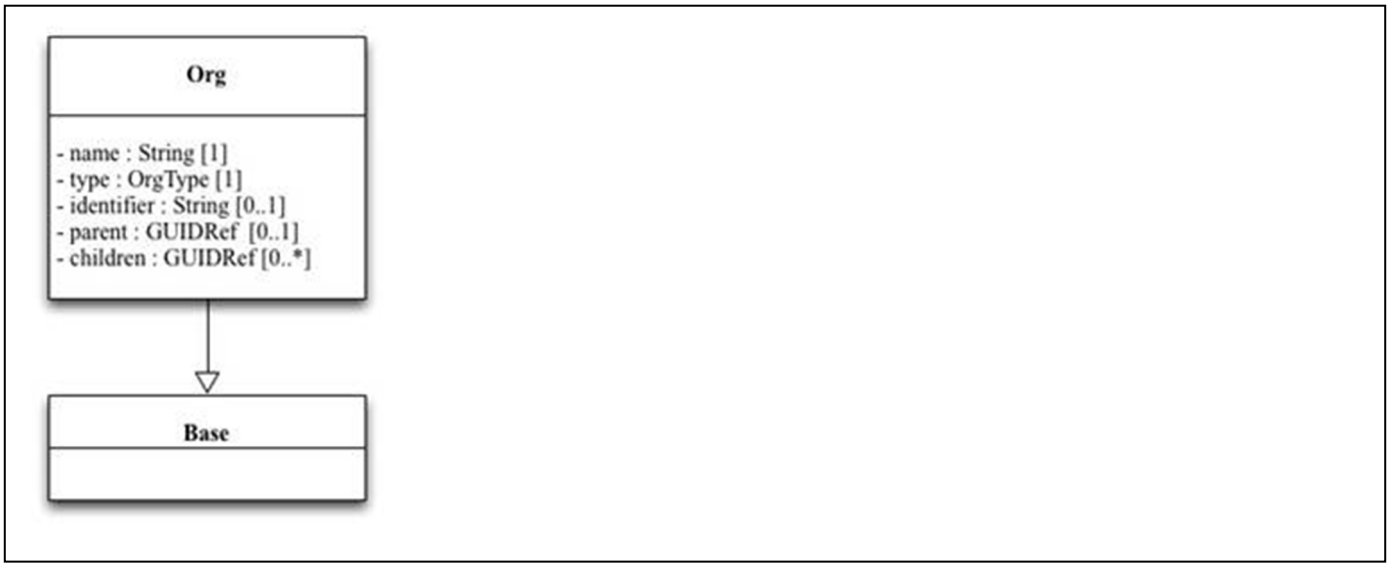


Figure 4.10 - Org Data Model.

Table 4.9 - Data Elements for Organizations.

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	

name	1	For example: 1EdTech High
type	1	See Subsection 4.13.4 for enumeration list.
identifier	0..1	Human readable identifier for this org (e.g. NCES ID).
parent	0..1	Link to Org i.e. the parent Org 'sourcedId'
children	0..*	Link to Org i.e. the child Org 'sourcedId'.

4.10. Resources [R46, R47, R48]

A resource is a description of learning content that is related to a course and/or a class. This identifies a resource that is used by a teacher, learner, etc. as part of the learning experience. A resource MUST be associated to a course and/or a class.

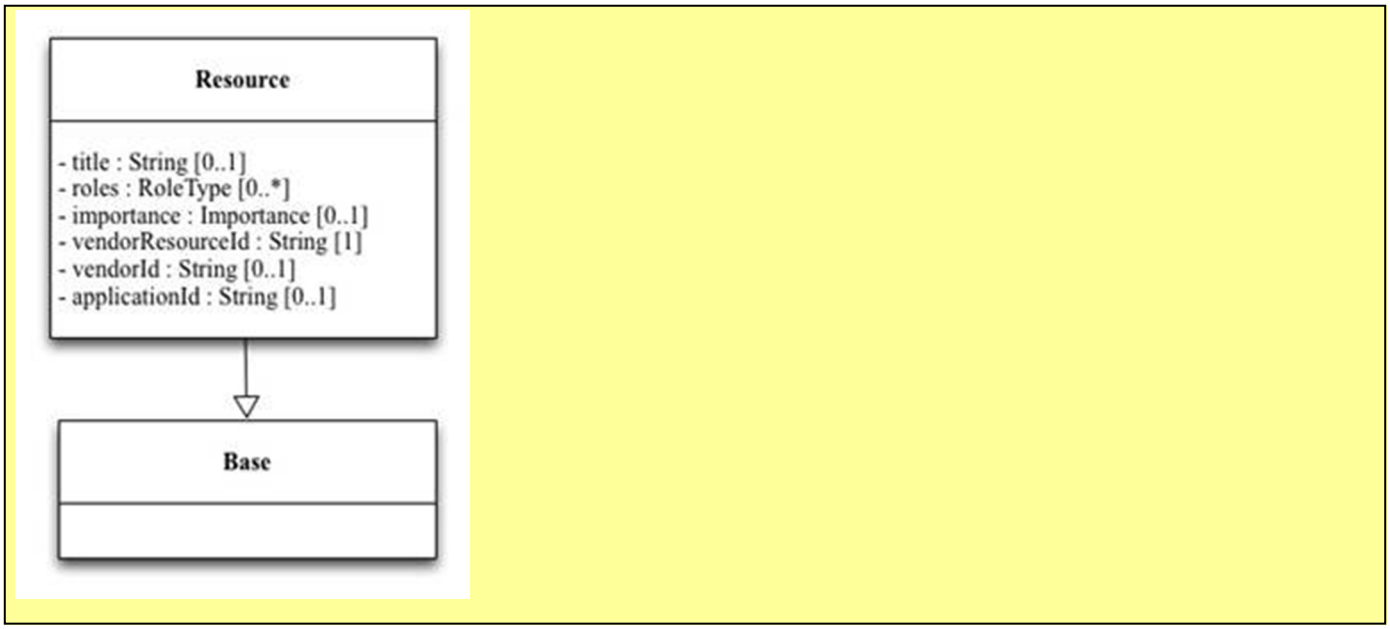


Figure 4.11 - Resource Data Model

Table 4.10 - Data Elements for Resources

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	
title	0..1	For example: Organic Chemistry

roles	0..*	The set of roles. See subsection 4.13.5 for the enumeration list.
importance	0..1	See subsection 4.13.3 for the enumeration list.
vendorResourceId	1	Unique identifier for the resource allocated by the vendor.
vendorId	0..1	Identifier for the vendor who created the resource. This will be assigned by 1EdTech as part of Conformance Certification.
applicationId	0..1	Identifier for the application associated with the resource.

4.11. Results [R5, R27, R28]

It is proposed that the OneRoster Result object takes a subset of the equivalent LIS elements as shown in Figure 4.12/Table 4.11.

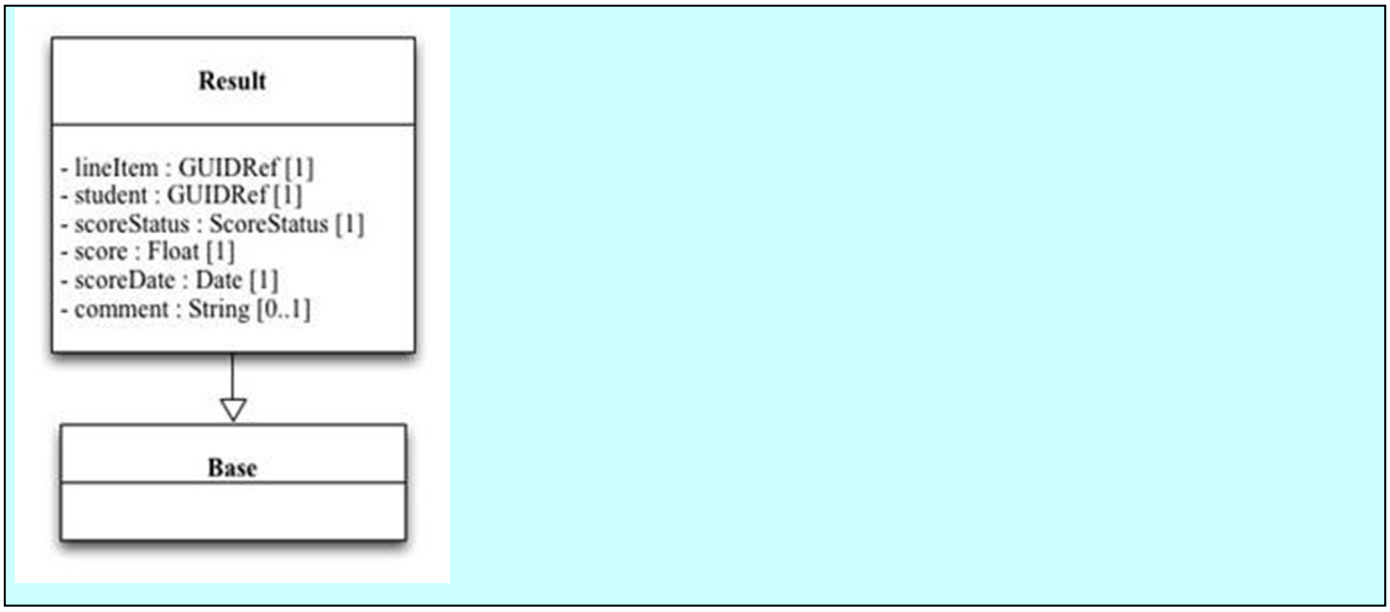


Figure 4.12 - Result Data Model.

Table 4.11 - Data Elements for Results

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z

metadata	0..1	
lineItem	1	Link to lineItem i.e. the lineItem's 'sourcedId'.
student	1	Link to student i.e. the user's 'sourcedId'.
scoreStatus	1	See subsection 4.13.6 for the enumeration list.
score	1	For example: 67.0
scoreDate	1	For example: 2012-01-05
comment	0..1	For example: excellent

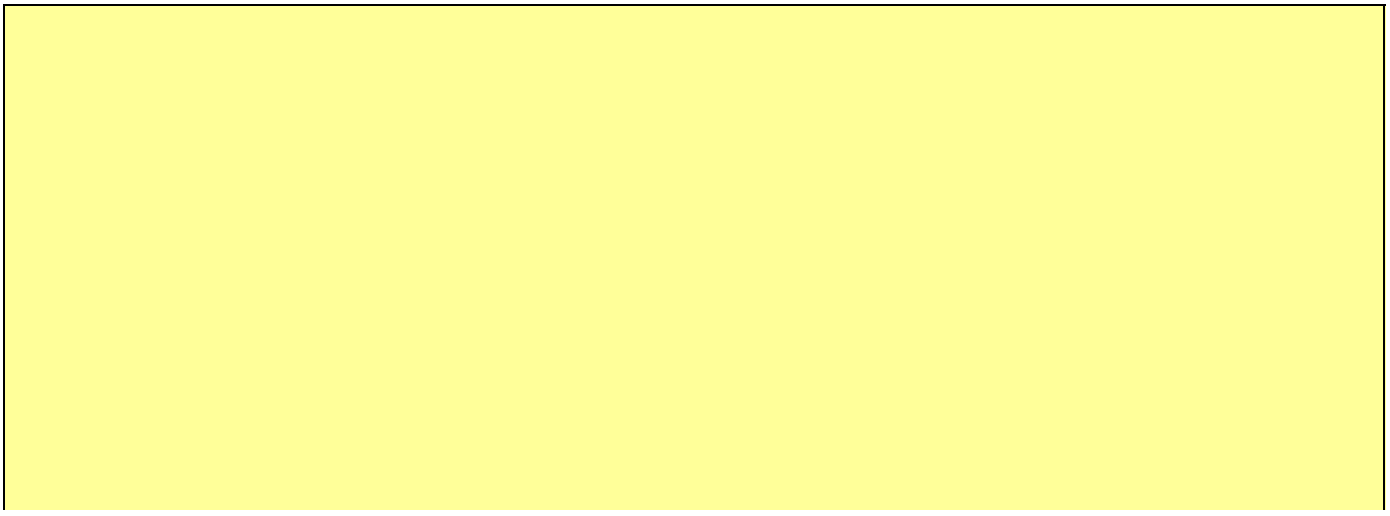
4.12. Users, Students, Teachers [R2, R14, R17, R31, R32, R33, R38, R41, R42]

Users, Teachers and Students are human beings that are teaching or studying in a class respectively. LIS represents these with Person. For the case of binding, it is proposed that a single User class is used to represent both teachers and students, and that a role element be used to distinguish a user's natural role. In the rest binding to follow, it is possible to select teachers and students within a school, course or class. In LIS, users have an "institution role" set within the person record to identify their (primary) role.

Note that this requirement is expanded to introduce other types of human: parents, guardians, relatives and aides.

Humans may have relationships with other humans. For example, a student may have parents. The "agents" attribute allows for relationships between humans to be expressed. Note that these are typically from the point of view of the student - so a student will link to its parents (via the agent attribute). The reverse view MUST also be modeled, so for example, a user of role "parent" MUST have agents that are of type "student".

Note: Teachers MUST NOT be set as agents of students - the teaching relationship is covered via enrollments.



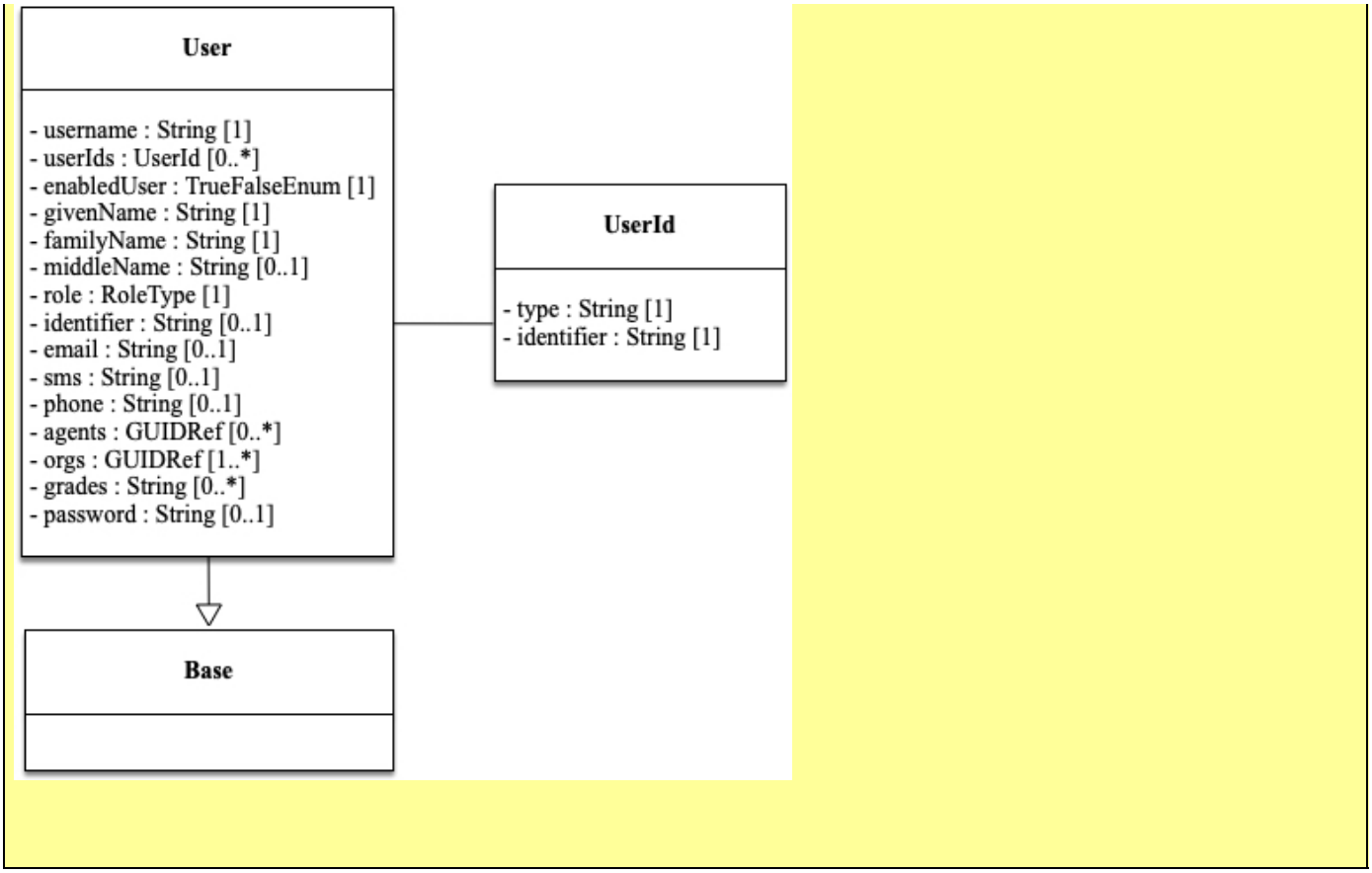


Figure 4.13 - User Data Model.

Table 4.12 - Data Elements for Users

Data Element Name	Multiplicity	Notes
sourcedId	1	For example: 9877728989-ABF-0001
status	1	See subsection 4.13.8 for the enumeration list.
dateLastModified	1	For example: 2012-04-23T18:25:43.511Z
metadata	0..1	
username	1	For example: pjn@imsglobal.org
userIds	0..*	This is the set of external user identifiers that should be used for this user, if for some reason the sourcedId cannot be used. This might be an active directory id, an LTI id, or some other machine-readable identifier that is used for this person.
type	1	For example: LDAP
identifier	1	For example: 9877728989-ABF-0001

enabledUser	1	Enumeration. Permitted values: ("true" | "false"). 'false' denotes that the record is active but system access is curtailed according to the local administration rules. This field is used to determine whether or not the record is active in the local system. NOTE: This field was added as part of the rationalization of the 'status' field and the removal of the enumerated value of 'inactive'.
givenName	1	For example: Phil
familyName	1	For example: Nicholls
middleName	0..1	If more than one middle name is needed separate using a space "Wingarde Granville"
role	1	See subsection 4.13.6 for the enumeration list.
identifier	0..1	For example: 9898-PJN
email	0..1	For example: pjn@imsglobal.org
sms	0..1	For example: +44 07759 555 922
phone	0..1	For example: +44 07759 555 922
agents	0..*	Links to other people i.e. a User 'sourcedId'
orgs	1..*	Links to orgs. In most cases, this is a single link to a school, but could be to a district or national org. People might also be linked to multiple organizations.
grades	0..*	Grade(s) for which a user with role 'student' is enrolled. The permitted vocabulary is from CEDS (Version 5): https://ceds.ed.gov/ and the 'Entry Grade Level' element https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=7100.
password	0..1	For example: Xwyz//123

4.13. Enumerated Vocabularies

4.13.1. ClassType

The set of permitted tokens for the type of class are listed below.

Token	Description
homeroom	The homeroom (form) assigned to the class.
scheduled	The class as assigned in the timetable.

4.13.2. Gender

The set of permitted tokens for the type of gender are listed below.

Token	Description
male	Gender of Male.
female	Gender of Female.

4.13.3. Importance

The set of permitted tokens for the importance are listed below.

Token	Description
primary	A resource of primary usage.
secondary	A resource of secondary usage/significance.

4.13.4. OrgType

The set of permitted tokens for the type of organization are listed below.

Token	Description
department	Denotes a department. A department may be a subset in a school or a set of schools. Added in V1.1.
school	Denotes a school. This is the unit of assignment for classes and enrollments.
district	Denotes a school district. Added in V1.1.
local	V1.0 instances will use this value to identify districts.
state	Denotes a state level organization.
national	Denotes a national level organization.

The explicit hierarchy is: national -> state -> local -> district -> school.

Note that a 'department' may be inserted below any entity other than national and above any entity other than national and state i.e. national -> state-> department -> local -> department -> district -> department -> school -> department.

4.13.5. RoleType

The set of permitted tokens for the type of role are listed below.

Token	Description
administrator	Administrator in the organization (e.g. School). May be used for enrollment.
aide	Someone who provides appropriate aide to the user but NOT also one of the other roles.
guardian	Guardian of the user and NOT the Mother or Father. May also be a Relative.
parent	Mother or father of the user.
proctor	Exam proctor. Added in V1.1. May be used for enrollment.
relative	A relative of the user and NOT the Mother or Father. May also be the Guardian.
student	A student at a organization (e.g. School). May be used for enrollment.
teacher	A Teacher at organization (e.g. School). May be used for enrollment.

4.13.6. ScoreStatus

The set of permitted tokens for the type of score status are listed below.

Token	Description
exempt	The result is exempt i.e. this score does NOT contribute to any summative assessment.
fully graded	The result is fully graded.
not submitted	The result is not submitted.
partially graded	The result is partially graded. Further scoring will be undertaken and this score must NOT be used in summative assessment i.e. it must become 'fully graded'.
submitted	The result is submitted. This is a FINAL score and can only be changed as part of a formal review process.

The standard workflow is based upon the cycle of: not submitted -> submitted -> partially graded -> fully graded.

4.13.7. SessionType

The set of permitted tokens for the type of academic session are listed below.

Token	Description
gradingPeriod	Denotes a period over which some grade/result is to be awarded.
semester	Denotes a semester period. Typically there a two semesters per schoolYear.
schoolYear	Denotes the school year.
term	Denotes a term period. Typically there a three terms per schoolYear.

4.13.8. StatusType

The set of permitted tokens for the type of status are listed below.

Token	Description
active	An active record.
tobedeleted	Denotes that it is safe to delete the record.
inactive	DEPRECATED. To be mapped to 'tobedeleted.

4.14. Common Data Models

4.14.1. GUID

This is a globally unique identifier: it may or may not take the form of a Universal Unique Identifier (UUID). It is a derived from the 'String' base type. Best practice is that the value is globally unique using an appropriate naming/numbering system. Note that the value of a GUID should be treated as case-sensitive.

4.14.2. GUIDRef

This is a reference to a globally unique identifier. The structure is defined in Figure 4.14/Table 4.13.

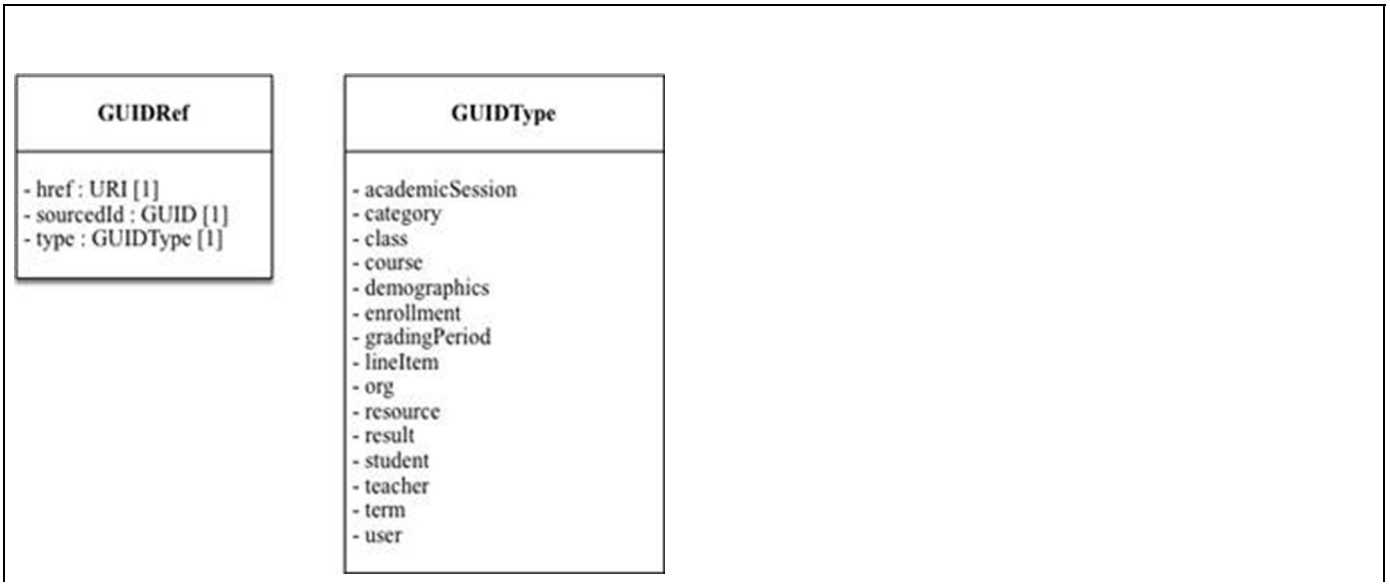


Figure 4.14 - GUIDRef Data Model.

Table 4.13 - Data Elements for GUIDRef.

Data Element Name	Multiplicity	Example / notes
href	1	The URI for the type of object being referenced.
sourcedId	1	The globally unique identifier of the object being referenced.
type	1	The type of object being referenced. This is enumerated.

4.15. Base Data-types

The set of base data-types are defined in Table 4.14.

Table 4.14 - The set of base data-types.

Data-type	Definition
Boolean	The boolean data-type with permitted values of "true" and "false".
Date	Denotes a date format. Dates MUST be expressed using ISO 8601 format (http://tools.ietf.org/html/rfc3339), more commonly formatted as "YYYY-MM-DD" e.g. "2002-04-23"
DateTime	Denotes a timestamp format. DateTimes MUST be expressed in W3C profile of ISO 8601 and MUST contain the UTC timezone e.g. "2012-04-23T18:25:43.511Z"
Float	Denotes a floating point number
gYear	Denotes a date format of year only. Years MUST be expressed using ISO 8601 format (http://tools.ietf.org/html/rfc3339), more commonly formatted as "YYYY" e.g. "2002"
String	Denotes a sequence of characters that should follow the description. Unless otherwise constrained, a system must be capable of handling strings that are at least 256 characters long.
URI	Denotes a URI string format.

5. JSON Binding

5.1. AcademicSessions

The JSON data structure for the academic sessions data model is shown in Code 5.1[2].

Code 5.1 - JSON binding of the AcademicSessions data model.

```
0000 {
0001 "academicSession": {
0002 "sourcedId" : "<sourcedid of this academicSession (term)>"
0003 "status" : "active | tobedeleted"
0004 "dateLastModified" : "<date this object was last modified>"
0005 "title" : "<name of the academicSession (term)>"
0006 "startDate" : "<academicSession (term) start date>"
0007 "endDate" : "<academicSession (term) end date>"
0008 "type" : "term"
0009 "parent" : {
0010 "href" : "<href of the parent for this academic session>"
0011 "sourcedId" : "<sourcedId of the parent for this session>"
0012 "type" : "academicSession"
0013 }
0014 "children" : [{
0015 "href" : "<href of 1st child for this academic session>"
0016 "sourcedId" : "<sourcedId of the 1st child for this session>"
0017 "type" : "academicSession"
0018 },
0019 {...}
0020 ]
0021 "schoolYear" : "2015"
0022 }
0023 }
```

Key points to note are:

a) The parent academic session is identified using lines [0009-0013];

b) The children academic sessions are identified using lines [0014-0020];

c) The addition of the new 'schoolYear' value in line 0021.

NOTE: In the case of properties that could be a collection e.g. 'children', then a JSON array must always be used i.e. even when there is only a single instance. This rule must be applied to all of the JSON encoding for all objects.

The JSON representation of a Grading Period has the same structure and carries the relationship to the term.

Code 5.2 - JSON binding of the Grading Period data model.

```
0000 {
0001 "academicSession": {
0002 "sourcedId" : "<sourcedid>"
0003 "status" : "active | tobedeleted"
0004 "dateLastModified" : "<date this object was last modified>"
0005 "title" : "<name of the academicSession (grading period)>"
0006 "startDate" : "<academicSession (grading period) start date>"
0007 "endDate" : "<academicSession (grading period) end date>"
0008 "type" : "gradingPeriod"
0009 "parent" : {
0010 "href" : "<href of the parent for this academic session>"
0011 "sourcedId" : "<sourcedId of the parent for this session>"
0012 "type" : "academicSession"
0013 }
0014 "schoolYear" : "2015"
0015 }
0016 }
```

Key points to note are:

a) The data structure is defined as a grading period using the 'type=gradingPeriod' field in line 0008;

b) The addition of the new 'schoolYear' value in line 0014.

5.2. Class

The extended JSON data structure for the class data model in v1.1 is shown in Code 5.3.

Code 5.3 - JSON binding of the extended Class data model.

```
0000 {
0001 "class" : {
0002 "sourcedId": "<sourcedId of this class >"
0003 "status" : "active | tobedeleted"
0004 "dateLastModified" : "<date this class was last modified>"
0005 "title" : "<name of this class>"
0006 "classCode" : "<human readable code for this class>"
0007 "classType" : "homeroom | scheduled"
0008 "location" : "<physical location of this class">
0009 "grades" : [ "<grade of this class>" ]
0010 "subjects" : [<"1st subject ","2nd subject".."nth subject">]
0011 "course" : {
0012 "href": "<href of the course that this is a class of>"
0013 "sourcedId": "<sourcedId of the course that this is a class of>"
0014 "type" : "course"
0015 }
0016 "school" : {
0017 "href": "<href of the school that this is a class of>"
0018 "sourcedId": "<sourcedId of the school that this is a class of>"
0019 "type" : "org"
0020 }
0021 "terms" : [{
0022 "href": "<href of the first term that this class is in>"
0023 "sourcedId": "<sourcedId of the 1st term that this class is in>"
0024 "type" : "academicSession"
0025 }, {
0026 ...
0027 }]
```

```
0028  "subjectCodes" : ["1st subject code".."n'th subject code" ] "periods" : ["<List of associated periods that class is
0029  taught>"]
0030  "resources" : [
0031  {
0032  "href" : "<href of the resource related to this class>"
      "sourcedId" : "<sourcedId of the 1st resource>"
0033  "type": "resource"
0034  }
0035  ...
0036  {
0037  "href" : "<href of the resource related to this class>"
0038  "sourcedId" : "<sourcedId of the nth resource>"
0039  "type": "resource"
0040  }]
0041  }
0042  }
```

Key points to note are:

a) The subject codes assigned in line 0028;
b) The periods that the class are taught are listed in line 0029;
c) The set of resources are identified using lines [0030-0041].

5.3. Course

The JSON data structure for the original (v1.0) data model is shown in Code 5.4.

Code 5.4 - JSON binding of the Course data model.

```
0000  {
0001  "course" : {
0002  "sourcedId": "<sourcedId of the course>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this object was last modified>"
      "metadata" : {
0005  "duration" : "<how long this course takes to teach>"
0006  }
0007  "title" : "<title of the course>"
```

```
0008  "schoolYear" : {
0009  "href": "<href of the academicSession (school year) related to this course>"
0010  "sourcedId": "<sourcedId of the academicSession (school year) related to this course>"
0011  "type" : "academicSession"
0012  }
0013  "courseCode" : "<course code for the course>"
0014  "grades" : [ "<the grade for this course>" ]
0015  "subjects" : ["1st subject","2nd subject".."n'th subject" ]
0016  "org" : {
0017  "href": "<href of the org related to this course>"
0018  "sourcedId": "<sourcedId of the org related to this course>"
0019  "type" : "org"
0020  }
0021  }
0022  }
0023
0024
0025
```

Key points to note are:

a) The 'duration' metadata is shown in lines [0005-0007;
b) The organization associated with this course is identified in lines [0019-0023].

The JSON data structure for the extended (v1.1) data model to include reference to the associated course resources is shown in Code 5.5.

Code 5.5 - JSON binding of the new Course data model.

```
0000  {
0001  "course" : {
0002  "sourcedId": "<sourcedId of the course>"
0003  "status" : "active | tobedeleted"
0004        "dateLastModified" : "<date this object was last modified>"
      "metadata" : {
```

```
0005  "duration" : "<how long this course takes to teach>"
0006  }
0007  "title" : "<title of the course>"
0008  "schoolYear" : {
0009  "href": "<href of the academicSession (school year) related to this course>"
0010  "sourcedId": "<sourcedId of the academicSession (school year) related to this course>"
0011  "type" : "academicSession"
0012  }
0013  "courseCode" : "<course code for the course>"
0014  "grades" : [ "<the grade for this course>" ]
0015  "subjects" : ["1st subject","2nd subject".."n'th subject" ]
0016  "org" : {
0017  "href": "<href of the org related to this course>"
0018  "sourcedId": "<sourcedId of the org related to this course>"
0019  "type" : "org"
0020  }
0021  "subjectCodes" : ["1st subject code".."n'th subject code" ]
0022  "resources" : [
0023  {
0024  "href" : "<href of the resource related to this course>"
0025  "sourcedId" : "<sourcedId of the 1st resource>"
0026  "type": "resource"
0027  }
0028  ...
0029  {
0030  "href" : "<href of the resource related to this course>"
0031  "sourcedId" : "<sourcedId of the nth resource>"
0032  "type": "resource"
      }]
0033  }
0034  }
0035
```

```
0036
0037
```

Key points to note are:

a) The subject codes assigned in line 0024;
b) The set of resources are identified using lines [0025-0036].

5.4. Demographics

The JSON data structure for the demographics data model is shown in Code 5.6.

Code 5.6 - JSON binding of the Demographics data model.

```
0000 {
0001 "demographics" : {
0002 "sourcedId" : "<sourcedid of this demographics record (same as user referenced)>"
0003 "status" : "active | tobedeleted"
0004 "dateLastModified" : "<date these demographics were last modified>"
0005 "birthDate" : "<value>" (e.g. 1980-01-01)
0006 "sex" : "<value>" (e.g. Male)
0007 "americanIndianOrAlaskaNative" : "<value>" (e.g. false)
0008 "asian" : "<value>" (e.g. false)
0009 "blackOrAfricanAmerican" : "<value>" (e.g. true)
0010 "nativeHawaiianOrOtherPacificIslander: "<value>"
0011 "white" : "<value>"
0012 "demographicRaceTwoOrMoreRaces" : "<value>"
0013 "hispanicOrLatinoEthnicity" : "<value>"
0014 "countryOfBirthCode" : "<value>" (e.g. US)
0015 "stateOfBirthAbbreviation" : "<value>" (e.g. NY)
0016 "cityOfBirth" : "<value>" (e.g. New York)
0017 "publicSchoolResidenceStatus : "<value>" (e.g. 01652)
0018 }
     }
0019
```

```
0020
0021
```

5.5. Enrollments

The abstract data model requires the sourcedIds of the user, school and class for the enrolment. In the JSON these need to be specified as links. This means a full 'href' as well as a 'sourcedId', as shown in Code 5.7.

Code 5.7 - JSON binding of the Enrollment data model.

```
0000 {
0001 "enrollment" : {
0002 "sourcedId" : "<sourced id of this enrollment>"
0003 "status" : "<status of this enrollment>"
0004 "dateLastModified": "<date this enrollment was last modified>"
0005 "role" : "teacher | student | administrator | proctor"
0006 "primary" : "true" | "false"
0007 "user" : {
0008 "href" : "<href of the user for this enrollment>"
0009 "sourcedId" : "<sourcedId of the user for this enrollment>"
0010 "type" : "user"
0011 }
0012 "class" : {
0013 "href" : "<href of the class for this enrollment>"
0014 "sourcedId" : "<sourcedId of the class for this enrollment>"
0015 "type" : "class"
0016 }
0017 "school" : {
0018 "href" : "<href of the school for this enrollment>"
0019 "sourcedId" : "<sourcedId of the school for this enrollment>"
     "type" : "org"
```

```
0020  }
0021  "beginDate" : "<Enrollment start date>" (e.g. 2015-01-01Z)
0022  "endDate" : "<Enrollment end date>" (e.g. 2015-12-31Z)
0023  }
0024  }
0025
0026
```

Key points to note are:

a) The User sourcedId is given in lines [0008-0012];
b) The Class sourcedId is given in lines [0013-0017];
c) The School sourcedId is given in lines [0018-0022];
d) The new 'beginDate' and 'endDate' structures are shown in lines [0023-0024].

5.6. LineItem

The JSON data structure for the line item data model is shown in Code 5.8.

Code 5.8 - JSON binding of the LineItem data model.

```
0000  {
0001  "lineItem" : {
0002  "sourcedId" : "<sourcedId of this lineItem>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this object was last modified>"
0005  "title" : "<title of this lineItem>"
0006  "description" : "<description of this lineItem>"
0007  "assignDate" : "<date that this lineItem was assigned>"
0008  "dueDate" : "<date that this lineItem is due>"
0009  "category" : {
0010  "href" : "<href to this category>"
0011  "sourcedId" : "<sourcedId of this category>"
0012  "type" : "category"
```

```
0013  }
0014  "class" : {
0015  "href" : "<href to the class this line item is for>"
0016  "sourcedId" : "<sourcedId of the class this line item is for>"
0017  "type" : "class"
0018  }
0019  "gradingPeriod" : {
0020  "href" : "<href to this grading period>"
0021  "sourcedId" : "<sourcedid of this grading period>"
0022  "type" : "academicSession"
0023  }
0024  "resultValueMin" : "<floating point value of the minimum score>"
0025  "resultValueMax": "<floating point value for the maximum score>"
0026  }
0027  }
```

Key points to note are:

a) The related category information is given in lines [0009-0013];
b) The identifier for the associated class is given in lines [0014-0018];
c) The identifier for the associated gradingPeriod is given in lines [0019-0023];
d) The constraints for an associated results and defined in lines [0024-0025].

5.7. LineItem Categories

The JSON data structure for the line item categories data model is shown in Code 5.9.

Code 5.9 - JSON binding of the LineItem Categories data model.

```
0000  {
0001  "category" : {
0002  "sourcedId" : "<sourcedId of this category>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this object was last modified>"
```

```
0005  "title" : "<title of this category>"
0006  }
0007  }
```

5.8. Org

The JSON data structure for the Org model is shown in Code 5.10.

Code 5.10 - JSON binding of the Org data model.

```
0000  {
0001  "org": {
0002  "sourcedId" : "<sourcedId of this org>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this ORG was last modified>"
0005  "name" : "<name of the org>"
0006  "type" : "school | local | state | national"
0007  "identifier" : "<human readable identifier for this organization>"
0008  "parent" : {
0009  "href" : "<href to the parent org>"
0010  "sourcedId" : "<sourcedId of the parent org>"
0011  "type" : "org"
0012  },
0013  "children" : [
0014  {
0015  "href" : "<href of the first child org>"
0016  "sourcedId" : "<sourcedId of the first child org>"
0017  "type" : "org"
0018  },
0019  ...
```

```
0020  {
0021  "href" : "<href of the n'th child org>"
0022  "sourcedId" : "<sourcedId of the n'th child org>"
0023  "type" : "org"
0024  }]
0025  }
0026  }
```

Key points to note are:

a) The parent organization is identified using lines [0008-0012];
b) The children organizations are identified using lines [0013-0024].

The JSON representation of an array of schools is shown in Code 5.11.

Code 5.11 - JSON binding of the Orgs data model.

```
0000  {
0001  "orgs": [
0002  {
0003  "sourcedId": "<sourcedid of first school>"
0004  "status" : "active | tobedeleted"
0005  "dateLastModified" : "<date this school was last modified>"
0006  "identifier": "0808120938"
0007  "name" : "<name of first school>"
0008  "type" : "school"
0009  "parent" : {
0010  "href" : "<href to the parent org>"
0011  "sourcedId" : "<sourcedid of the parent org>"
0012  "type" : "org"
0013  }
0014  },
0015  ....
0016  {
```

```
0017  "sourcedId": "<sourcedid of n'th school>"
0018  "status" : "active | tobedeleted"
0019  "dateLastModified" : "<date this school was last modified>"
0020  "identifier": "0808120938"
0021  "name" : "<name of n'th school>"
0022  "type" : "school"
0023  "parent" : {
0024  "href" : "<href to the parent org>"
0025  "sourcedId" : "<sourcedid of the parent org>"
0026  "type" : "org"
0027  }
0028  }]
0029  }
```

Key points to note are:

a) The set of 'org' exchanges is denoted by 'orgs' in line 0001;
b) The parent organization for the first 'org' is identified using lines [0009-0013];
c) The parent organization for the last 'org' is identified using lines [0023-0027].

5.9. Resource

The JSON data structure for the extended (v1.1) data model to include reference to the associated course resources is shown in Code 5.12.

Code 5.12 - JSON binding of the new Resource data model.

```
0000  {
0001  "resource" : {
0002  "sourcedId": "<sourcedId of the resource>"
0003  "title": "<title of the resource>"
0004  "roles": ["teacher | student | parent | guardian | relative | aide | administrator | proctor"]
0005  "importance": "primary | secondary"
```

```
0006  "vendorResourceId" : "<vendor allocated unique resource ID>"
0007  "vendorId": "<GUID of the vendor who created the resource>"
0008  "applicationId": "<GUID of the application to use the resource>"
0009  }
0010  }
0011
```

Key points to note are:

a) Each resource MUST have a 'sourcedId' [line 0002] (used for the interoperability exchange) and the unique identifier allocated by the vendor to the resource [line 0007] used to provide identification of the resource within the learning context.

5.10. Result

The JSON data structure for the result data model is shown in Code 5.13.

Code 5.13 - JSON binding of the Result data model.

```
0000  {
0001  "result" : {
0002  "sourcedId" : "<sourcedid of this grade>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this result was last modified>"
0005  "lineItem" : {
0006  "href" : "<href to this lineItem>"
0007  "sourcedId" : "<sourcedId of this lineItem>"
0008  "type" : "lineItem"
0009  }
0010  "student" : {
0011  "href" : "<href to this student>"
0012  "sourcedId" : "<sourcedId of this student>"
0013  "type" : "user"
0014  }
0015  "score" : <score of this grade in floating point format>
0016  "scoreStatus" : "not submitted | submitted | partially graded | fully graded | exempt"
      "scoreDate" : "<date that this grade was assigned>"
```

```
0017  "comment" : "<a comment to accompany the score>"
0018  }
0019  }
0020
0021
```

Key points to note are:

a) The actual result, the score, is supplied in Line 0015.

5.11. Teachers and Students (Users)

The JSON data structure, for v1.0, for the users data model is shown in Code 5.14. A Teacher and Student are an instance of User.

Code 5.14 - JSON binding of the User data model.

```
0000  {
0001  "user" : {
0002  "sourcedId" : "<sourcedid of this user>"
0003  "status" : "active | tobedeleted"
0004  "dateLastModified" : "<date this user was last modified>"
0005  "username" : "<username to use for this user>"
0006  "userIds" : [{
0007  {
0008  "type" : "< type of identifier >",
0009  "identifier" : "< assigned value for the identifier>"
0010  },
0011  {...}
0012  ]
0013  "enabledUser" : "true | false"
0014  "givenName" : "<this user's given name>"
0015  "familyName : "<this user's family name>"
0016  "role" : "teacher | student | parent | guardian | relative | aide | administrator"
0017  "identifier" : "<human readable ID, such as student id>"
0018  "email" : "<email address for this user>"
```

```
0019  "sms" : "<sms number for this user>"
0020  "phone" : "<phone number for this user>"
0021  "agents" : [
0022  {
0023  "href" : "href of the first agent (e.g. parent) for this user"
0024  "sourcedId" : "sourcedid of the first agent for this user"
0025  "type" : "user"
0026  }
0027  ...
0028  {
0029  "href": "href of the n'th agent for this user"
0030  "sourcedId" : "sourcedid of the n'th agent for this user"
0031  "type" : "user"
0032  }]
0033  "orgs" : [{
0034  "href": "<href of the 1st org to which this user is attached>"
0035  "sourcedId": "<sourcedId of the 1st org to which this user is attached>"
0036  "type" : "org"
0037  }
0038  ...
0039  {
0040  "href": "<href of the nth org to which this user is attached>"
0041  "sourcedId": "<sourcedId of the nth org to which this user is attached>"
0042  "type" : "org"
0043  }]
0044  }
0045  }
0046
0047
```

Key points to note are:

a) The role of the user must be correctly identified in line 0009;
b) The links to the set of agents are denoted by lines [0015-0026];
c) The links to the set of organizations are denoted by lines [0027-0039].

The JSON data structure for the extended users data model, v1.1, is shown in Code 5.15.

Code 5.15 - JSON binding of the extended User data model.

```
0000 {
0001 "user" : {
0002 "sourcedId" : "<sourcedid of this user>"
0003 "status" : "active | tobedeleted"
0004 "dateLastModified" : "<date this user was last modified>"
0005 "username" : "<username to use for this user>"
0006 "userIds" : [{
0007 "type" : "<Type of identifier>"
0008 "identifier" : "<active directory/lti user id/some other id >"
0009 }]
0010 "enabledUser" : "true | false"
0011 "givenName" : "<this user's given name>"
0012 "familyName : "<this user's family name>"
0013 "middleName : "name1 name2 name3"
0014 "role" : "teacher | student | parent | guardian | relative | aide | administrator | proctor"
0015 "identifier" : "<human readable ID, such as student id>"
0016 "email" : "<email address for this user>"
0017 "sms" : "<sms number for this user>"
0018 "phone" : "<phone number for this user>"
0019 "agents" : [
0020 {
0021 "href" : "href of the first agent (e.g. parent) for this user"
0022 "sourcedId" : "sourcedid of the first agent for this user"
0023 "type" : "user"
0024 },
     ...
```

```
0025  {
0026  "href": "href of the n'th agent for this user"
0027  "sourcedId" : "sourcedid of the n'th agent for this user"
0028  "type" : "user"
0029  }]
0030  "orgs" : [{
0031  "href": "<href of the 1st org to which this user is attached>"
0032  "sourcedId": "<sourcedId of the 1st org to which this user is attached>"
0033  "type" : "org"
0034  }
0035  ...
0036  {
0037  "href": "<href of the nth org to which this user is attached>"
0038  "sourcedId": "<sourcedId of the nth org to which this user is attached>"
0039  "type" : "org"
0040  }]
0041  "grades" : [ "1st Grade", .. , "nth Grade"]
0042  "password" : "<Password for the user>"
0043  }
0045  }
0046
0047
0048
```

Key points to note are:

a) The restructured, and renamed, 'userId' is shown in lines [0006-0009];
b) The new 'middleName' structure is shown in line 0012;
c) The new 'grades' and 'password' structures are shown in lines [0045-0046].

5.12. Returning An Array of Objects

It is possible to request the return of a collection of objects available e.g. all users, all schools, all courses, etc. The outline JSON returned for each of the single and collection calls is shown in Table 5.1. If the returned response **could** be a collection then the equivalent returned JSON must use an array with a single entry.

Table 5.1 JSON structure for returning a single objects and collection of objects.

Single Object Returned	Collection of Objects Returned

{
"academicSession" : {
}
}
 	
This is also used for 'gradingPeriod' and 'term' payloads.
	{
	"academicSessions" : [
	{ ... },
	...
	{ ... }
	]
	}
	
	This is also used for 'gradingPeriods' and 'terms' payloads.

{
"category" : {
}
}
	{
	"categories" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"class" : {
}
}
	{
	"classes" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"course" : {
}
}
 	
	{
	"courses" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"demographics" : {
}
}
 	
	{
	"demographics" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"enrollment" : {
}
}
 	
	{
	"enrollments" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"lineItem" : {
}
}
 	
	{
	"lineItems" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"org" : {
}
}
 	
	{
	"orgs" : [
	{ ... },
	...

This is also used for 'school' payloads.	{ ... }] } This is also used for 'schools' payloads.

{
"resource" : {
}
}
 	
	{
	"resources" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"result" : {
}
}
 	
	{
	"results" : [
	{ ... },
	...
	{ ... }
	]
	}

{
"user" : {
}
}
This is also used for 'student' and 'teacher' payloads.
	{
	"users" : [
	{ ... },
	...
	{ ... }
	]
	}
	This is also used for 'students' and 'teachers' payloads.

5.13. Metadata

In Code 5.16 is an example in JSON (showing an ncesId for a fictitious private female only boarding school that is also an 1EdTech associate member).

Code 5.16 - JSON binding of the Metadata data model.

```
0000 {
0001 ...
     "sourcedId" : "<sourcedId of this object>"
0002 "status" : "active | tobedeleted"
0003 "dateLastModified" : "<date this object was last modified>"
0004 "metadata" : {
0005 "ims.classification" : "private"
0006 "ims.boarding" : "true"
0007 "http://www.nbrs.org": "FL"
0008 }
0009 }
0010
```

Key points to note are:

a) The actual metadata is listed in lines [0005-0009].

5.14. Error Payloads

In Code 5.17 is an example of the transaction status code payload that MUST be returned in the case of a failure to service the request. This payload may also be appended to a partially successfully completed request.

Code 5.17 - Transaction-level error code payload.

```
0000 {
     "statusInfoSet" : [
0001 {
0002 "imsx_codeMajor" : "success | failure | unsupported"
0003 "imsx_severity" : "status | warning | error"
0004 "imsx_messageRefIdentifier" : "<request message ID reference>"
0005 "imsx_operationRefIdentifier" : "<request operation ID reference>"
0006 "imsx_description" : "<human readable description>"
0007 "imsx_codeMinor" : "<from the set of permitted values>"
0008 }
0009
```

0010	{ ... }
0011	]
	}

Key points to note are:

a) The first statusInfo information is given in lines [0002-0009];

b) The 'codeMajor' value is given in line 0003 (a required attribute);

c) The 'severity' value is given in line 0004 (a required attribute);

d) The 'codeMinor' value is given in line 0008 (a required attribute);

e) A human readable description is given in line 0007 (an required optional);

f) If the request provides some form of message identifier then it can be returned as shown in line 0005 (an required optional);

g) If it is important to return some indication of the operation being requested the some ID can be returned as shown in line 0006 (for example the name of the endpoint e.g. getUser, getResources, etc.).

Appendix A - Recommended Vocabularies

1EdTech recommends that the following vocabularies and terms be used in the data model.

Element Name	Parent Class	Vocabulary
grades	Classes, Courses, Users	The permitted vocabulary is from CEDS (Version 5) and the 'Entry Grade Level' element: https://ceds.ed.gov/CEDSElementDetails.aspx?TermId=7100.
subjectCodes	Classes, Courses	This is a machine-readable set of codes and the number should match the associated 'subjects' attribute. For systems deployed in the USA this vocabulary SHOULD be a School Courses for the Exchange of Data (SCED) code: http://nces.ed.gov/forum/SCED.asp.
countryOfBirthCode	Demographics	The permitted vocabulary is from CEDS (Version 5) and the "Country of Birth Code" element: https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20002

stateOfBirthAbbreviation	Demographics	The permitted vocabulary is from CEDS (Version 5) and the "State of Birth Abbreviation" element: https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20837
publicSchoolResidenceStatus	Demographics	The permitted vocabulary is from CEDS (Version 5) and the "Public School Residence Status" element: https://ceds.ed.gov/CEDSElementDetails.aspx?TermxTopicId=20863

Appendix B - The Complete Data Model

The complete data model is shown in Figure B1.

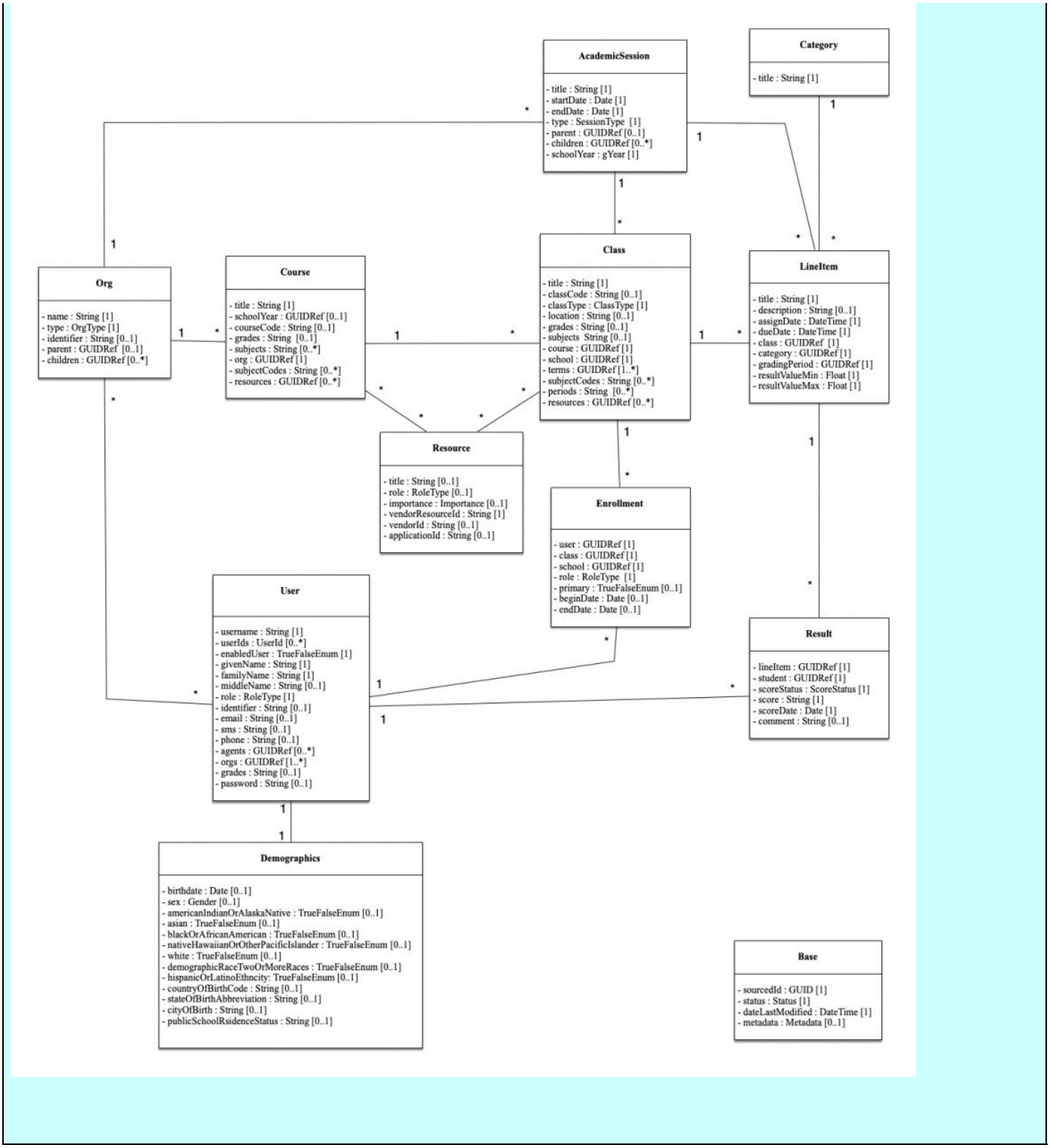


Figure B1 - The complete data model for OneRoster.

About This Document

Title: 1EdTech OneRoster® Specification

Editor: Colin Smythe (1EdTech) and Phil Nicholls (Oracle)

Version: 1.1

Document Release: 2.0

Version Date: 21st October, 2021

Status: Final Release

Summary: This document outlines a vision for a K12 focused Learning Information Services, built upon 'technical simplicity'. The OneRoster core specification uses a RESTful binding using JSON data structures to achieve technical interoperability.

Purpose: This document is made available for public adoption.

Document Location: http://www.imsglobal.org/lis

toc | top

List of Contributors

The following individuals contributed to the development of this document:

William Baker	Pearson (USA)
Arthur Barstow	McGraw-Hill (USA)
Sari Connard	Performance Matters (USA)
Hank Davidson	Pearson (USA)
Vijay Dhanaraj	Classlink (USA)
David Gappa	Safari Montage (USA)
Linda Feng	Unicom (USA)
Tom Ingram	Escambia County School District (USA)
Oxana Jurosevic	Instructure (USA)
Mike Kaastra	Desire2Learn (Canada)
Jong Kim	Pearson (USA)
Andrew Kuritzky	HMH (USA)
Lisa Mattson	1EdTech (USA)
David Mayes	Gwinnett County Schools (USA)
Andy Miller	Learning.com (USA)
Phil Nicholls	Oracle (UK)
Padraig O'hiceadha	HMH (UK)
Upendra Penegalapati	Pearson (USA)

George Perreault	Orange County Public Schools (USA)
James Perreault	FLVS (USA)
Patrick Porter	Houston ISD (USA)
Wendy Riedy	Sungard K12 (USA)
Kurt Rompot	Pearson (USA)
Marc Sheftel	Pearson (USA)
Colin Smythe	1EdTech (UK)
Konrad Stimeling	K12 (USA)
Aditya Subramaniam	Schoology (USA)
Matt Vella	Schoology (USA)
TJ Vering	Microsoft (USA)
Mark Walls	Gwinnett County Schools (USA)
Stanley Watts	Classlink (USA)
Mike Zackerson	Instructure (USA)

Revision History

Version No.	Release Date	Comments
V1.0 Final	3rd June 2015	Final Release of the OneRoster Specification.
V1.1 Final Release	17th April, 2017	Second Final Release. The key changes are • Support for describing resources associated with a Course and/or a Class has been introduced to the data model; • A number of minor data model additions and field renaming has been completed; • An extensive set of new operations have been added for managing LineItems, Results and Categories; • New optional security features have been added.
V1.1 Final Release (Document Version 1.0.1)	12th June, 2017	A number of editorial corrections ONLY.
V1.1 Final Release	12th September, 2018	The modifications made are: • The use of HMAC-SHA256 to replace HMAC-SHA1 has been defined in Section 3.6;

(Document Version 1.0.2)		• The payload examples in Section 5.12 for collections of objects have been corrected; • The Code block 5.15 has been corrected i.e. to include the new definition of 'userId' • In Section 3.4.3 the examples explaining the filtering rules for equals and contains have been corrected.
V1.1 Final Release (Document Version 1.0.3)	10th July, 2019	The modifications made are: • In Section 3.6.3 the details for the use of the required set of scopes in the OAuth 2 Client Credentials exchange has been added.
V1.1 Final Release (Document Version 1.0.4)	7th March, 2020	The modifications made are: • In Section 3.6.2 the details for the format of the access token request have been corrected i.e. to use the message payload and NOT query parameters.

V1.1.1 Final Release	31st December, 2020	Minor revision of this specification in response to the identification of a number of clarifications. These clarifications are:
		1. Clarification that filtering MUST be supported for:-
		  ◦ All of data fields that are REQUIRED
		  ◦ The OPTIONAL data fields that are supported by the Service Provider;
		2. Clarification that for systems deployed in USA, School Courses for the Exchange of Data (SCED) codes SHOULD be used for the values of the property 'subjectCodes' in the 'Class' and 'Course' payloads;
		3. Section 3.7 (Serialization Format) has been extended to explain that the occurrence of NULL/EMPTY data fields is PROHIBITED;
		4. Clarification that the Boolean data types are enumerations with permitted values of ‘true’ and ‘false’ ONLY. The properties clarified are:
		  ◦ Demographics.americanIndianOrAlaskaNative
		  ◦ Demographics.asian
		  ◦ Demographics.blackOrAfricanAmeircan
		  ◦ Demographics.nativeHawaiianOrOtherPacificIslander
		  ◦ Demographics.white
		  ◦ Demographics.demographicRaceTwoOrMoreRaces
		  ◦ Demographhics.hispanicOrLatinoEthnicity
		  ◦ Enrollments.primary
		  ◦ Users.enabledUser

V1.1.2 Final Release	23rd June, 2021	The modifications made are: • Clarification on the use of the HTTP 200 and 404 codes when responding to request for collections when a valid request has been made and where no records are to be returned.
V1.1 (Document Version 2.0)	21st October 2021	The modifications made are: • ALL references to, and descriptions of, OAuth 1.0a message signing have been removed. ALL implementations MUST now use OAuth 2.0 Bearer Token Client Credentials.
V1.1 (Document Version 2.0.1)	29th Aoril 2022	Clarification that the ONLY permitted values for roles in an enrollment are: { administrator | proctor | student | teacher }.

[1] The format of the GUID is an implementation specific decision i.e. it is NOT restricted to the 128-bit form of a Universal Unique Identifier (UUID).

[2] NOTE: In order to make all examples look clearer, commas have been omitted from the end of lines.