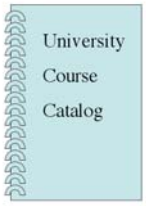
Open Service Interface Definition

CourseManagement



Document Release: 2.0
OSID version 2.0

Summary

The CourseManagement Open Service Interface Definition (OSID) primarily supports creating and managing a CourseCatalog. The catalog is organized into:

- CanonicalCourses, which are general and exist across terms;
- CourseOfferings for CanonicalCourses, which occur in a specific term, have a CourseGradeType[1], a Status, etc; and
- CourseSections for CourseOfferings, which have a meeting location, schedule, student roster, etc.

When used in concert, the OSIDs comprise a complete system with each Service focused exclusively on a particular area. For example, the roles related to a CourseOffering and CourseSection are defined through the Authorization OSID; coursework and course material can be defined in the Assessment and Repository OSIDs; course grades are assigned through the Grading OSID, and so on.

Service Definition

An examination of an Open Service Interface Definition usually begins with the Manager. All Managers[2] provide the way to create the objects that implement the principal interfaces in the Service. Before dis-

[1] Refer to “Discussion of Type” for more information.

[2] org.osid.OsidManager defines the interface extended by the Managers in each OSID. Here we will be discussing osid.coursemanagement.CourseManagementManager. Refer to ”OsidManager” for more information.

cussing the Manager in detail, we will review the intended function of the CourseManagement Service overall.

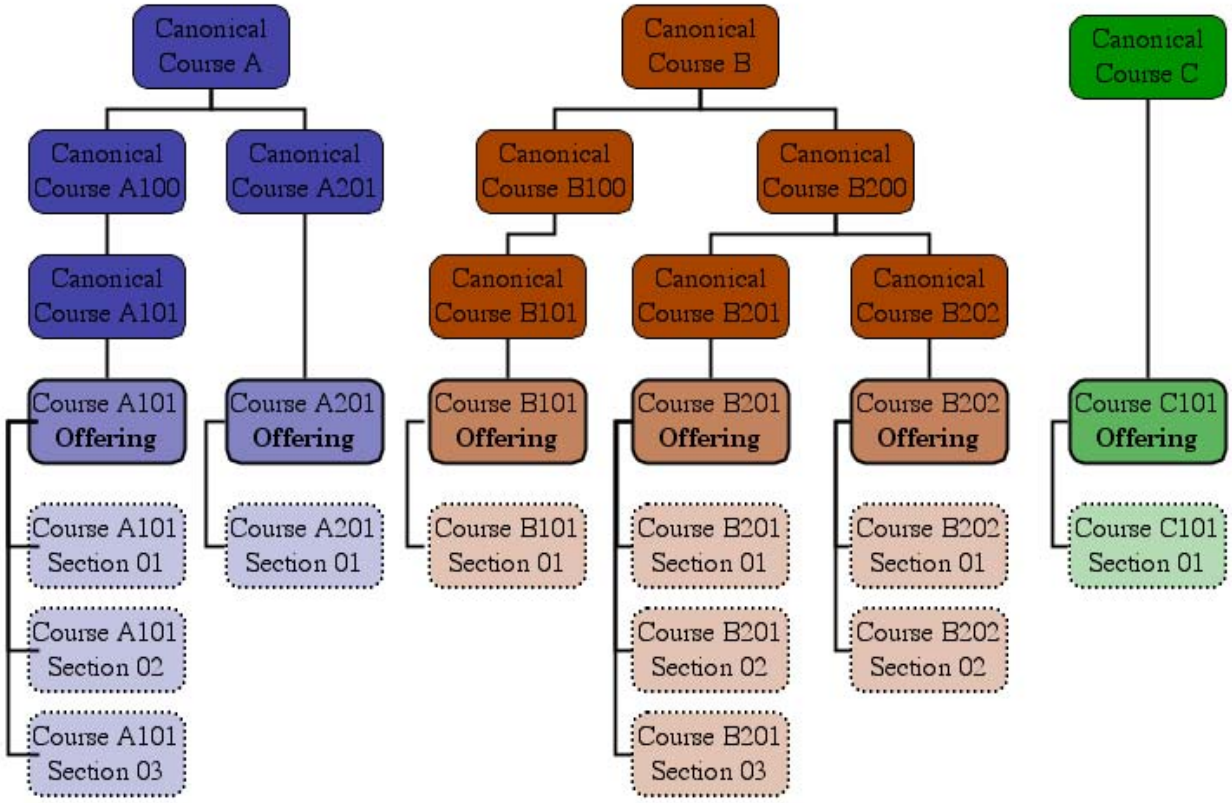


Figure 1: Organization of CanonicalCourses, CourseOfferings, and CourseSections

The top-tier organizing structure for CourseManagement is the CanonicalCourse. A CanonicalCourse includes several properties: a title, a description, a number, a unique Id[3], a type, and number of credits. The purpose of these properties is to provide enough structure and flexibility to map a CanonicalCourse to the various institutions' systems for describing academic courses.

CanonicalCourse also include:

- a list of any equivalent CanonicalCourses (for cross-listing);
- a list of any prerequisites;
- a list of any CanonicalCourses (hierarchical children); and
- a list of any CourseOfferings.

CanonicalCourses are designed for use across many Terms and might be archived, even after there are no longer CourseOfferings for it. CanonicalCourses are not intended to capture information about the Course for a specific Term. That is the role of the CourseOffering.

The CourseOffering contains the same title, Id, number, and description properties as a CanonicalCourse. These properties' values can be the same as those for the CanonicalCourse or overridden for this particu-

[3] Refer to "Discussion of Id" for more information.

lar CourseOffering.[4] In place of the CanonicalCourse's CourseType there is an OfferingType[5]. The CourseOffering does not include a credits property but it does include a GradeType. The CourseManagementManager supports creating, deleting, and getting grades of this Type for any Agent and CourseOffering. The CourseOffering is specific to a Term[6].

CourseOfferings include an AssetId property. This is a place to put a reference to some Repository or other Asset associated with the CourseOffering[7]. CourseOfferings also include a StatusType. This could be used to indicate open, closed, discontinued, etc but as with all Types, there are no restrictions on the meaning of the Type. The CourseOffering includes both a list of the parent CanonicalCourses (there must be at least one) as well as any created CourseSections.

The CourseSection again contains title, description, number and Id properties which can be those of the parent CourseOffering or overridden. The CourseSection is in what Students actually enroll and it has a Schedule[8], a location, and manages a roster. As with the CourseOffering, there is a property for a Asset reference.

org.osid.CourseManagement.CourseManagementManager

The CourseManagementManager is primarily responsible for creating CanonicalCourses and for returning the CourseManagement organization and various Types. The **createCanonicalCourse**() method creates a new CanonicalCourse with the specified properties. A unique Id for this CanonicalCourse is assigned by the implementation. While the title, number, and description may change, the Id and CourseType are immutable. The **deleteCanonicalCourse**() method deletes a CanonicalCourse by Id. The **getCanonicalCourse**() method returns a specific CanonicalCourse and **getCanonicalCourses**() returns all the CanonicalCourses or only those of a specific CourseType.

There are a variety of Types used in this OSID. For each, there is a "get" method. For example, each of the Course, Offering, and Section objects has both a Type and a Status Type. The Type of an Offering might be "Lecture" while the Status of that Offering might be "Open". Refer to the Method Summary table for details.

CourseOfferings and CourseSections are created and deleted elsewhere but they can be returned here, by Id, using **getCourseOffering**() and **getCourseSection**(). In addition to getting the Offerings and Sections, there are methods (**getCourseOfferings**() and **getCourseSections**(), respectively) to get just the Courses and Sections a particular Agent (student) is in.

CourseOfferings and CourseSections are associated with specific Terms. Terms, which have a Type, are created, deleted, and enumerated in various ways using **createTerm**(), **deleteTerm**(), **getTerm**(), **getTerms**(), and **getTermTypes**(), respectively. Terms are returned in chronological order.

Each CourseOffering includes a CourseGradeType. This Type, which might characterize a final grade, along with an Agent (student who took the course), and a specific CourseOffering can be created as a CourseGradeRecord. These CourseGradeRecords, which are involved in the methods **createCourseGradeRecord**(), **deleteCourseGradeRecord**(), **getCourseGradeRecord**(), and **get-**

[4] This is the implementation's choice.

[5] The OfferingType might be used to indicate an honors course or other designation. There are others ways to designate honors, for example through the Number property. It may also be the case that there are only specific honor Sections. All these configurations can be accommodated.

[6] An implementation may want to provide for rolling over CourseOfferings to successive terms.

[7] Refer to "OSID Repository" for more information.

[8] Refer to "OSID Scheduling" for more information.

CourseGradeRecords(), are for summative information[9], distinct from the assignment grading support in the Grading OSID. While the Grading OSID focuses on the learning systems, these course grades tie into the student information systems. Note that **getCourseGradingRecords**() can return all CourseGradeRecords or, more practically, a subset filtered by Agent, CourseGradeType, CourseOffering, or permutation.

CanonicalCourses can be organized into CourseGroups. The intent of the CourseGroup is to associate CanonicalCourses into a sequence, a major, a minor, a chain of prerequisites, corequisites, and so on. The kind of association is defined by the CourseGroupType. The **createCourseGroup()**, **deleteCourseGroup()**. **getCourseGroup()**, **getCourseGroups()**, and **getCourseGroupTypes()** methods manage this facility. The CourseGroup interface is responsible for adding, removing, and enumerating CanonicalCourses in the CourseGroup.

CourseManagementManager Method Summary

CanonicalCourse	createCanonicalCourse(String title, String number, String description, Type courseType, Type courseStatusType, float credits)
CourseGroup	createCourseGroup(Type courseGroupType)
CourseGradeRecord	createCourseGradeRecord(osid.shared.Id agentId, Id courseOfferingId, Type courseGradeType, Serializable grade)
Term	createTerm(Type termType, ScheduleItem[] schedule)
void	deleteCanonicalCourse(Id canonicalCourseId)
void	deleteCourseGradeRecord(Id courseGradeRecordId)
void	deleteCourseGroup(Id courseGroupId)
void	deleteTerm(Id termId)
CanonicalCourse	getCanonicalCourse(Id canonicalCourseId)
CanonicalCourseIterator	getCanonicalCourses()
CanonicalCourseIterator	getCanonicalCoursesByType(Type courseType)
CourseGroup	getCourseGroup(Id courseGroupId)
CourseGroupIterator	getCourseGroups(Id canonicalCourseId)
CourseGroupIterator	getCourseGroupsByType(Type courseGroupType)
TypeIterator	getCourseGroupTypes()
CourseOffering	getCourseOffering(Id courseOfferingId)
CourseOfferingIterator	getCourseOfferings(Id agentId)
CourseSection	getCourseSection(Id courseSectionId)

[9] This Service uses a simple, unstructured, Serializable grade while the Grading OSID offers more structure.

CourseSectionIterator	getCourseSections(Id agentId)
TypeIterator	getCourseStatusTypes()
TypeIterator	getCourseTypes()
CourseGradeRecordIterator	getCourseGradeRecords(Id agentId, Id courseOfferingId, Type courseGradeType)
TypeIterator	getEnrollmentStatusTypes()
TypeIterator	getCourseGradeTypes ()
TypeIterator	getOfferingStatusTypes()
TypeIterator	getOfferingTypes()
TypeIterator	getSectionStatusTypes()
TypeIterator	getSectionTypes()
Term	getTerm(Id termed)
TermIterator	getTerms()
TermIterator	getTermsByDate(long date)
TypeIterator	getTermTypes()
boolean	supportsUpdate()

org.osid.CourseManagement.CanonicalCourse

CanonicalCourses occupy the top-tier in the CourseManagement organization. A CanonicalCourse can be associated with others hierarchically[10]. If a CanonicalCourse is created through the CourseManagementManager's **createCanonicalCourse**(), it is assumed to be a root with no parent. If a CanonicalCourse is created through a CanonicalCourse's **createCanoncialCourse**() method, it is assumed to be the child of the creator. As in the CourseManagementManager, CanonicalCourses are managed further with the methods **deleteCanonicalCourse**() and **getCanonicalCourses**().

CanonicalCourses can create, delete, and get CourseOfferings related to them. The intent is that CanonicalCourses can be organized in a hierarchy but not CourseOfferings; they do not have CourseOfferings. It is possible for the same CourseOffering to have multiple parents since a CanonicalCourse can have equivalent Courses. The CourseOffering is only created once.

CanonicalCourses can be organized by the CourseManagementManager into CourseGroups. The intent of the CourseGroup is to associate CanonicalCourses into a sequence, a major, a minor, a chain of prerequisites, corequisites, and so on. The kind of association is defined by the CourseGroupType. CanonicalCourses can also have equivalents, perhaps for use in cross-listing. The methods **addEquivalentCourse()**, **removeEquivalentCourse()**, and **getEquivalentCourses()** facilitate this capability.[11]

[10] Refer to "OSID Hierarchy" for more information. The intent of this Service is to enable implementations to use a non-recursive Hierarchy.

[11] The intent is that if CanonicalCourse "A" is equivalent to CanonicalCourse "B", the reverse is also true.

CanonicalCourses, CourseOfferings, and CourseSections share common properties for title, description, number, and Id. All but the Id may be updated. It is an implementation's choice whether CanonicalCourses, CourseOfferings, and CourseSections simply use the same properties or adjust them for each level. Courses. CanonicalCourse has a property for the number of credits[12].

Each CanonicalCourse may have Topics associated with it. These Topics are simply Strings that might be defined in an institutional database of Topics and the intent is to provide a mechanism for calling out this information. CanonicalCourses have a CourseType. The specific values for this Type are the responsibility of an implementation and are not defined in the Service. Communities will develop common Types to characterize

CanonicalCourse Method Summary

void	addEquivalentCourse(Id canonicalCourseId)
void	addTopic(String topic)
CanonicalCourse	createCanonicalCourse(String title, String number, String description, Type courseType, Type courseStatusType, float credits)
CourseOffering	createCourseOffering(String title, String number, String description, Id term,Id Type offeringType, Type offeringStatusType, Type courseGradeType)
void	deleteCourseOffering(Id courseOfferingId)
CanonicalCourseIterator	getCanonicalCourses()
CanonicalCourseIterator	getCanonicalCoursesByType(Type courseType)
CourseOfferingIterator	getCourseOfferings()
CourseOfferingIterator	getCourseOfferingsByType(Type offeringType)
Type	getCourseType()
float	getCredits()
String	getDescription()
String	getDisplayName()
CanonicalCourseIterator	getEquivalentCourses()
Id	getId()
String	getNumber()
ProperitesIterator	getProperties()
Properties	getPropertiesByType(Type propertiesType)

[12] The intent is that credits are set at the CanonicalCourse level and not the CourseOffering or CourseSection levels. A single CanonicalCourse does not have different numbers of credits, rather each number of credits should be associated with in its own CanonicalCourse.

TypeIterator	getPropertyTypes()
Type	getStatus()
String	getTitle()
StringIterator	getTopics()
void	removeEquivalentCourse(Id canonicalCourseId)
void	removeTopic(String topic)
void	updateCredits(float credits)
void	updateDescription(String description)
void	updateDisplayName()
void	updateNumber(String number)
void	updateStatus(Type statusType)
void	updateTitle(String title)

org.osid.CourseManagement.CourseOffering

CourseOfferings are essentially CanonicalCourses associated with a specific Term. CanonicalCourses could be considered abstract while CourseOfferings are specific, concrete instances of CanonicalCourses in a particular Term[13]. The **getCanonicalCourse**() method returns the CourseOffering that created the CourseSection.

Note that there are people and roles associated with a CourseOffering, particularly as it is for a specific Term. Any Instructor, Teaching Assistant, Proctor, etc are defined in the Authorization OSID and are not covered in this OSID. For example, there might be an Authorization Function for Instructor with a Qualifier for the CourseOffering Id.

Students in the Offering who are managed with **addStudent**(), **changeStudent**(), **removeStudent**(), and **getRoster**() methods. Students are in both the Offering and a Section Roster. Students are added and changed with an EnrollmentStatusType. An example of these types might be “regular”, “auditing”, and “withdrawn with fail”. The Roster is enumerated through an EnrollmentRecordIterator that contains an Id and EnrollmentStatusType.

CourseOfferings have the same properties for title, description, number and Id as CanonicalCourse and CourseSection. CourseOfferings manage their CourseSections through the methods **createCourseSection**(), **deleteCourseSection**(), and **getCourseSections**(). As with CanonicalCourse, CourseOffering has a Type returned by getOfferingType().

There is a Status property associated with an Offering, perhaps whether it is open, closed, available for registration etc. CourseOfferings occur in a specific Term which is returned and updated by **getTerm**() and **updateTerm**(), respectively. The OfferingType and the Term are initialized to values set by the CanonicalCourse’s **createCourseOffering**() method. The getInCanonicalCourses() method enumerates the CanonicalCourse that created the CourseOffering and any equivalent CanonicalCourses.

[13] This terminology is not intended to reflect specific object oriented constructions.

The CourseOffering includes a CourseGradeType. This Type, returned by **getCourseGradeType**(), gives information about the summative grade for the CourseOffering. The CourseManagementManager is responsible for managing these grades.

CourseOfferings have a status that is identified through a Type.

CourseOfferings can have zero or more associated Assets. These Assets are referenced by Id and may reside in a Digital Repository or elsewhere.

CourseOffering Method Summary

void	addAsset(Id assetId)
void	addStudent(Id agentId, Type enrollmentStatusType)
void	ChangeStudent(Id agentId, Type enrollmentStatusType)
CourseSection	createCourseSection(String title, String number, String description, Type sectionType, Type sectionStatusType, Serializable location)
void	deleteCourseSection(Id courseSectionId)
IdIterator	getAssets()
CanonicalCourse	getCanonicalCourse()
CourseSectionIterator	getCourseSections()
CourseSectionIterator	getCourseSectionsByType(Type sectionType)
String	getDescription()
String	getDisplayName()
Type	getCourseGradeType()
Id	getId()
String	getNumber()
Type	getOfferingType()
EnrollmentRecordIterator	getRoster()
EnrollmentRecordIterator	getRosterByType(Type enrollmentStatusType)
PropertiesIterator	getProperties()
Properties	getPropertiesByType(Type propertiesType)
TypeIterator	getPropertyTypes()
Type	getStatus()
Term	getTerm()

String	getTitle()
void	removeAsset(Id assetId)
void	removeStudent(Id agentId)
void	updateStatus(Type statusType)
void	updateCourseGradeType(Type courseGradeType)
void	updateDescription(String description)
void	updateDisplayName(String displayName)
void	updateNumber(String number)
void	updateTitle(String title)

org.osid.CourseManagement.CourseSection

CourseSections are associated with specific CourseOfferings. It is the CourseSection that contains the Schedule for the Course, the location, and the Student Roster. The same CourseOffering may contain more than one Section[14].

The Schedule is a sequence of ScheduleItems. ScheduleIterms are defined in the Scheduling OSID. If a course is offered, say, Mondays, Wednesdays and Fridays at 2:00 PM throughout the Term, those ScheduleItems need to be set through an implementation process or explicitly through the **updateSchedule**() method. The **getSchedule**() method returns an enumeration of all the ScheduleItems for this Section. Note that there can be ScheduleItems for events associated with the Section separate from a recurring meeting time. An example is a mid-term exam. ScheduleItems are intended to be returned in chronological order.

The location for the Section is returned and updated by **getLocation**() and **updateLocation**(), respectively. The location is not a simple String, which would accommodate a building and room designation, but Serializable so that something else, for example a map, can be provided. Of course, a String is Serializable too.

As with the CourseOffering people may have specific roles that are defined in the Authorization OSID and are not covered here. An exception is the Students in the Section who are managed with **addStudent**(), **changeStudent**(), **removeStudent**(), and **getRoster**() methods. Students are added and changed with an EnrollmentStatusType. An example of these types might be “regular”, “auditing”, and “withdrawn with fail”. The Roster is enumerated through an EnrollmentRecordIterator that contains an Id and EnrollmentStatusType. Note that Students (Agents) are not created here, rather they are created in the Shared OSID.

CourseSections have the same properties for title, description, number and Id as CanonicalCourse and CourseOfferings. As with CourseOffering, CourseSection has a Type returned by **getSectionType**(). The SectionType is initialized to the value set by the CanonicalOffering’s **createCourseSection**() method. The **getCourseOffering**() method returns the CourseOffering that created the CourseSection.

Similar to the CourseOffering, the CourseSection can have a reference to an Asset and a status that is identified through a Type.

[14] Any algorithm for creating Sections or assigning Students to Sections is left to the implementation and not defined by this Service.

CourseSections can have zero or more associated Assets. These Assets are referenced by Id and may reside in a Digital Repository or elsewhere.

CourseSection Method Summary

void	addAsset(Id assetId)
void	addStudent(Id agentId, Type enrollmentStatusType)
void	changeStudent(Id agentId, Type enrollmentStatusType)
IdIterator	getAssets()
CourseOffering	getCourseOffering ()
String	getDescription()
String	getDisplayName()
Id	getId()
Serializable	getLocation()
String	getNumber()
PropertiesIterator	getProperties()
Properties	getPropertiesByType(Type propertiesType)
TypeIterator	getPropertyTypes()
EnrollmentRecordIterator	getRoster()
EnrollmentRecordIterator	getRosterByType(Type enrollmentStatusType)
ScheduleItemIterator	getSchedule()
Type	getSectionType()
Type	getStatus()
String	getTitle()
void	removeAsset(Id assetId)
void	removeStudent(Id agentId)
void	updateDescription(String description)
void	updateDisplayName(String displayName)
void	updateLocation(Serializable location)
void	updateNumber(String number)

void	updateSchedule(ScheduleItem[] scheduleItems)
void	updateStatus(Type statusType)
void	updateTitle(String title)

org.osid.CourseManagement.Term

The Term is a simple interface for capturing a Schedule. The Term has a unique Id so it can be referenced unambiguously and a Type so it can be characterized, for example as a Fall, Spring, or Summer Term. The Schedule can be something as simple as a start and end date, but can also include Items for holidays or other institution-wide events.

Term Method Summary

String	getDisplayName()
Id	getId()
ScheduleItemIterator	getSchedule()
Type	getType()
void	updateDisplayName(String displayName)

org.osid.CourseManagement.CourseGradeRecord

The CourseGradeRecord returns information about a Grade of a certain CourseGradeType and for a specific Agent and CourseOffering. The intent is to hold summative Grades as part of interaction with a student information system.

CourseGradeRecord Method Summary

Agent	getAgent()
Serializable	getCourseGrade()
Id	getCourseOffering()
String	getDisplayName()
Id	getId()
Type	getType()
void	updateCourseGrade(Serializable courseGrade)
void	UpdateDisplayName(String displayName)

org.osid.CourseManagement.EnrollmentRecord

The EnrollmentRecord holds the information for each Student in an Offering or Section. The Id is the agentId used when the student was added; the Status is the EnrollmentStatusType used when the student was added or changed.

EnrollmentRecord Method Summary

Type	getStatus()
Id	getStudent()

The CourseGroup manages the list of CanonicalCourses in a CourseGroup. CanonicalCourses are grouped for a variety of purposes, for example, prerequisites, corequisites, majors, minors, and sequences. The CourseGroupType will determine how to interpret membership and order in the CourseGroup.

osid.CourseManagement.CourseGroup

CourseGroup Method Summary

void	addCourse(Id canonicalCourseId)
CanonicalCourseIterator	getCourses()
String	getDisplayName()
Id	getId()
Type	getType()
void	removeCourse(Id canonicalCourseId)
void	updateDisplayName(String displayName)

Iterators

Iterators[15] return a set of elements of a specific type, one at a time. The purpose of Iterators is to offer a way for methods to return multiple values of a common type and not use an array. Returning an array may not be appropriate if the number of values returned is large or is fetched remotely. Iterators do not allow access to values by index, rather you must access values in sequence. Similarly, there is no way to go backwards through the sequence unless you place the values in a data structure, such as an array, that allows for access by index.

All iterators contain two methods. The **hasNext<*Object type*>**() method returns true if there are more values of the iterator type available; false otherwise. The **next<*Object type*>** () method returns the next element in the sequence. Note that in many cases, the order of elements is not guaranteed.

The following iterators are defined in this OSID:

- CanonicalCourseIterator

[15] Refer to “Discussion of Iterators” for more information.

- CourseGradeRecordIterator
- CourseGroupIterator
- CourseOfferingIterator
- CourseSectionIterator
- EnrollmentRecordIterator
- TermIterator

The following iterators are used in this OSID:

- TypeIterator (defined in Shared)
- StringIterator (define in Shared)
- ScheduleItemIterator (defined in Scheduling)
- AgentIterator (defined in Shared)

org.osid.coursemanagement.CourseManagementException

The OSIDs make use of Exceptions as a mechanism for responding to error or unusual conditions. All methods in the Shared OSID throw a **CourseManagementException**. The Exception contains a message that is a String. The following message Strings are defined in CourseManagementException:

- INVALID_CREDITS_VALUE
- UNKNOWN_TOPIC

If an implementation uses these messages, consumers of the implementation can easily test and conditionally respond to the Exception. Note that other kinds of Exception constructors are not used as all do or can devolve to a String. All methods of all interfaces of all OSIDs throw a subclass of org.osid.OsidException. This requires the caller of any implementation method handle the Exception.

If a method performs an operation without incident, an object or primitive may be returned, but in most cases, methods do not return error codes or a success or failure boolean. For example, a method that deletes an object with a particular identifier, would throw an Exception if the identifier were unknown; the method would not return, for example, false.